ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
Государственное образовательное учреждение высшего профессионального образования
«**ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**»

А.А. Дубаков, А.Е. Пинжин

ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ И ТЕЛЕКОММУНИКАЦИОННЫХ СИСТЕМ

Рекомендовано в качестве учебного пособия
Редакционно-издательским советом
Томского политехнического университета

Издательство
Томского политехнического университета
2009

УДК 621.321.001(075.8)
ББК 32.968я73
Д79

Дубаков А.А.

Д79 Проектирование информационных и телекоммуникационных систем: учебное пособие / А.А. Дубаков, А.Е. Пинжин. – Томск: Изд-во Томского политехнического университета, 2009. – 424 с.

Учебное пособие содержит описание жизненного цикла разработки информационных систем, методологий создания систем от классического метода «водопада» до успешно применяемых в настоящее время Agile-методологий. Рассматриваются теоретические и практические основы технологий создания распределенных приложений с применением открытой архитектуры Java Enterprise Edition и среды разработки Eclipse.

Пособие разработано в рамках реализации Инновационной образовательной программы ТПУ по направлению «Информационно-коммуникационные системы и технологии» и предназначено для магистрантов, обучающихся по направлениям «Информационные системы в экономике» и «Сети и телекоммуникационные системы».

УДК 621.321.001(075.8)
ББК 32.968я73

Рецензент

Доктор технических наук,
профессор Томского государственного университета

В.Т. Калайда

Единственное постоянство в индустрии информационных технологий – изменчивость.

Введение

В течение последних 40 лет информационные технологии продолжают совершенствовать мир и есть все основания полагать, что мы станем свидетелями технологической революции, которая изменит все аспекты жизни и бизнеса. Информация в современном мире превратилась в один из наиболее важных ресурсов, а информационные системы (ИС) стали необходимым инструментом практически во всех сферах деятельности и критическим фактором получения конкурентных преимуществ. ИС используются для обеспечения системы автоматизации предприятий и учреждений самого различного профиля (финансовых, промышленных, офисных) и самых различных размеров с разнообразными схемами иерархии, начиная от малых предприятий численностью несколько десятков человек и заканчивая крупными корпорациями численностью в десятки тысяч сотрудников. Также ИС используется на всех уровнях иерархии организаций, и предназначаются для решения задач как предприятия в целом (управление финансовыми ресурсами, управление запасами, планирование и производство, сбыт и снабжение, техническое обслуживание и ремонт оборудования, управление персоналом и т. п.), так и уровня его производственных подразделений, цехов и участков. Вряд ли удастся обнаружить организацию, которая не использует какую-либо систему информационной поддержки бизнес процессов в организации.

В настоящее время уже не стоит вопрос “надо или не надо автоматизировать” деятельность предприятий и организаций, на этот вопрос давно уже получен утвердительный ответ – это необходимо делать. Проблема формулируется в плоскости вариантов создания информационной системы, состава задач автоматизации и эффективного функционирования. Успешным решением создания информационной системы является компьютерная поддержка бизнес процессов, способствующая повышению эффективности функционирования организации и повышению уровня доходности.

Современные ИС характеризуются, прежде всего, высоким уровнем сложности разрабатываемых компонент, что накладывает опреде-

ленные требования на реализацию этапов жизненного цикла создания ИС, начиная от этапа анализа и завершая этапом сопровождения действующей ИС. Определяющую роль в реализации этапов жизненного цикла играет методология создания информационной системы, применяемые подходы, а также применяемые технологии создания приложений, реализующих ИС.

Сейчас наблюдается устойчивая тенденция роста заказов на разработку интегрированных распределенных информационных систем управления. Автоматизация отдельной функций, например, задач бухгалтерского учета или сбыта готовой продукции, считается уже пройденным этапом для многих предприятий и, кроме того, автоматизация отдельных задач не дает должного эффекта информатизации и не позволяет мобилизовать все информационные ресурсы организации для повышения эффективности деятельности и принятия решений.

Интегрированный подход к созданию информационных систем основан на концепции распределенных приложений, которые, в свою очередь, предполагают телекоммуникационную обработку. Телеобработка данных в части создания интегрированных информационных систем должна обеспечивать коллективное использование ресурсов систем обработки данных удаленными пользователями с возможностью организации межмашинного обмена. В соответствии со стандартом 24402-88 телеобработка предполагает рассмотрение различных компонентов аппаратной, программной и технологической архитектуры.

В рамках настоящего курса наряду с рассмотрением методологических основ проектирования информационных систем особое внимание уделяется программным и технологическим компонентам создания распределенных систем обработки данных.

Методологические основы создания ИС

Основные понятия информационной системы

Под информационной системой понимается объединение людей, данных, процессов, интерфейсов, сетей и информационных технологий, которые взаимодействуют для целей поддержки и улучшения рутинных операций, а также для обеспечения потребностей подготовки и принятия решений руководства и пользователей. Информационная система реализуется на основе приложений, которые представляют собой основанное на компьютере решение одной или более бизнес-проблем или потребностей в обработке данных. ИС, как правило, состоит из одного или нескольких приложений.

Информационная технология – обозначает современный термин, описывающий комбинацию компьютерной технологии (аппаратного и программного обеспечения) с телекоммуникационной технологией (передача данных и изображений, голосовых сетей).

Проектирование информационных и телекоммуникационных систем – это дисциплина, определяющая подсистемы, компоненты и способы их соединения, задающая ограничения, при которых система должна функционировать, выбирающая наиболее эффективное сочетание людей, аппаратного и программного обеспечения для реализации системы.

Основная цель разработки ИС состоит в разработке качественного программно-технического решения в рамках выделенного времени и бюджета, удовлетворяющая реальным потребностям заказчиков.

Концепции создания информационной системы

Создание информационной системы представляет собой сложную задачу, и естественно, должна регламентироваться некоторой методологией, которая, в свою очередь, состоит в регламентации процесса проектирования ИС и обеспечении управления этим процессом с тем, чтобы гарантировать выполнение требований, как к самой ИС, так и к характеристикам процесса разработки. Задачами используемой методологии являются:

1. Обеспечивать создание корпоративных ИС, отвечающих целям и задачам организации, а также предъявляемым требованиям по автоматизации деловых процессов заказчика;
2. Гарантировать создание системы с заданным качеством в заданные сроки и в рамках установленного бюджета проекта;
3. Поддерживать удобную дисциплину сопровождения, модификации и развития системы;
4. Обеспечивать преемственность разработки, т. е. использование в разрабатываемой ИС существующей информационной инфраструктуры организации (задела в области информационных технологий).

Современные крупномасштабные проекты ПО характеризуются, как правило, следующими особенностями:

Характеристики объекта внедрения:

- структурная сложность (многоуровневая иерархическая структура организации) и территориальная распределенность;
- функциональная сложность (многоуровневая иерархия и большое количество функций, выполняемых организацией; сложные взаимосвязи между ними);

- информационная сложность (большое количество источников и потребителей информации (министерства и ведомства, местные органы власти, организации-партнеры), разнообразные формы и форматы представления информации, сложная информационная модель объекта – большое количество информационных сущностей и сложные взаимосвязи между ними), сложная технология прохождения документов;
- сложная динамика поведения, обусловленная высокой изменчивостью внешней среды (изменения в законодательных и нормативных актах, нестабильность экономики и политики) и внутренней среды (структурные реорганизации, текучесть кадров).

Технические характеристики проектов создания ПО:

- различная степень унифицированности проектных решений в рамках одного проекта;
- высокая техническая сложность, определяемая наличием совокупности тесно взаимодействующих компонентов (подсистем), имеющих свои локальные задачи и цели функционирования (транзакционных приложений, предъявляющих повышенные требования к надежности, безопасности и производительности, и приложений аналитической обработки (систем поддержки принятия решений), использующих нерегламентированные запросы к данным большого объема);
- отсутствие полных аналогов, ограничивающее возможность использования каких-либо типовых проектных решений и прикладных систем, высокая доля вновь разрабатываемого ПО;
- большое количество и высокая стоимость унаследованных приложений (существующего прикладного ПО), функционирующих в различной среде (персональные компьютеры, миникомпьютеры, мэйнфреймы), необходимость интеграции унаследованных и вновь разрабатываемых приложений;
- большое количество локальных объектов внедрения, территориально распределенная и неоднородная среда функционирования (СУБД, операционные системы, аппаратные платформы);
- большое количество внешних взаимодействующих систем различных организаций с различными форматами обмена информацией (налоговая служба, налоговая полиция, Госстандарт,

Госкомстат, Министерство финансов, МВД, местная администрация).

Организационные характеристики проектов создания ПО:

- различные формы организации и управления проектом: централизованно управляемая разработка тиражируемого ПО, экспериментальные пилотные проекты, инициативные разработки, проекты с участием, как собственных разработчиков, так и сторонних компаний на контрактной основе;
- большое количество участников проекта как со стороны заказчиков (с разнородными требованиями), так и со стороны разработчиков (более 100 человек), разобщенность и разнородность отдельных групп разработчиков по уровню квалификации, сложившимся традициям и опыту использования тех или иных инструментальных средств;
- значительная длительность жизненного цикла системы, в том числе значительная временная протяженность проекта, обусловленная масштабами организации-заказчика, различной степенью готовности отдельных ее подразделений к внедрению ПО и нестабильностью финансирования проекта;
- высокие требования со стороны заказчика к уровню технологической зрелости организаций-разработчиков (наличие сертификации в соответствии с международными и отечественными стандартами).

Проблемы создания информационной системы

С конца 60-х годов прошлого века до сегодняшних дней продолжается так называемый *«кризис ПО»*. Выражается он в том, что большие проекты выполняются с превышением сметы расходов и/или сроков отведенных на разработку, а разработанное ПО не обладает требуемыми функциональными возможностями, имеет низкую производительность и качество. По результатам исследований американской индустрии разработки ПО, выполненных в 1995 году Standish Group (www.standishgroup.com), только 16 % проектов завершились в срок, не превысили запланированный бюджет и реализовали все требуемые функции и возможности. 53 % проектов завершились с опозданием, расходы превысили запланированный бюджет, требуемые функции не были реализованы в полном объеме. 31 % проектов были аннулированы до завершения. Для двух последних категорий проектов бюджет среднего проекта оказался превышенным на 89 %, а срок выполнения – на

122 %. В последние годы процентное соотношение трех перечисленных категорий проектов незначительно изменяется в лучшую сторону.

1995	1998	2000	2004
31 % аннулируются до завершения	28 %	23 %	18 %
53 % не укладываются в поставленные сроки, превышают запланированные расходы и не реализуют в полном объеме требуемые функции	46 %	49 %	53 %
16 % завершаются в срок	26 %	28 %	29 %

Причинами неудач являются:

- нечеткая и неполная формулировка требований;
- недостаточное вовлечение пользователей в работу над проектом;
- отсутствие необходимых ресурсов;
- неудовлетворительное планирование и отсутствие грамотного управления проектом;
- частое изменение требований и спецификаций;
- новизна и несовершенство используемой технологии;
- недостаточная поддержка со стороны высшего руководства;
- недостаточно высокая квалификация разработчиков, отсутствие необходимого опыта.

При планировании проектов зачастую по тем или иным причинам устанавливаются невыполнимые сроки, закладываются недостаточные ресурсы. Таким образом, возникают *безнадежные проекты*. Признаки безнадежного проекта:

- план проекта сжат более чем наполовину по сравнению с нормальным расчетным планом;
- количество разработчиков уменьшено более чем наполовину по сравнению с действительно необходимым для проекта данного размера и масштаба;
- бюджет и связанные с ним ресурсы урезаны наполовину;
- требования к функциям, производительности и другим характеристикам вдвое превышают значения, которые они могли бы иметь в нормальных условиях.

Еще одной причиной неверного планирования является заблуждение относительно производительности проектировщиков. В большом проекте общая производительность группы разработчиков не равна сумме производительностей отдельных членов группы (посчитанной как если бы они работали в одиночку). Об этом в книге «Мифический человеко-месяц» пишет Фредерик Брукс. Выводы Брукса:

- самая частая причина провала – нехватка времени;
- иногда работы нельзя ускорить, не испортив результат;
- человеко-месяц – опасное заблуждение, поскольку предполагается, что количество людей и месяцы можно поменять местами;
- разделение задачи между несколькими людьми вызывает накладные затраты;
- если проект не укладывается в срок, то добавление людей задержит его еще больше;
- «серебряной пули» нет!

Последнее положение касается технологии разработки. Брукс утверждает, что технологии, позволяющей на порядок повысить производительность разработчиков, не существует. То есть, нельзя полагать, что какая-либо новейшая технология разработки позволит осуществить проект в 10 раз быстрее.

Особенности современных проектов создания ПО:

- сложность – неотъемлемая характеристика создаваемого ПО;
- отсутствие полных аналогов и высокая доля вновь разрабатываемого ПО;
- наличие унаследованного ПО и необходимость его интеграции с разрабатываемым ПО;
- территориально распределенная и неоднородная среда функционирования;
- большое количество участников проектирования, разобщенность и разнородность отдельных групп разработчиков по уровню квалификации и опыту.

Разработка ПО имеет следующие специфические особенности:

- неформальный характер требований к ПО и формализованный основной объект разработки – программы;
- творческий характер разработки;
- дуализм ПО, которое, с одной стороны, является статическим объектом – совокупностью текстов, с другой стороны, – динамическим, поскольку при эксплуатации порождаются процессы обработки данных;
- при своем использовании (эксплуатации) ПО не расходуется и не изнашивается;
- «неощутимость», «воздушность» ПО, что подталкивает к безответственному переделыванию, поскольку легко стереть и переписать, чего не сделаешь при проектировании зданий и аппаратуры.

Выходом из кризиса ПО стало создание программной инженерии (software engineering). *Инженерия ПО* (software engineering- SE) – сово-

купность инженерных методов и средств создания ПО. *Фундаментальная идея программной инженерии: проектирование ПО является формальным процессом, который можно изучать и совершенствовать.*

Освоение и правильное применение методов и средств программной инженерии позволяет повысить качество, обеспечить управляемость процесса проектирования.

Этапы становления и развития SE:

- 70-е и 80-е годы – систематизация и стандартизация процессов создания ПО (структурный подход)
- 90-е годы – начало перехода к сборочному, индустриальному способу создания ПО (объектно-ориентированный подход)

Программная инженерия применяется для удовлетворения требований заказчика ПО. *Основными целями программной инженерии являются*:

- Системы должны создаваться в короткие сроки и соответствовать требованиям заказчика на момент внедрения.
- Качество ПО должно быть высоким.
- Разработка ПО должна быть осуществлена в рамках выделенного бюджета.
- Системы должны работать на оборудовании заказчика, а также взаимодействовать с имеющимся ПО.
- Системы должны легко сопровождаться и масштабироваться.

Вопросы для самопроверки

1. Дайте определение информационной системе.
2. Что является задачами использования методологии при разработке ИС?
3. Дайте характеристику объекта внедрения крупномасштабного проекта создания ИС.
4. Дайте техническую характеристику объекта внедрения крупномасштабного проекта создания ИС.
5. Дайте организационную характеристику объекта внедрения крупномасштабного проекта создания ИС.
6. Перечислите причины неудач проектов создания ИС.
7. Каковы организационные признаки безнадежности проекта создания ИС?
8. В чем состоят особенности современных проектов создания ИС?
9. В чем состоит фундаментальная идея программной инженерии и ее основные цели?

Содержание этапов жизненного цикла

Стандарт ISO/IEC 12207

Все методологии разработки информационных систем основываются на модели жизненного цикла разработки системы. Основной нормативный документ, регламентирующий ЖЦ ПО – стандарт ISO/IEC 12207: 1995 “Information Technology – Software Life Cycle Processes” (ГОСТ Р ИСО/МЭК 12207-99). В рамках технологий создания ПО понятие ЖЦ уточняется, но указанные стандарты не нарушаются.

С точки зрения статической структуры ЖЦ является совокупностью процессов ЖЦ.

Процесс ЖЦ – набор взаимосвязанных действий, преобразующих некоторые входные данные и ресурсы в выходные.

Каждый процесс характеризуется задачами, методами их решения, действующими лицами, результатами. Процессы ЖЦ протекают параллельно. Каждый процесс разделен на набор действий, каждое действие – на набор задач. Каждый процесс, действие или задача инициируется и выполняется по мере необходимости, причем не существует заранее определенных последовательностей выполнения.

- основные (приобретение, поставка, разработка, эксплуатация, сопровождение);
- вспомогательные (документирование, управление конфигурацией, обеспечение качества, верификация, аттестация, совместная оценка, аудит, разрешение проблем);
- организационные (управление, создание инфраструктуры, усовершенствование, обучение).

Приведем содержание всех процессов ЖЦ.

Процесс приобретения включает следующие действия: инициирование приобретения; подготовку заявочных предложений; подготовку и корректировку договора; надзор за деятельностью поставщика; приемку и завершение работ. Действующие лица: заказчик, поставщик. Задачи приобретения: определение заказчиком своих потребностей в ПО; анализ требований к ПО; принятие решения о приобретении ПО; выработка плана приобретения и заявочных предложений; выбор поставщика; подготовка и заключение договора с поставщиком; контроль соблюдения условий договора; корректировка договора при необходимости.

Процесс поставки включает в себя следующие действия: инициирование поставки; подготовку ответа на заявочные предложения; подготовку договора; планирование и выполнение поставки; контроль поставки; проверку и оценку. Действующие лица: заказчик, поставщик. Задачи поставки: оценка поставщиком заявочных предложений; подго-

товки и заключение договора с заказчиком, контроль со стороны поставщика за соблюдением условий договора, принятие решения о привлечении субподрядчика или выполнении работ своими силами, выработка плана управления проектом и др.

Процесс разработки включает в себя следующие действия: подготовительную работу; анализ требований к ПО; проектирование архитектуры ПО; детальное проектирование ПО; кодирование ПО; тестирование ПО; интеграцию ПО; установку ПО; приемку ПО. Действующие лица: разработчик, заказчик. Задачи разработки: выбор модели ЖЦ ПО и согласование с заказчиком; определение требований к ПО (функциональных и нефункциональных); определение состава компонентов ПО и создание документации по каждому компоненту; моделирование и спецификация компонент ПО; планирование интеграции компонент; создание исходных текстов компонент; поиск и исправление ошибок в исходных текстах и документации; сборка ПО; развертывание ПО; оценка результатов.

Процесс эксплуатации включает в себя следующие действия: подготовительную работу; эксплуатационное тестирование; эксплуатацию; поддержку пользователей. Действующие лица: оператор (организация, эксплуатирующая ПО), пользователи. Задачи эксплуатации: выработка плана эксплуатации и эксплуатационных стандартов; составление процедур локализации и разрешения проблем эксплуатации; поиск ошибок в ПО перед вводом в эксплуатацию его новых версий; оказание помощи пользователям и консультирование.

Процесс сопровождения включает в себя следующие действия: подготовительную работу; анализ проблем и запросов на модификацию ПО; проверку и приемку; перенос ПО в другую среду; снятие ПО с эксплуатации. Действующие лица: служба сопровождения, пользователи. Задачи сопровождения: выработка плана сопровождения; составление процедур локализации и разрешения проблем сопровождения; оценка целесообразности внесения модификаций в ПО; принятие решения о модификации; поиск ошибок в ПО после его модификации; проверка целостности ПО; архивирование при снятии с эксплуатации; обучение пользователей.

Процесс документирования включает в себя следующие действия: подготовительную работу; проектирование и разработку документации; выпуск документации; сопровождение.

Процесс управления конфигурацией в себя следующие действия: подготовительную работу; создание базы знаний о ПО (конфигурации); контроль конфигураций; учет состояния конфигурации; оценку конфигурации; управление выпуском и поставку ПО. *Конфигурация ПО* – это

совокупность сведений о его функциональных и физических характеристиках на всех стадиях ЖЦ ПО. Основная задача управления конфигурацией: организация, систематический учет и контроль внесения изменений в ПО.

Процесс обеспечения качества включает в себя следующие действия: подготовительную работу; обеспечение качества продукта; обеспечение качества процесса; обеспечение других показателей качества ПО. Задачи обеспечения качества: гарантированное соответствие ПО требованиям заказчика, зафиксированным в договоре; гарантированнее соответствие процессов ЖЦ ПО, методов разработки, квалификации персонала установленным стандартам.

Процесс верификации включает в себя следующие действия: подготовительную работу; верификацию. Основная задача верификации – проверка соответствия разработанных программ в составе ПО их спецификациям.

Процесс аттестации состоит в определении полноты соответствия разработанного ПО требованиям заказчика. Основная задача аттестации – оценка достоверности тестирования ПО. Как правило, для аттестации привлекают независимых экспертов.

Процесс совместной оценки включает в себя следующие действия: подготовительную работу; оценку управления проектом; техническую оценку. Основная задача совместной оценки – контроль планирования и управления ресурсами, персоналом, инфраструктурой проекта.

Процесс аудита состоит в определении полноты соответствия проекта условиям договора.

Процесс разрешения проблем предусматривает анализ и разрешение проблем, возникающих в течение ЖЦ ПО.

Процесс управления включает в себя следующие действия: подготовительную работу; планирование; выполнение и контроль; проверку и оценку; завершение. Задачи управления: проверка достаточности имеющихся ресурсов; составление графиков работ; оценка затрат; выделение ресурсов; распределение ответственности; оценка рисков.

Процесс создания инфраструктуры состоит в выборе и поддержке технологии разработки ПО, стандартов и инструментальных средств; выборе и установке аппаратных и программных средств, необходимых для разработки, эксплуатации и сопровождения ПО.

Процесс усовершенствования предусматривает оценку, измерение, контроль и усовершенствование процессов ЖЦ ПО. Основная задача усовершенствования – повышение производительности труда.

Процесс обучения включает в себя следующие действия: подготовительную работу; разработку учебных планов, курсов, материалов;

реализацию планов обучения. Задачи обучения: первоначальное обучение персонала; повышение квалификации персонала.

Все процессы ЖЦ ПО взаимосвязаны и их динамику, т. е. развитие ЖЦ во времени, определяет *модель жизненного цикла*.

Модель ЖЦ ПО – это структура, определяющая последовательность выполнения и взаимосвязи процессов, действий и задач на протяжении всего ЖЦ и рассматривается как совокупность стадий ЖЦ.

Стадия ЖЦ – это часть ЖЦ ограниченная временными рамками, по завершении которой достигается определенный важный результат в соответствии с требованиями для данной стадии ЖЦ.

Модели жизненного цикла разработки системы

- каскадная (метод водопада);
- эволюционная;
- основанная на формальных преобразованиях;
- итерационные (пошаговая и спиральная).

Каскадная (метод водопада)

Принципы *каскадной модели*: фиксация требований к системе в начале проекта; переход от стадии к стадии только после полного завершения работ на текущей стадии; недопустимость возврата на пройденные стадии; жесткая привязка процессов ЖЦ к стадиям ЖЦ (рис. 1).

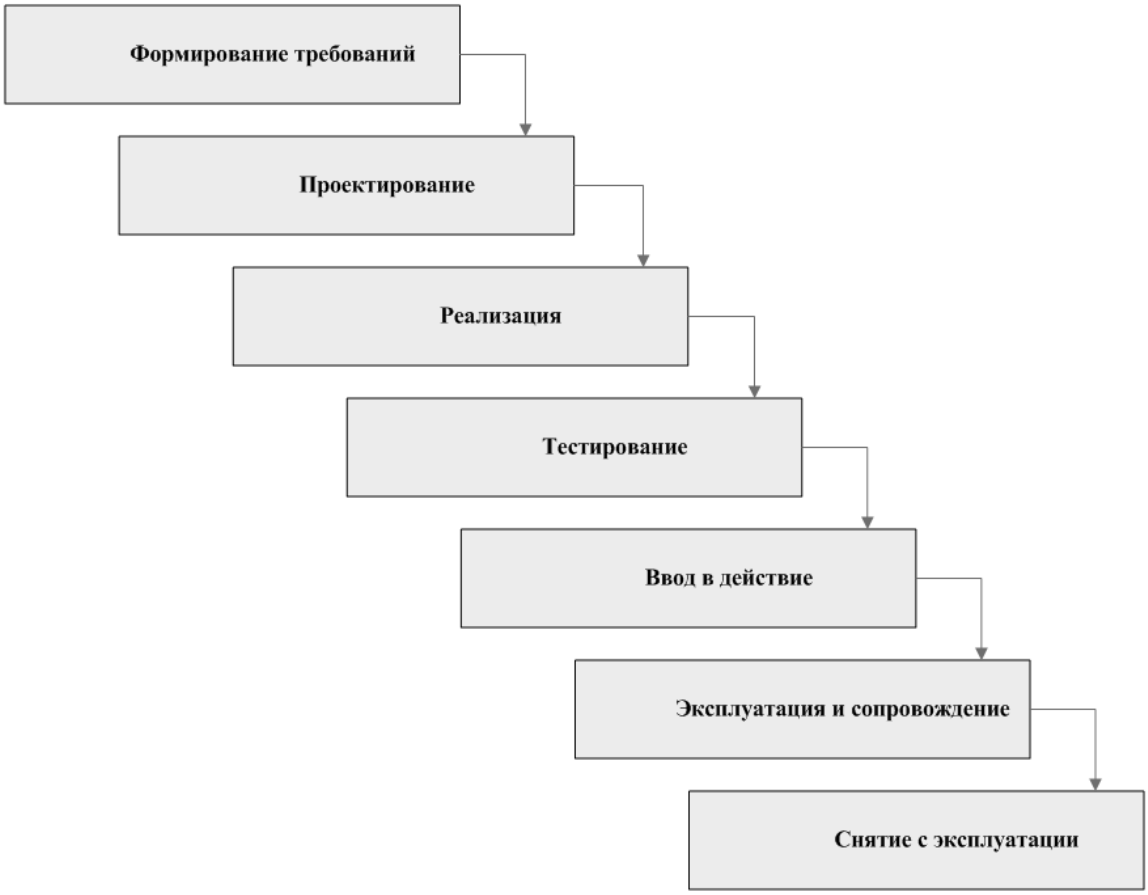


Рис. 1. Каскадная модель (метод водопада)

Стадия *формирования требований* включает процессы, приводящие к созданию документа, описывающего поведение ПО с точки зрения внешнего по отношению к нему наблюдателя с фиксацией требований относительно его качества.

Проектирование включает следующие процессы: разработку архитектуры ПО, разработку структур программ в его составе и их детальную спецификацию.

Реализация или кодирование включает процессы создания программного кода на языках программирования.

На этапе *тестирования* производится собственно тестирование, а также отладка и оценка качества ПО.

Ввод в действие – это развертывание ПО на соответствующей архитектуре вычислительной системы, обучение пользователей и т. п.

Эксплуатация ПО – это использование ПО для решения практических задач на компьютере путем выполнения ее программ.

Сопровождение ПО – это процесс сбора информации о качестве ПО в эксплуатации, устранения обнаруженных в нем ошибок, его дора-

ботки и модификации, а также извещения пользователей о внесенных в него изменениях.

Достоинства: на каждой стадии формируется законченный набор проектной документации, отвечающий критериям полноты и согласованности; выполняемые в логичной последовательности стадии работ облегчают планирование сроков завершения всех работ и соответствующих затрат. **Недостатки:** позднее обнаружение проблем; выход из календарного графика, запаздывание с получением результатов; высокий риск создания системы, не удовлетворяющей изменившимся потребностям пользователей; избыточность документации; неравномерная нагрузка членов группы, работающей над проектом в ходе ЖЦ.

На самом деле невозможно двигаться строго поступательно, возможны возвраты на предыдущие этапы для исправления ошибок и недоделок, сделанных на ранних стадиях, учета меняющиеся в ходе проекта требований. В этом кроется причина недостатков водопадной модели.

Эволюционная модель

Особенности **эволюционной модели,** представленной на рис. 2: поэтапное уточнение требований к ПО с помощью прототипирования; параллельное осуществление анализа требований, разработки и верификации. Достоинства: полный учет требований заказчика, большее его участие в проекте; равномерная нагрузка на группу; раннее обнаружение проблем и их разрешение по мере возникновения. Недостатки: плохая документированность; запутанность создаваемого ПС и сложность внесения изменений; сложность планирования; необходимость специальных средств и технологий разработки ПС; годится лишь для небольших ПС.

Рис. 2. Эволюционная модель создания ПО

Модели ЖЦ, основанной на формальных преобразованиях

Особенности *модели ЖЦ, основанной на формальных преобразованиях,* представленной на рис. 3: использование специальных нотаций для формального описания требований; кодирование и тестирование заменяется процессом предобразования формальной спецификации в исполняемую программу. Достоинства: формальные методы гарантируют соответствие ПО спецификациям требований к ПО, т. о. вопросы надежности и безопасности решаются автоматически. Недостатки: большие системы сложно описать формальными спецификациями; требуются специально подготовленные и высококвалифицированные разработчики; есть зависимость от средств разработки и нотации спецификаций.

Рис. 3. Модель ЖЦ, основанная на формальных преобразованиях

Итерационные модели

Особенности итерационных моделей (рис.4):

- процесс разработки разбивается на последовательность шагов, выполняемых циклически;
- модель напоминает несколько последовательных «каскадов»;
- разные виды деятельности не привязаны намертво к определенным этапам разработки, а выполняются по мере необходимости, иногда повторяются, до тех пор, пока не будет получен нужный результат;
- с каждой пройденной итерацией ПО наращивается, в него интегрируются новые разработанные компоненты.

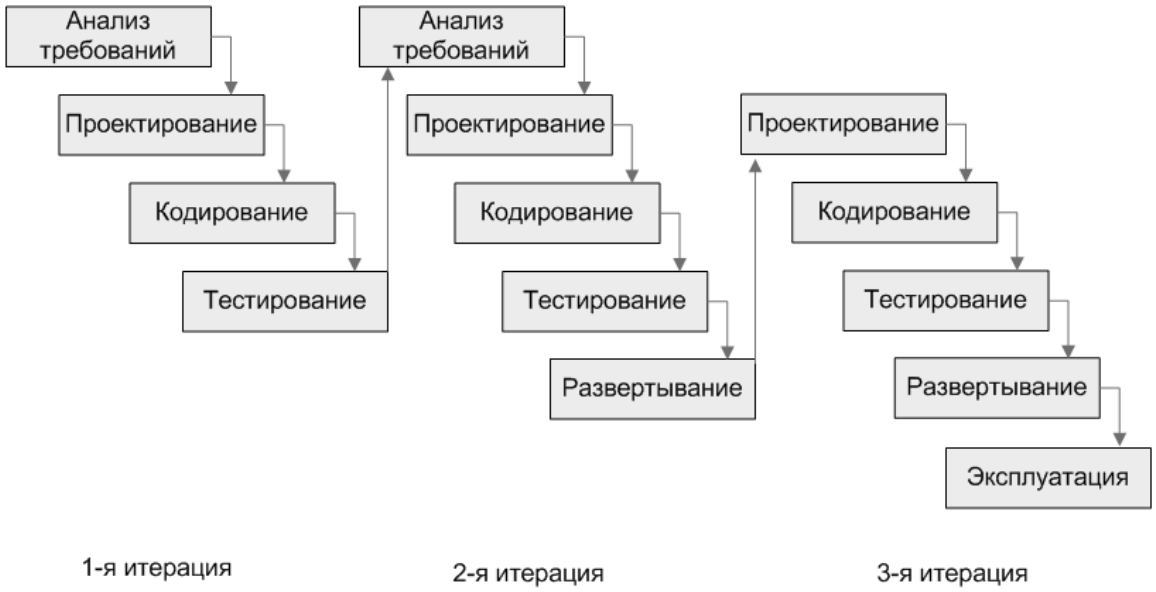


Рис. 4. Схема пошаговой итерационной модели ЖЦ

Спиральные модели

Особенности спиральной модели, представленной на рис. 5:

- общая структура действий на каждой итерации – планирование, определение задач, ограничений и вариантов решений, оценка предложенных решений и рисков, выполнение основных работ итерации и оценка их результатов;
- решение о начале новой итерации принимается на основе результатов предыдущей;
- досрочное прекращение проекта в случае обнаружения его нецелесообразности.

Достоинства итерационных моделей:

- полный учет требований заказчика, большее его участие в проекте;
- равномерная нагрузка на группу;
- раннее обнаружение проблем и их разрешение по мере возникновения, уменьшение рисков на каждой итерации.

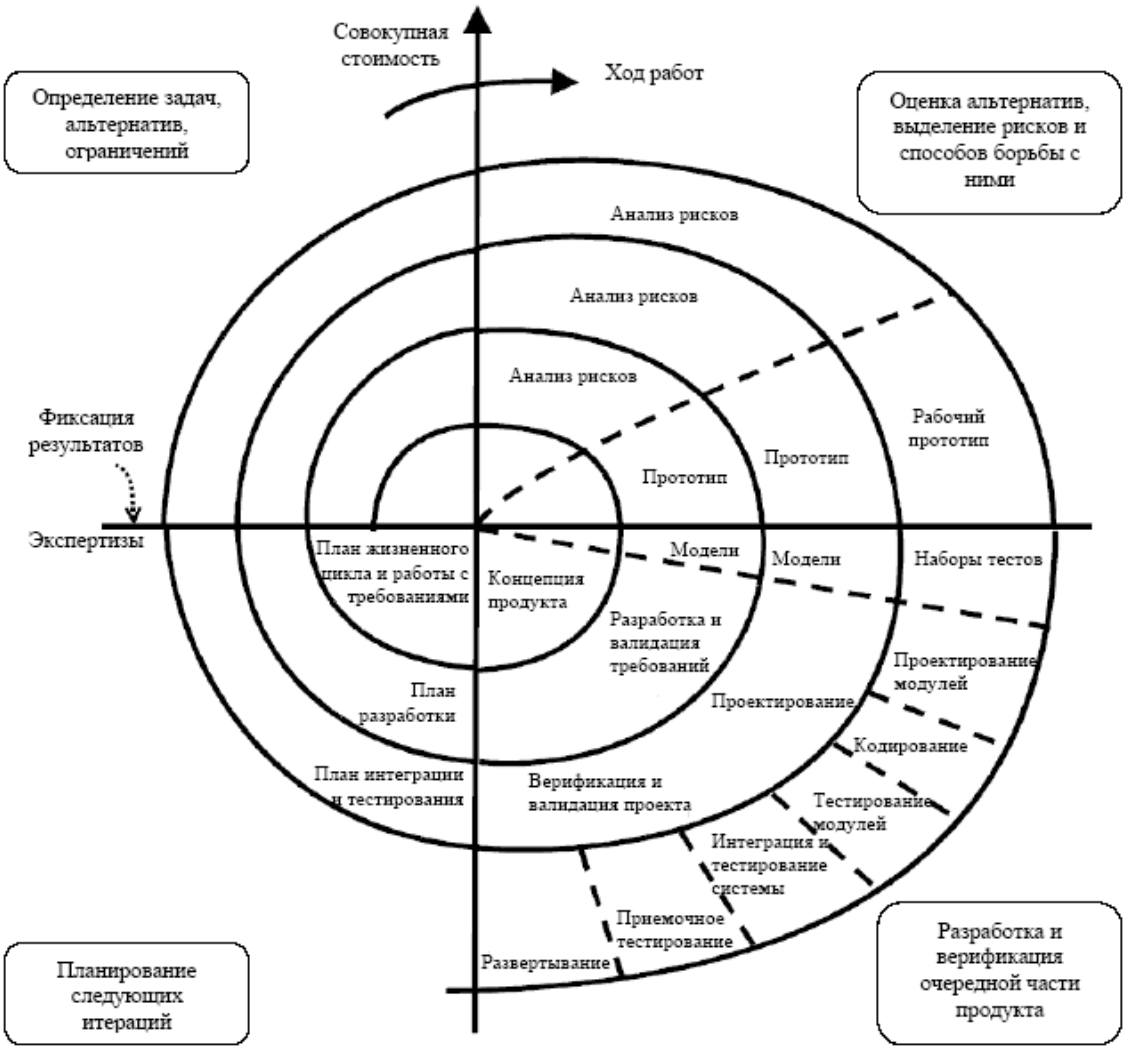


Рис. 5. Схема спиральной модели ЖЦ

Недостатки итерационных моделей: сложность планирования; плохая документированность создаваемого ПО.

Проблемой современной программной инженерии являются «тяжелые» процессы. Чаще всего их обвиняют в бюрократизме – чтобы следовать такой методологии, нужно выполнять так много различных предписаний, что замедляется весь темп работ, потому их и называют тяжеловесными или, иногда, монументальными. Характеристики *«тяжелого» процесса*:

1. необходимость документировать каждое действие разработчиков;
2. множество рабочих продуктов (в первую очередь – документов), создаваемых в бюрократической атмосфере;
3. отсутствие гибкости;
4. детерминированность (долгосрочное детальное планирование и предсказуемость всех видов деятельности, распределение челове-

ческих ресурсов на длительный срок, охватывающий большую часть проекта).

Противоположностью «тяжелого» процесса является «*легковесный» процесс* – основа быстрой разработки ПО (agile software development). Быстрая разработка ориентируется на эффективную коммуникацию между разработчиками, высокую их квалификацию и другие факторы, позволяющие сократить расходы на «бюрократию». Принципы быстрой разработки:

1. Диалог «лицом к лицу» – самый эффективный способ обмена информацией.
2. Избыточная «тяжесть» технологии (дополнительные рабочие продукты, планы, диаграммы, документы) стоит дорого.
3. Более многочисленные команды требуют более «тяжелых» технологий.
4. Большая «тяжесть» процесса подходит для проектов с большей критичностью.
5. Возрастание обратной связи и коммуникации сокращает потребность в промежуточных продуктах.
6. Дисциплина, умение и понимание противостоят процессу, формальности и документированию.
7. Потеря эффективности в некритических видах деятельности вполне допустима.

Под критичностью понимаются масштабы последствий отказа разрабатываемого ПО. Уровни критичности:

- потеря удобства;
- потеря важных данных и/или рабочего времени;
- потеря невозместимых средств, дорогостоящего оборудования;
- потеря человеческой жизни.

Основные направления развития современной программной инженерии:

1) Визуальное моделирование ПО
2) Управление требованиями
3) Управление конфигурацией и изменениями
4) Управление качеством ПО
5) Итерационная разработка ПО
6) Использование компонентной архитектуры (объектно-ориентированный подход)

Современные тенденции в программной инженерии

Как уже отмечалось, проблемой современной программной инженерии являются «тяжелые» процессы и, в этой связи, в последнее время

широкое признание и распространение получили современные, так называемые, гибкие (agile) методологии. Вообще, можно констатировать, что индустрия информационных технологий переживает фазу перехода от тяжеловесных (регламентных) технологий создания программных продуктов к гибким методологиям. До недавнего времени руководителей подразделений ИТ возмущало нежелание разработчиков следовать инструкциям в 1000 страниц в процессе разработки программных систем. Вместе с гибкими технологиями, такими как экстремальное программирование (extreme programming (XP), разработка, основанная на потребностях в функциональности заказчиков – feature-driven development (FDD), гибкое моделирование (agile modeling) появилась возможность значительно сократить элементы бюрократических подходов к созданию программных систем.

Современные методологии гибкой разработки

Agile Software Development Alliance (http://www.agilealliance.org), образованный в феврале 2001 года, сделал вывод, о том, что для достижения цели в разработке программных продуктов необходимо сосредоточиться на проблемах людей и технологиях разработки, существенно поддерживающих изменения. Слово agile (быстрый, ловкий, стремительный) отражает в целом их подход к разработке ПО, основанный на богатом опыте участия в разнообразных проектах в течение многих лет. Фактически был разработан Манифест, определяющий четыре ценности, способствующие лучшему методу создания программных систем:

1. Личности и взаимодействия доминируют над процессами и инструментальными средствами. Наиболее важные факторы, которые необходимо рассматривать являются люди и как они работают совместно, поскольку если вы не мобилизуете этот фактор, то лучшие инструментальные средства и процессы не найдут своего применения.
2. Работающее программное обеспечение доминирует над всеобъемлющей документацией. Первичная цель разработки ПО – создание этого ПО, а не документации. Документация имеет свое место; хорошо написанная документация является ценным руководством для понимания людьми, как и почему система построена и как с ней работать.
3. Совместная работа с заказчиком преобладает над обсуждением контракта. Только ваш заказчик может выразить, что им необходимо. Они, конечно, не имеют знаний для точной спецификации системы, они, вероятно, не смогут достичь понимания с

первого раза, но они смогут менять свои представления. Работа с заказчиком тяжела, но это реальная работа.

4. Реагирование на изменения преобладает над следованием плану. Изменения являются реальностью разработки ПО, реальность, которую должно отражать ваше ПО. Люди изменяют свои приоритеты по различным причинам, их представления о предметной области изменяются, когда они наблюдают результаты вашей работы, изменяется среда бизнеса, также как и технологии. Хотя вам необходим план проекта, он должен быть податливый и он может быть очень простым, во многом отличный от графика Ганта.

Манифест определяет предпочтения, а не альтернативы, поощряя концентрацию на определенные области, но, не устраняя другие.

Кроме того, Agile Allianse предлагает набор из 12 принципов, основанных на описанных ранее четырех ценностях:

1. Высший приоритет состоит в удовлетворении заказчика посредством своевременной и непрерывной поставки полезного ПО.

2. Приветствовать изменения требований, даже если это приводит к задержкам в разработке. Методика задействует изменения для конкурентных преимуществ заказчика.

3. Чаще выдавайте работающее ПО, от нескольких недель до нескольких месяцев

4. Представители бизнеса должны ежедневно работать совместно над проектом

5. Строить проект следует в окружении мотивированных личностей. Обеспечьте им среду и требуемую поддержку, доверяйте им в выполнении работы.

6. Наиболее результативный и эффективный метод подачи информации для и внутри команды разработчиков является беседа в виде диалога.

7. Работающее ПО является основной единицей изменения прогресса.

8. Гибкие процессы способствуют устойчивой разработке. Спонсоры, разработчики и пользователи должны поддерживать постоянные шаги беспредельно.

9. Постоянное внимание техническому совершенству и хорошее проектирование улучшают гибкость.

10. Существенной является простота, искусство максимизации количества не сделанной работы.

11. Лучшие архитектуры, требования и проекты появляются от самоорганизующихся команд.

12. В определенный момент команда разработчиков рассматривает варианты организации более эффективной работы и затем соответствующим образом настраивает и регулирует свое поведение.

Эти принципы формируют фундамент здравого смысла, на основе которого может строиться успешная разработка ПО.

При этом следует четко понимать: при всех достоинствах быстрой разработки ПО этот подход не является универсальным и применим только в проектах определенного класса. Быстрая разработка ПО применима только в проектах малого и среднего масштаба с низкой критичностью.

Extreme Programming – XP

Одним из наиболее известных примеров практической реализации подхода быстрой разработки ПО является "Экстремальное программирование" (Extreme Programming – XP). Этот метод предназначен для небольших компактных команд, нацеленных на получение как можно более высокого качества и продуктивности, и достигает этого посредством насыщенной, неформальной коммуникации, придания на персональном уровне особого значения умению и навыкам, дисциплине и пониманию, сводя к минимуму все промежуточные рабочие продукты. Иногда само понятие гибкие методологии явно или неявно отождествляют с XP. На русском языке издано уже несколько книг, посвященных этой методологии. XP проповедует коммуникабельность, простоту, обратную связь и отвагу. Методология описывается как 12 практик: игра в планирование, короткие релизы, метафоры, простой дизайн, переработки кода (refactoring), разработка "тестами вперед", парное программирование, коллективное владение кодом, 40-часовая рабочая неделя, постоянное присутствие заказчика и стандарты кода. Огромная роль в методологии отводится тестированию. Все прочие процессы тоже упоминают тестирование, но делают это как-то поверхностно. Что касается XP, то в нем тестирование является той основой, на которой строится разработка. При этом каждый программист пишет тесты одновременно с кодом разрабатываемой системы. Эти тесты используются при постоянной интеграции и в процессе сборки системы, что дает стабильный фундамент для дальнейшей работы. На этом фундаменте XP строит эволюционный процесс проектирования, основанный на реорганизации кода системы в течение каждой последующей итерации. При этом проектируется только та функциональность, которая относится к текущей итерации, а любые будущие потребности не учитываются. Получившийся в результате процесс требует от разработчиков дисциплины, и в то же время сочетает

ее с высокой адаптивностью. Такое удивительное сочетание позволяет предположить, что XP является наиболее развитой адаптивной методологией.

Интерес к XP рос снизу вверх, от разработчиков и тестировщиков, замученных тягостным процессом, документацией, метриками и прочим формализмом. Они не отрицали дисциплину, но не желали бессмысленного соблюдения формальных требований. И искали новые быстрые и гибкие подходы к разработке высококачественных программ.

При использовании XP тщательное предварительное проектирование ПО заменяется, с одной стороны, постоянным присутствием в команде заказчика, готового ответить на любой вопрос и оценить любой прототип, а с другой стороны, регулярными переработками кода (так называемый рефакторинг). Основой проектной документации считается тщательно прокомментированный код. Очень большое внимание в методологии уделяется тестированию. Как уже отмечалось, для каждого нового метода сначала пишется тест, а потом уже разрабатывается собственно код метода до тех пор, пока тест не начнет выполняться успешно. Эти тесты сохраняются в наборах, которые автоматически выполняются после любого изменения кода.

Парное программирование и 40-часовая рабочая неделя, хотя и являются, возможно, наиболее известными чертами XP, носят все же вспомогательный характер и способствуют высокой производительности разработчиков и низкому количеству ошибок при разработке.

Crystal Clear

Алистэр Коуберн, автор рассматриваемой методологии, изучает методологии разработки ПО с начала 90-х, с тех пор как компания IBM дала ему задание написать работу на эту тему. При этом его подход существенно отличается от подхода большинства других методологов. Его теории основаны не только на личном опыте, но и на постоянных исследованиях других проектов и процессов. Более того, он не боится менять свои воззрения в результате своих находок в этой области.

Crystal – семейство методологий, определяющих необходимую степень формализации процесса разработки в зависимости от количества участников и критичности задач. Коуберн называет это "семейством", так как убежден, что разным проектам нужны разные методологии. Он вводит следующую градацию проектов: по одной оси откладывается количество занятых в проекте людей, по другой – критичность ошибок. Каждая из методологий "семейства" предназначена для определенной ячейки получившейся сетки. Таким образом, проект, в кото-

ром занято 40 человек, и на котором компания может позволить себе потерять некоторую сумму, будет работать по другой методологии, нежели проект для 6 разработчиков, от которого зависит существование компании.

Как и XP, методологии семейства Crystal тоже ориентированы на человека, однако несколько иначе. Считается, что людям сложно использовать в работе процесс, который требует от них высокой дисциплины. В результате своих исследований был определен уровень дисциплины, который является для действенной методологии минимальным. При этом производительность методологии сознательно приносилась в жертву простоте ее использования. Crystal уступает XP в производительности, зато ей сможет пользоваться большее количество людей, поскольку, требует минимальных усилий для внедрения, так как ориентирована на человеческие привычки, полагает Алистэр. Считается, что она описывает тот естественный порядок разработки ПО, который устанавливается в достаточно квалифицированных коллективах, если в них не занимаются целенаправленным внедрением другой методологии. Особое значение Алистэр придает пересмотрам процесса в конце каждой итерации, благодаря которым процесс можно усовершенствовать в ходе работы. При итеративной разработке все проблемы, связанные с используемым процессом, обнаруживаются рано, и их можно успеть исправить. Поэтому так важно контролировать процесс разработки и адаптировать его к конкретной ситуации.

Основные характеристики методологии:

- Итеративная инкрементная разработка;
- Автоматическое регрессионное тестирование;
- Пользователи привлекаются к активному участию в проекте;
- Состав документации определяется участниками проекта;
- Как правило, используются средства контроля версий кода.

Помимо Crystal Clear в семейство Crystal входит еще несколько методологий, предназначенных для выполнения более крупных или более критических проектов. Они отличаются несколько более жесткими требованиями к объему документации и вспомогательным процедурам, таким как управление изменениями и версиями.

SCRUM-методология

Scrum – одна из самых популярных методолгий гибкой разработки. Методология Scrum устанавливает правила управления процессом разработки и позволяет использовать уже существующие практики кодирования, корректируя требования или внося тактические изменения.

Использование этой методологии дает возможность выявлять и устранять отклонения от желаемого результата на более ранних этапах разработки программного продукта.

Основа Scrum – итеративная разработка. Scrum определяет правила, по которым должен планироваться и управляться список требований к продукту для достижения максимальной прибыльности от реализованной функциональности; правила планирования итераций для максимальной заинтересованности команды в результате; основные правила взаимодействия участников команды для максимально быстрой реакции на существующую ситуацию; правила анализа и корректировки процесса разработки для совершенствования взаимодействия внутри команды. Каждую итерацию можно описать так: планируем – фиксируем – реализуем – анализируем. За счет фиксирования требований на время одной итерации и изменения длины итерации можно управлять балансом между гибкостью и плановостью разработки.

Концепция Scrum

Scrum – простой каркас, который можно использовать для организации команды и достижения результата более продуктивно и с более высоким качеством за счет анализа сделанной работы и корректировки направления развития между итерациями. Методология позволяет команде выбрать задачи, которые должны быть выполнены, учитывая бизнес-приоритеты и технические возможности, а также решить, как их эффективно реализовать. Это позволяет создать условия, при которых команда работает с удовольствием и максимально продуктивно. К примеру, возможность самостоятельного выбора объема и пути решения задач без внешнего давления позволяет всем участникам команды почувствовать себя активными игроками, вовлеченными в процесс, а не простыми исполнителями, от которых требуется лишь четкая реализация поручений.

Scrum фокусируется на постоянном определении приоритетных задач, основываясь на бизнес целях, что увеличивает полезность и доходность проекта на его ранних стадиях. Так как при инициации проекта его доходность определить почти невозможно, Scrum предлагает концентрироваться на качестве разработки и к концу каждой итерации иметь промежуточный продукт, который можно использовать, пусть и с минимальными возможностями. Например, результатом итерации может быть каркас сайта, который можно показать на презентации.

Методология Scrum ориентирована на то, чтобы оперативно приспосабливаться к изменениям в требованиях, что позволяет команде бы-

стро адаптировать продукт к нуждам заказчика. Такая адаптация достигается за счет получения обратной связи по результатам итерации: имея после каждой итерации продукт, который уже можно использовать, показывать и обсуждать, легче собирать информацию и делать правильные корректировки и изменять приоритеты требований. Например, если каркас сайта показать потенциальным пользователям, то появится много вопросов, на основании которых можно скорректировать то, что уже написано или еще не реализовано, понять что более важно пользователю.

Девиз Scrum – "анализируй и адаптируй": анализируй то, что получил, адаптируй то, что есть, к реальной ситуации, а потом анализируй снова. Чем меньше формализма, тем более гибко и эффективно можно работать, – это основной принцип данной методологии. Но это не означает, что формальных процессов не должно быть совсем, их должно быть достаточно для организации эффективного взаимодействия и управления проектом. Формальная часть Scrum состоит из трех ролей, трех практик и трех основных документов.

Роли

Владелец продукта (Product Owner) – человек, поставляющий требования программистам. От того, как четко написаны требования, зависит, насколько часто команде придется переключаться с задачи на задачу в связи с отсутствием нужной информации, как много нужно задавать вопросов, на которые уходит дополнительное время, как сильно придется изменять уже написанную функциональность от итерации к итерации и, соответственно, эффективность разработки в целом. Обычно владелец продукта является представителем или доверенным лицом заказчика, а для компаний, выпускающих коробочные продукты, он представляет рынок, на котором реализуется продукт. Владелец продукта должен составить бизнес план, показывающий ожидаемую доходность и план развития с требованиями, отсортированными по коэффициенту окупаемости инвестиций. Исходя из имеющейся информации, владелец продукта подготавливает список требований, отсортированный по значимости. Чем лучше владелец продукта описывает требования, управляет приоритетами и чем быстрее выдает информацию, тем больший финансовый эффект получит компания от методологии. В обязанности этого сотрудника входит своевременное предоставление требований к продукту, определение дат и содержания релизов, эффективное управление приоритетами и корректировка требований для достижения максимальной окупаемости инвестиций в продукт.

От человека, исполняющего роль Scrum-мастера (Scrum Master), во многом зависит самостоятельность, инициативность программистов, удовлетворенность сделанной работой, атмосфера в команде и результат всей работы. Этот человек должен быть одним из членов команды разработки и участвовать в проекте как разработчик. Он отвечает за своевременное решение текущих проблем, от ремонта сломанного стула до обеспечения необходимой информацией членов команды для продолжения их работы и загруженности, за поддержание нужных технических практик, используемых на проекте. В обязанности Scrum-мастера входит обеспечение максимальной работоспособности и продуктивности команды, четкого взаимодействия между всеми участниками проекта, своевременное решение всех проблем, тормозящих или останавливающих работу любого члена команды, ограждение команды от всех воздействий извне во время итерации и обеспечение следования процессу всех участников проекта.

Scrum-команда (Scrum Team) – группа, состоящая из пяти-девяти самостоятельных, инициативных программистов. Первая задача этой команды – поставить реально достижимую, прогнозируемую, интересную и значимую цель для итерации. Вторая задача – сделать все для того, чтобы эта цель была достигнута в отведенные сроки и с заявленным качеством. Цель итерации считается достигнутой только в том случае, если все поставленные задачи реализованы, весь код написан по определенным проектом "стандартам кодирования" (coding guidelines), программа протестирована полностью, а все найденные дефекты устранены. Программисты этой команды должны уметь оценивать и планировать свою работу, работать в команде, постоянно анализировать и улучшать качество взаимодействия и работы. В обязанности всех членов Scrum-команды входит участие в выборе цели итерации и определение результата работы. Они должны делать все возможное и невозможное для достижения цели итерации в рамках, определенных проектом, эффективно взаимодействовать со всеми участниками команды, самостоятельно организовывать свою работу, предоставлять владельцу рабочий продукт в конце каждого цикла.

Практики

Подготовка к первой итерации, называемой спринт (Sprint), начинается после того, как владелец продукта разработал план проекта, определил требования и отсортировал их в количестве, достаточном для наполнения одной итерации. Такой список требований называется журналом продукта (Product Backlog). При планировании итерации проис-

ходит детальная разработка сессий планирования спринта (Sprint Planning Meeting), который начинается с того, что владелец продукта, Scrum-команда и Scrum-мастер проверяют план развития продукта, план релизов и список требований. Scrum-команда проверяет оценки требований, убеждается, что они достаточно точны, чтобы начать работать, решает, какой объем работы она может успешно выполнить за спринт, основываясь на размере команды, доступном времени и производительности. Важно, чтобы Scrum-команда выбирала первые по приоритету требования из журнала продукта. После того как Scrum-команда обязуется реализовать выбранные требования, Scrum-мастер начинает планирование спринта. Scrum-команда разбивает выбранные требования на задачи, необходимые для его реализации. Эта активность в идеале не должна занимать больше четырех часов, и ее результатом служит список требований, разбитый на задачи, – журнал спринта (Sprint Backlog). Необходимо, чтобы все участники команды приняли на себя обязательство по реализации выбранной цели.

После окончания планирования начинается итерация. Каждый день Scrum-мастер проводит "скрам" (Daily Scrum Meeting) – пятнадцатиминутное совещание, цель которого – достичь понимания того, что произошло со времени предыдущего совещания, скорректировать рабочий план к реалиям сегодняшнего дня и обозначить пути решения существующих проблем. Каждый участник Scrum-команды отвечает на три вопроса: что я сделал со времени предыдущего скрама, что меня тормозит или останавливает в работе, что я буду делать до следующего скрама? В этом митинге может принимать участие любое заинтересованное лицо, но только участники Scrum-команды имеют право принимать решения. Правило обосновано тем, что они давали обязательство реализовать цель итерации, и только это дает уверенность в том, что она будет достигнута. На них лежит ответственность за их собственные слова, и, если кто-то со стороны вмешивается и принимает решения за них, тем самым он снимает ответственность за результат с участников команды. Последовательность шагов разработки методологии представлена на рис. 6.

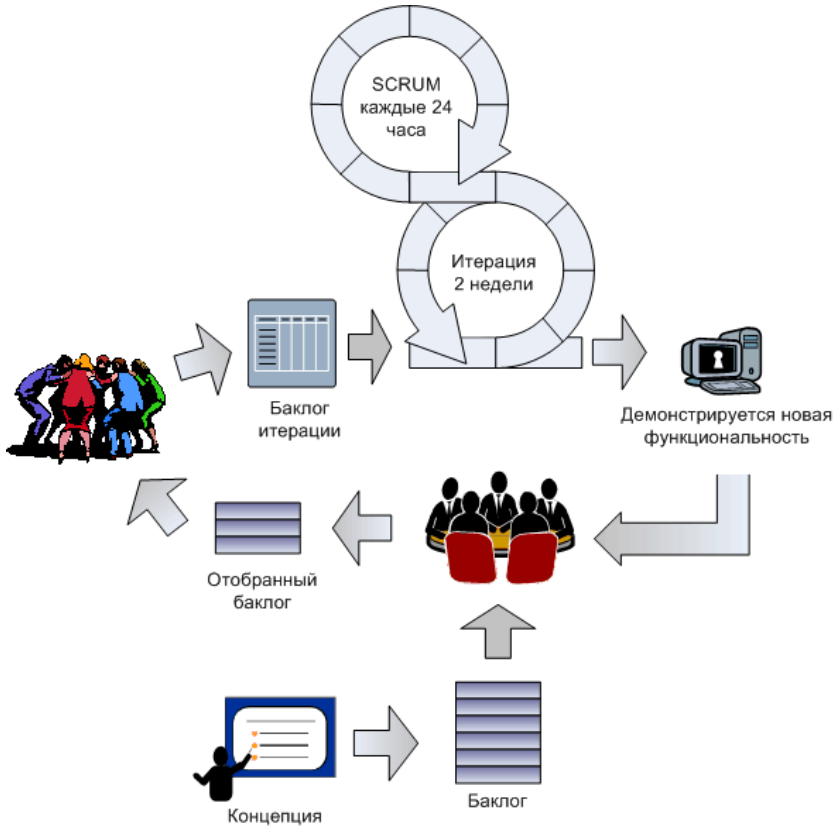


Рис. 6. Последовательность разработки Sprint

В конце каждого спринта проводится демонстрационный митинг (Sprint Review Meeting) продолжительностью не более четырех часов. Сначала Scrum-команда демонстрирует владельцу продукта сделанную в течение спринта работу, а тот в свою очередь ведет эту часть митинга и может пригласить к участию всех заинтересованных заказчиков. Владелец продукта определяет, какие требования из журнала спринта были выполнены, и обсуждает с командой и заказчиками, как лучше расставить приоритеты в журнале продукта для следующей итерации. Во второй части митинга производится анализ прошедшего спринта, который ведет Scrum-мастер. Scrum-команда ищет использованные в последнем спринте положительные и отрицательные способы совместной работы, анализирует их, делает выводы и принимает важные для дальнейшей работы решения. Scrum-команда также определяет программы, которые могут работать лучше, и ищет пути для увеличения эффективности дальнейшей работы. Затем цикл замыкается, и начинается планирование следующего спринта.

Документы

В начале проекта владелец продукта готовит журнал продукта (рис. 7) – список требований, отсортированный по значимости, а Scrum-команда дополняет этот журнал оценками стоимости реализации требований. Список должен включать функциональные и технические требо-

вания, необходимые для реализации продукта. Самые приоритетные из них должны быть достаточно детально прописаны, чтобы их можно было оценить и протестировать. О своевременной детализации требований должен заботиться владелец продукта и предоставлять необходимый объем в нужное время. В этом смысле программисты являются заказчиками требований для владельца продукта. В дальнейшем остальные требования должны постепенно уточняться и детализироваться до такого же уровня. Главное, чтобы у команды всегда был достаточный объем подготовленных к реализации требований.

Номер требования	Описание требования	Ценность бизнеса	Приоритет	Высокоуровневая оценка (часы)
FE1	Покупатель может зарегистрироваться на сайте	$10,000	1	20
FE2	Покупатель может ввести свои персональные данные	$12,000	5	36
FE3	Покупатель может увидеть список доступных изделий	$15,000	10	30
FE4	Покупатель может купить изделие	$100,000	20	48
FE5	Покупатель может делать поиск изделий	$80,000	30	32
FE6	Покупатель может подписаться на новости	$30,000	40	56

Рис. 7. Журнал продукта

После того как команда во время сессии планирования выбрала и обязалась реализовать набор требований из журнала продукта, эти требования разбиваются на более мелкие задачи, составляющие детализированный список требований – журнал спринта. Разбивка на задачи должна быть сделана таким образом, чтобы выполнение одной задачи не занимало больше двух дней (считается, что менее детальная, например, полдня, разбивка приводит к более грубой оценке, чем приемлема в большинстве проектов, использующих методологию Scrum). Разбивка на задачи поможет так спланировать итерацию, чтобы в конце не осталось ни одной невыполненной задачи и, соответственно, достичь ее цели. После завершения детализации оценка журнала спринта сравнивается с первичной оценкой в журнале продукта. Если существует значительное расхождение, команда договаривается с владельцем продукта об объеме работ, который должен быть выполнен в течение итерации, и о том, какой объем будет перенесен на следующую. Менее важные и мало влияющие на цель итерации задачи выносятся из журнала спринта.

График спринта (Burndown Chart) показывает ежедневное изменение общего объема работ, оставшегося до окончания итерации. Это график позволяет команде разработчиков делать анализ текущей ситуации и своевременно реагировать на отклонения. График спринта позволяет также владельцу продукта наблюдать за ходом итерации – если общий объем работ не уменьшается каждый день, значит, что-то идет не так. Во время сессии планирования команда находит и оценивает задачи, которые надо выполнить для успешного завершения итерации. Сумма оценок всех задач в журнале спринта является общим объемом работы, который надо выполнить за итерацию. После завершения каж-

дой задачи Scrum-мастер пересчитывает объем оставшейся работы и отмечает это на графике спринта. Только в том случае, если объем работ по окончании итерации закончился (в журнале спринта не осталось незавершенных задач), итерация считается успешной. График спринта используется как вспомогательный инструмент, позволяющий корректировать работу для завершения итерации вовремя, с работающим кодом и требуемым качеством.

Время между итерациями – это время принятия основополагающих решений, влияющих на ход всего проекта. Во время итерации никакие изменения извне не могут быть сделаны. После того как команда дала обязательство реализовать журнал спринта, он фиксируется, и изменения в нем могут быть сделаны только по следующим причинам:

- Scrum-команда в течение итерации получила лучшее представление о требованиях и нуждается в дополнительных задачах для успешного завершения итерации;
- найдены дефекты, которые нужно обязательно исправить для успешного завершения итерации;
- Scrum-мастер и Scrum-команда могут решить, что небольшие изменения, не влияющие на общий объем работ, могут быть реализованы в связи с возникшей у владельца продукта необходимостью.

Исходя из того, что журнал спринта не может быть изменен извне во время итерации, нужно выбирать ее длину, основываясь на стабильности требований. Если требования стабильны, меняются или дополняются редко, то можно выбрать шестинедельный цикл. В этом случае экономится время на переключение команды с активной разработки на планирование и демонстрационные митинги. Если требования часто меняются и дополняются, нужно отталкиваться от двухнедельного цикла, в любом случае длина итерации – это величина экспериментальная.

Функционально-ориентированная разработка (Feature Driven Development-FDD)

Эта методология (кратко именуемая FDD) была разработана Джеффом Де Люка (Jeff De Luca) и признанным гуру в области объектно-ориентированных технологий Питером Коадом (Peter Coad). Как и остальные адаптивные методологии, она делает основной упор на коротких итерациях, каждая из которых служит для проработки определенной части функциональности системы. Согласно FDD, одна итерация длится две недели. На самом деле используемое в FDD понятие функции или свойства (feature) системы достаточно близко к понятию

прецедента использования, используемому в RUP. Едва ли не самое существенное отличие – это дополнительное ограничение: "каждая функция должна допускать реализацию не более, чем за две недели". То есть если сценарий использования достаточно мал, его можно считать функцией. Если же велик, то его надо разбить на несколько относительно независимых функций.

FDD включает пять процессов, последние два из которых повторяются для каждой функции:

- Разработка общей модели
- Составление списка необходимых функций системы
- Планирование работы над каждой функцией
- Проектирование функции
- Конструирование функции

Разработчики в FDD делятся на "хозяев классов" и "главных программистов". Главные программисты привлекают хозяев задействованных классов к работе над очередным свойством. Для сравнения, в XP нет персонально ответственных за классы или методы. К работе над любым классом может привлекаться любой из участников разработки.

Работа над проектом предполагает частые сборки и делится на итерации, каждая из которых предполагает реализацию определенного набора функций.

Вопросы для самопроверки

1. Структура ЖЦ разработки информационных систем по стандарту ISO/IEC 12207. Охарактеризуйте основные процессы ЖЦ ИС.
2. Структура ЖЦ разработки информационных систем по стандарту ISO/IEC 12207. Охарактеризуйте вспомогательные процессы, обеспечивающие выполнение основных процессов ЖЦ ИС.
3. Структура ЖЦ разработки информационных систем по стандарту ISO/IEC 12207. Охарактеризуйте организационные процессы.
4. Охарактеризуйте содержание, сферу применения, достоинства и недостатки эвалюционной модели
5. Охарактеризуйте содержание, сферу применения, достоинства и недостатки модели, основанной на формальных преобразованиях
6. В чем состоят особенности итерационных моделей
7. Охарактеризуйте содержание, сферу применения, достоинства и недостатки спиральной модели
8. Что отличает тяжеловесные модели от быстрой разработки
9. Определите четыре ценности, положенные в основу современной методологии гибкой разработки

10. Определите 12 принципов гибкой разработки
11. Дайте характеристику методологии экстремального программирования
12. Дайте характеристику методологии Crystal Clear
13. Дайте характеристику SCRUM-методологии
14. Дайте характеристику FDD-методологии

Характеристика технологий создания ИС

Как уже отмечалось ранее, существуют различные технологии реализации модели жизненного цикла при создании информационной системы. Вопрос применения какой-либо методологии в чистом виде либо компиляция и адаптация этапов жизненного цикла различных подходов создания ИС решается в каждом конкретной команде разработчиков индивидуально, основываясь, прежде всего, на эффективности применяемого подхода. Ниже рассматриваются наиболее распространенные из методологий реализации жизненного цикла разработки ИС.

Визуальное моделирование

Под моделью ИС в общем случае понимается формализованное описание системы на определенном уровне абстракции и этапе проектирования. Каждая модель представляет конкретный аспект системы, использует диаграммы и документы определенного формата, а также отражает точку зрения и является объектом деятельности различных категорий интересов, ролями или задачами.

Графические (визуальные) модели представляют собой средства для визуализации, описания, проектирования и документирования элементов системы, помогающих представлять и анализировать проблемы, определять требования бизнеса и проектировать ИС. Разработка модели системы ИС промышленного характера в такой же мере необходима, как и наличие проекта при строительстве большого здания. Хорошие модели являются в ряде случаев единственной основой взаимодействия участников проекта и гарантируют правильность постановки проблемы и архитектуры системы. Поскольку сложность систем повышается, важно располагать хорошими методами моделирования. Хотя существует много других факторов, от которых зависит успех проекта, но наличие строгого стандарта языка моделирования является весьма существенным. Применение тех или иных инструментальных средств вообще не является необходимым, если заказчик не настаивает на этом. Таким образом, построение графических моделей можно выполнять на бумаге или планшете и поддерживать ее актуальность до тех поре, пока в этом есть необходимость.

Состав моделей, используемых в каждом конкретном проекте, и степень их детальности в общем случае зависят от следующих факторов:

- сложности проектируемой системы;
- необходимой полноты ее описания;
- знаний и навыков участников проекта;
- времени, отведенного на проектирование.

Визуальное моделирование оказало большое влияние на развитие технологий создания ПО вообще, и CASE-средств в частности. Понятие CASE (Computer Aided Software Engineering) используется в настоящее время в весьма широком смысле. Первоначальное значение этого понятия, ограниченное только задачами автоматизации разработки ПО, в настоящее время приобрело новый смысл, охватывающий большинство процессов жизненного цикла создания ИС.

CASE-технология представляет собой совокупность методов проектирования компонентов ИС, а также набор инструментальных средств, позволяющих в наглядной форме моделировать предметную область, анализировать эту модель на всех стадиях разработки и сопровождения ПО, а также разрабатывать приложения в соответствии с информационными потребностями пользователей. Большинство существующих CASE-средств основано на методах структурного или объектно-ориентированного анализа и проектирования, использующих спецификации в виде диаграмм или текстов для описания внешних требований, связей между моделями системы, динамики поведения системы и архитектуры программных средств.

Методы структурного анализа и проектирования систем

Структурные методы анализа и проектирования систем – это группа методологий, разработанных, как правило, еще до широкого распространения объектно-ориентированных подходов. Все они предполагают каскадную разработку, и в рамках структурного анализа и проектирования предполагают создание различных моделей, описывающих:

1. Функциональную структуру системы;
2. Последовательность выполняемых действий;
3. Передачу информации между функциональными процессами;
4. Отношения между данными.

Современное структурное проектирование – ориентированная на процессы техника для разбиения большой программы на иерархию модулей в результате чего программа легче реализуется и сопровождается

(изменяется). Синоним, (хотя технически неточный)- программирование сверху вниз (нисходящее) и структурное программирование.

Модель ПО, полученная структурным проектированием называется «структурная схема» – Structure Chart.

В процессе выполнения структурного проектирования отыскиваются факторы программы в нисходящей иерархии модулей, которые имеют следующие свойства:

- Модули должны иметь сильную внутреннюю связность (highly cohesive); Каждый модуль должен выполнять одну и только одну функцию. Это позволяет многократно использовать (reusable) модули в будущих программах.
- Модули должны быть слабо связаны между собой (loosely coupled); иными словами, модули должны быть минимально зависимы между собой. Это минимизирует эффект влияния будущих изменений одного модуля на другой.

Группировать все модули (или процессы) вызванный одной операций для формирования операционного центра. Например, все задачи, выполняемые при получении заказа от поставщика, связаны. Часто, центр управления служит как модуль управления.

Наиболее распространенными моделями первых трех групп являются:

1. функциональная модель SADT (Structured Analysis and Design Technique);
2. модель IDEF3;
3. DFD (Data Flow Diagrams) – диаграммы потоков данных.

Метод SADT

Метод SADT представляет собой совокупность правил и процедур, предназначенных для построения функциональной модели объекта рассматриваемой предметной области. Функциональная модель SADT отображает функциональную структуру объекта, т. е. производимые им действия и связи между этими действиями. Метод SADT успешно применялся в военных, промышленных и коммерческих организациях США для решения широкого круга задач, таких как, долгосрочное и стратегическое планирование, автоматизированное производство и проектирование, разработка ПО для оборонных систем, управление финансами и материально-техническим снабжением и др. Метод SADT поддерживается Министерством обороны США, которое было инициатором разработки семейства стандартов IDEF (Icam DEFinition), являющегося основной частью программы ICAM (интегрированная компьютери-

зация производства), проводимой по инициативе ВВС США. Метод SADT реализован в одном стандартов этого семейства – IDEF0, который был утвержден в качестве федерального стандарта США в 1993 г., его подробные спецификации можно найти на сайте http://www.idef.com.

Модели SADT (IDEF0-модель) традиционно используются для функционально структурного моделирования организационных систем (бизнес-процессов) и основаны на декомпозиционном подходе. Следует отметить, что метод SADT успешно работает только при описании хорошо специфицированных и стандартизованных бизнес-процессов, поэтому он и принят в США в качестве стандартного.

Модель IDEF0 представляет собой набор диаграмм с поддерживающей их документацией, включающей сопровождающие тексты и словарь. Диаграммы – главные компоненты модели, все функции ИС и интерфейсы на них представлены как блоки и дуги. Диаграммы модели декомпозируют сложный объект на составные части. Первоначальная (исходная, корневая) диаграмма является наиболее общим и наиболее абстрактным описанием всей системы в целом. Она показывает основную функциональную составляющую системы в виде блока. Детали (компоненты) каждого из основных блоков показаны на других диаграммах в виде блоков. Далее они могут быть с помощью декомпозиции преобразованы в более подробные диаграммы, до тех пор, пока не будет достигнута требуемая степень детализации.

Блоки представляют собой функции (действия, процессы или операции), а входящие и исходящие из них стрелки представляют данные (информацию, предметы). Блок и его стрелки на диаграмме рассматриваемого уровня описывается более подробно блокам и стрелками диаграмм нижнего уровня.

Блоки-функции показывают, что должно выполняться, причем без идентификации каких-либо других аспектов, например, таких как потребности в них или средства их осуществления. Наименование функций записывается внутри блока. Оно должно содержать существительное, обозначающее действие. Блоки нумеруются в нижнем углу.

Стрелки на диаграмме играют роль интерфейсов (связей) блоков с внешней для них средой. Каждая из стрелок имеет метку, характеризующую ее, известную как ICOM-нотацию. Назначение стрелок зависит от стороны блока, в которую стрелка входит или выходит (рис. 8):

Входящие стрелки слева от блока представляет собой предметы (материальные объекты) или информацию (информационные объекты), необходимые для выполнения функции. Это сырье, материалы или "вход" функции (стрелка типа I – input).

- Выходящие стрелки справа из блока показывают данные, полученные в результате выполнения функции блока. Это результат (стрелки типа O – output). Функции преобразуют данные слева направо (от входа к выходу). Блок представляет собой переход от состояния "до" к состоянию "после".
- Входящие стрелки сверху блока является управлением, описывающим условия или обстоятельства, которые управляют выполнением функции (стрелки типа C – control). Они тоже могут нести данные, но назначение входа и управления различны. Их разделение является важным для понимания работы системы. Каждый функциональный блок имеет, по крайней мере, одну управляющую стрелку.
- Входящая снизу стрелка представляет собой "механизм", обозначающая собой либо человека, либо некоторое средство, выполняющее функцию (стрелка M – mechanism). Вход и выход показывают, что делается функцией, управление показывает, почему это делается, а механизм показывает, как это делается.

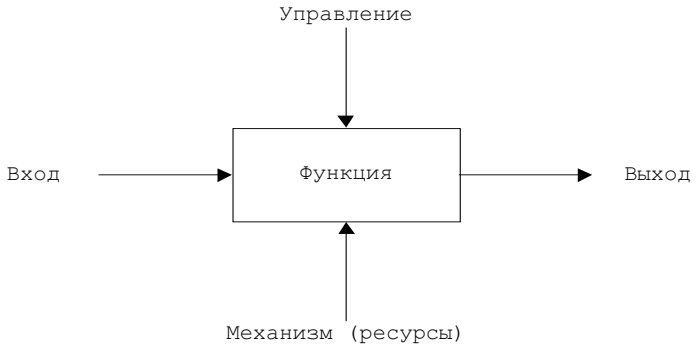


Рис. 8. Графическая модель процесса

Стрелки на диаграмме IDEF0 означают ограничения, задаваемые данными. Они не представляют собой поток или последовательность. Соединяя выход одного блока с входом другого, они показывают ограничения. Блок, получающий данные, "ограничен" в том смысле, что функция не может быть выполнена, пока не будут получены данные, производимые другими блоками. Стрелки, входящие в блок, показывают все данные, которые необходимы для выполнения функции.

Несколько функций на диаграмме могут выполняться одновременно, если удовлетворены все ограничивающие условия. Ни последовательность, ни время не являются точно определенными в IDEF0. Обратная связь, итерация, непрерывные процессы, перекрытие (во времени) функций легко отображаются стрелками.

Стрелки могут разветвляться или соединяться. Каждая из ветвей может представлять один и тот же объект или различные объекты одного и того же типа.

Пример IDEF0-диаграммы представлен на рис. 9.

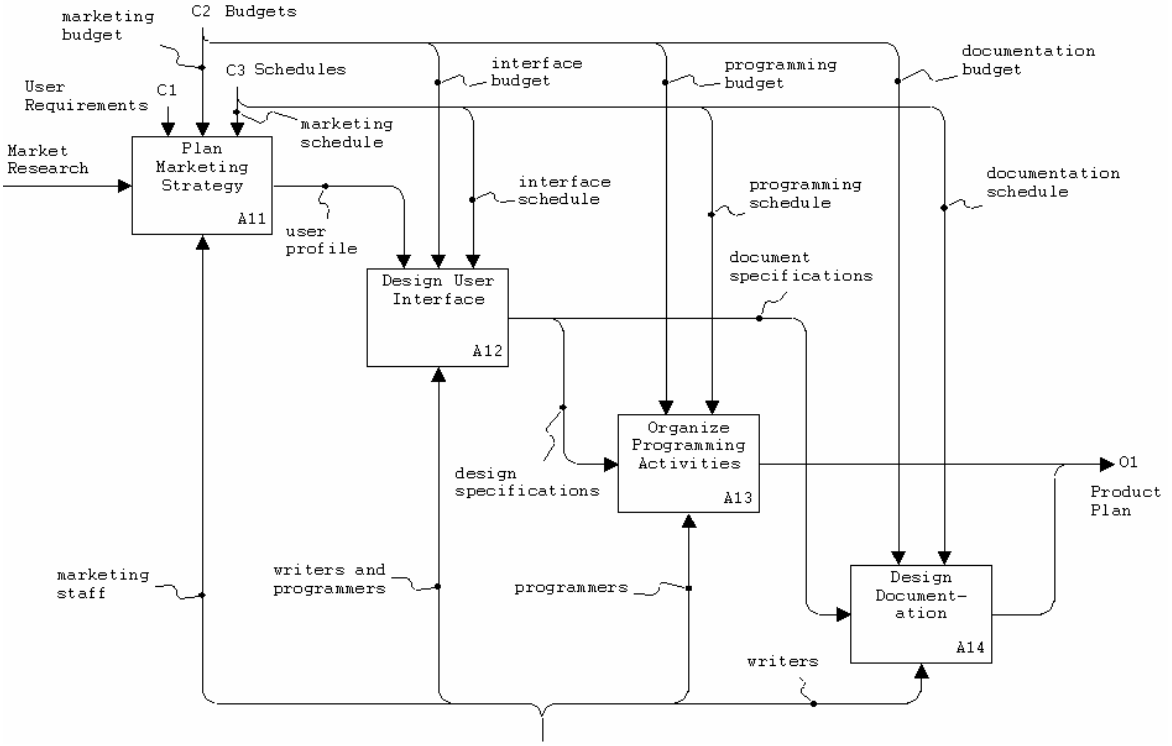


Рис. 9. Пример построенной IDEF0-диаграммы

Достоинствами применения моделей SADT для описания бизнес-процессов являются:

- полнота описания бизнес-процесса (управление, информационные и материальные потоки, обратные связи);
- жесткие требования метода, обеспечивающих получение моделей стандартного вида;
- соответствие подхода к описанию процессов стандартам ISO 9000.

Во многих российских компаниях в последнее время начали формироваться и развиваться процессные подходы организации деятельности, и, в этой связи, широкое применение получают функциональные модели на основе применения моделирования IDEF0.

Метод моделирования IDEF3

Метод моделирования IDEF3, являющийся частью Design/IDEF методологии, был разработан в конце 1980-х годов для закрытого проекта ВВС США. Этот метод предназначен для таких моделей процессов, в которых важно понять последовательность выполнения действий и

взаимозависимости между ними. Хотя IDEF3 и не является федеральным стандартом США, он приобрел широкое распространение среди системных аналитиков как дополнение к методу функционального моделирования IDEF0 (модели IDEF3 могут использоваться для детализации функциональных блоков IDEF0, не имеющих диаграмм декомпозиции). Основой модели IDEF3 служит, так называемый, сценарий процесса, который выделяет последовательность действий и подпроцессов анализируемой системы. IDEF3, называемая также workflow diagramming, — методология моделирования, использующая графическое описание информационных потоков, взаимоотношений между процессами обработки информации и объектов, являющихся частью этих процессов. Диаграммы Workflow могут быть использованы в моделировании бизнес-процессов для анализа завершенности процедур обработки информации. С их помощью можно описывать сценарии действий сотрудников организации, например последовательность обработки заказа или события, которые необходимо обработать за конечное время. Каждый сценарий сопровождается описанием процесса и может быть использован для документирования каждой функции.

IDEF3 — это метод, имеющий основной целью дать возможность аналитикам описать ситуацию, когда процессы выполняются в определенной последовательности, а также описать объекты, участвующие совместно в одном процессе.

Техника описания набора данных IDEF3 является частью структурного анализа. В отличие от некоторых методик описаний процессов IDEF3 не ограничивает аналитика чрезмерно жесткими рамками синтаксиса, что может привести к созданию неполных или противоречивых моделей.

IDEF3 может быть также использован как метод создания процессов. IDEF3 дополняет IDEF0 и содержит все необходимое для построения моделей, которые в дальнейшем могут быть использованы для имитационного анализа.

Каждая работа в IDEF3 описывает какой-либо сценарий бизнес-процесса и может являться составляющей другой работы. Поскольку сценарий описывает цель и рамки модели, важно, чтобы работы именовались существительным, обозначающим процесс действия, или фразой, содержащей такое существительное.

Точка зрения на модель должна быть документирована. Обычно это точка зрения человека, ответственного за работу в целом. Также необходимо документировать цель модели – те вопросы, на которые призвана ответить модель.

Единицы работы — Unit of Work (UOW) — также называемые работами (activity), являются центральными компонентами модели. В IDEF3 работы изображаются прямоугольниками с прямыми углами и имеют имя, выраженное отглагольным существительным, обозначающим процесс действия, одиночным или в составе фразы, и номер (идентификатор); другое имя существительное в составе той же фразы обычно отображает основной выход (результат) работы (например, "Изготовление изделия"). Часто имя существительное в имени работы меняется в процессе моделирования, поскольку модель может уточняться и редактироваться. Идентификатор работы присваивается при создании и не меняется никогда. Даже если работа будет удалена, ее идентификатор не будет вновь использоваться для других работ. Обычно номер работы состоит из номера родительской работы и порядкового номера на текущей диаграмме.

Диаграмма является основной единицей описания в IDEF3. Важно правильно построить диаграммы, поскольку они предназначены для чтения другими людьми (а не только автором). Пример диаграммы IDEF3 предствлен на рис. 10.

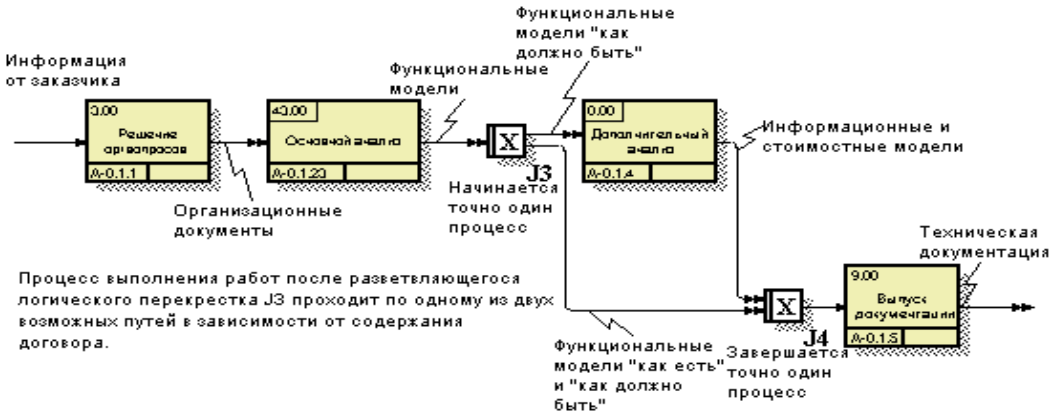


Рис. 10. Пример диаграммы IDEF3

Диаграммы потоков данных

Диаграммы потоков данных (Data Flow Diagrams – DFD) представляют собой иерархию функциональных процессов, связанных потоками данных. Цель такого представления – продемонстрировать, как каждый процесс преобразует свои входные данные в выходные, а также выявить отношения между этими процессами.

Для построения DFD традиционно используются две различные нотации, соответствующие методам Йордона-ДеМарко и Гейна-

Сэрсона. Эти нотации незначительно отличаются друг от друга графическим изображением символов. В соответствии с данными методами модель системы определяется как иерархия диаграмм потоков данных, описывающих асинхронный процесс преобразования информации от ее ввода в систему до выдачи потребителю. Практически любой класс систем успешно моделируется при помощи DFD-ориентированных методов. Диаграммы потоков данных показывают, как каждый *процесс* преобразует свои входные данные в выходные, и выявляют отношения между этими *процессами*. DFD-диаграммы успешно используются как дополнение к модели IDEF0 для описания документооборота и обработки информации. Подобно IDEF0, DFD представляет моделируемую систему как сеть связанных работ. Основные компоненты DFD (как было сказано выше) – *процессы* или работы, *внешние сущности*, потоки данных, накопители данных (хранилища).

В DFD работы (*процессы*) представляют собой функции системы, преобразующие входы в выходы. Хотя работы изображаются прямоугольниками со скругленными углами, смысл их совпадает со смыслом работ IDEF0 и IDEF3. Так же, как *процессы* IDEF3, они имеют входы и выходы, но не поддерживают управления и механизмы, как IDEF0.

Внешние сущности изображают входы в систему и/или выходы из системы. *Внешние сущности* изображаются в виде прямоугольника с тенью и обычно располагаются по краям диаграммы. Одна *внешняя сущность* может быть использована многократно на одной или нескольких диаграммах. Обычно такой прием используют, чтобы не рисовать слишком длинных и запутанных стрелок.

Потоки работ изображаются стрелками и описывают движение объектов из одной части системы в другую. Поскольку в DFD каждая сторона работы не имеет четкого назначения, как в IDEF0, стрелки могут подходить и выходить из любой грани прямоугольника работы. В DFD также применяются двунаправленные стрелки для описания диалогов типа "команда-ответ" между работами, между работой и *внешней сущностью* и между *внешними сущностями*. Пример диаграммы DFD приведен на рис. 11

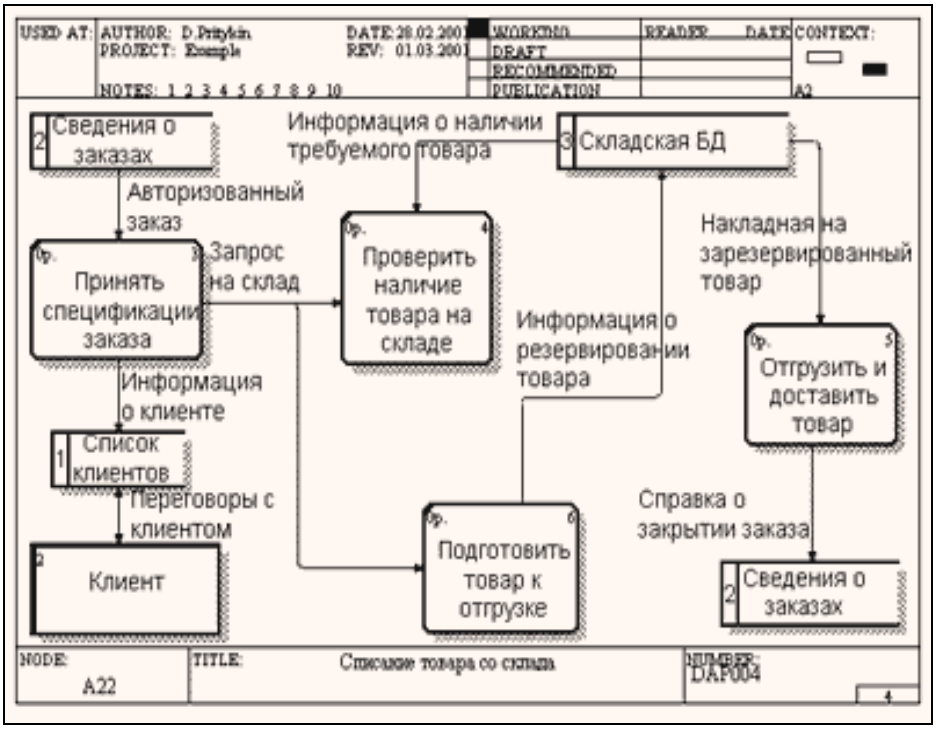


Рис. 11. Пример диаграммы DFD

Модель "сущность-связь"

Модель "сущность-связь" (Entity-Relationship Model — ERM) является наиболее распространенным средством моделирования данных предметной области. Она была впервые введена Питером Ченом в 1976 г. Эта модель традиционно используется в структурном анализе и проектировании, однако, по существу, представляет собой подмножество объектной модели предметной области. Одна из разновидностей модели Entity-Relationtion ("сущность-связь") используется в методе IDEF1X, входящем в семейство стандартов IDEF и реализованном в ряде распространенных CASE-средств (в частности, Design/IDEF, AllFusion ERwin Data Modeler, Oracle). Пример информационной модели приведен на рис. 12. Этапы и задачи моделирования IDEF1X можно представить в следующей последовательности:

- планирование проекта;
- сбор данных;
- определение сущностей;
- определение отношений;
- определение ключевых атрибутов;
- заполнение неключевых атрибутов;
- проверка правильности модели;
- приемка модели.

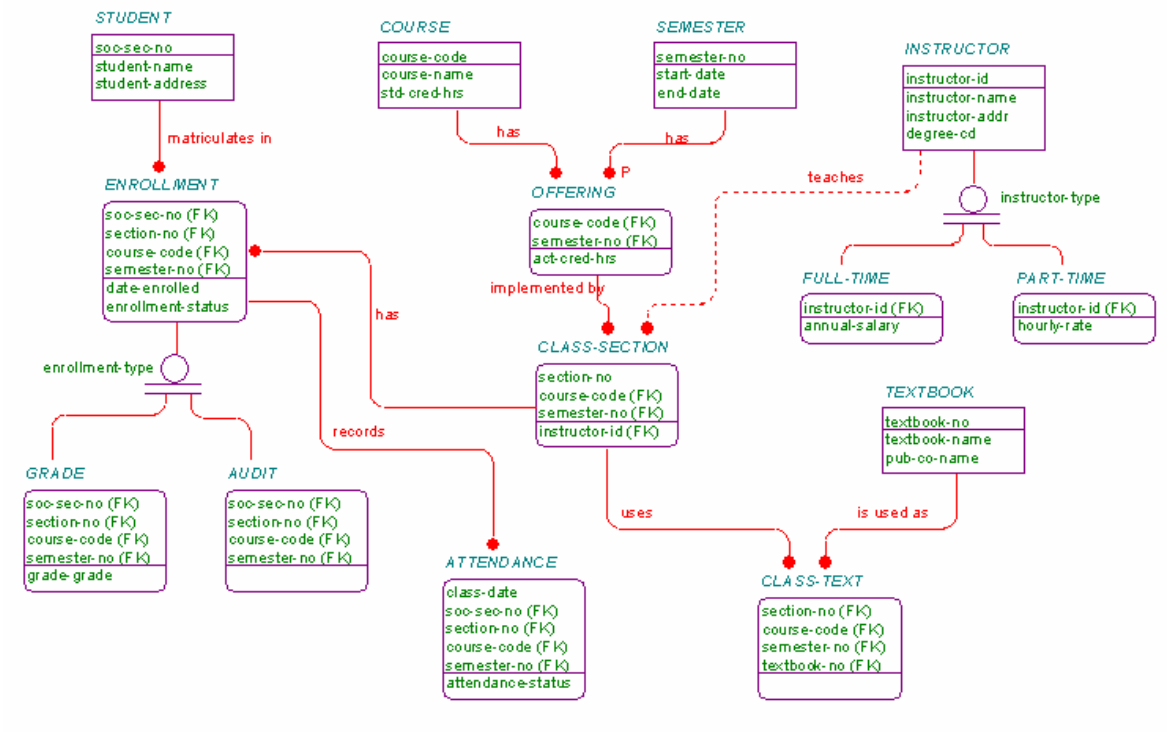


Рис. 12. Пример информационной модели

Объектно-ориентированный анализ и проектирование

Технология структурного анализа и проектирования не способна поставлять функционально полные системы вовремя и в пределах бюджета и часто не позволяет удовлетворить полные потребности распределенных информационных систем в условиях возрастающей сложности проектов. В последнее время наиболее популярной методологией состадия компонентов информационных технологий становится объектно-ориентированный анализу и проектирование (ООАП).

Объектно-ориентированный подход (ООП) сформировался и развивается вследствие ниже приведенных обстоятельств создания ИС:

- Необходимость повышения производительности разработки за счет многократного (повторного) использования разработанного ПО.
- Увеличение надежности за счет ранее проверенных решений, реализованных в повторно используемых компонентах.
- Необходимость упрощения сопровождения и модификации разработанных систем (локализация вносимых изменений).
- Упрощение проектирования систем на основе моделирования (за счет сокращения семантического разрыва между структурой решаемых задач и структурой ПО).

В основе объектно-ориентированного подхода лежит объектная декомпозиция, при этом статическая структура ПО описывается в терминах объектов и связей между ними, а динамический аспект ПО описывается в терминах обмена сообщениями между объектами. Каждый объект системы обладает своим собственным поведением, моделирующим поведение объекта реального мира.

Ее основные принципы (абстрагирование, инкапсуляция, модульность и иерархия) и понятия (объект, класс, атрибут, операция, интерфейс и др.) наиболее четко сформулированы в работах Гради Бучем.

Абстрагирование – это выделение наиболее важных, существенных характеристик некоторого объекта, которые отличают его от всех других видов объектов, и игнорирование менее важных или незначительных деталей. Абстрагирование позволяет управлять сложностью системы, концентрируясь на существенных свойствах объекта. Выбор правильного набора абстракций для заданной предметной области представляет собой главную задачу объектно-ориентированного проектирования. Объекты и классы – основные абстракции предметной области.

Инкапсуляция – локализация свойств и поведения в рамках единственной абстракции (рассматриваемой как «черный ящик»), скрывающей реализацию за общедоступным интерфейсом. Инкапсуляция – это отделение внутреннего устройства объекта от его внешнего поведения. Объектный подход предполагает, что внутренние ресурсы объекта, скрыты от внешней среды и доступны с помощью внешнего интерфейса.

Модульность – это декомпозиция системы в виде набора внутренне сильно сцепленных, но слабо связанных между собой подсистем (модулей). Модульность снижает сложность системы, позволяя выполнять независимую разработку отдельных модулей.

Иерархия – это упорядоченная система абстракций, задающая их расположение по уровням. Основными видами иерархических структур сложных систем являются структура классов и структура объектов. Иерархия классов строится по наследованию, а иерархия объектов – по агрегации.

К основным понятиям объектно-ориентированного подхода (*элементам объектной модели*) относятся: объект; класс; атрибут; операция; полиморфизм; наследование; компонент; связь.

Объект – это сущность предметной области или программной системы, имеющий четко определяемое поведение. Любой объект обладает состоянием, поведением и индивидуальностью. *Состояние объекта* определяется значениями его свойств (атрибутов) и связями с другими объектами, оно может меняться в процессе жизненного цикла объ-

екта. *Поведение* определяет действия объекта и его реакцию на запросы от других объектов в контексте состояния объекта. Поведение представляется с помощью набора сообщений, воспринимаемых объектом (операций, которые может выполнять объект). *Индивидуальность* – это свойства объекта, отличающие его от всех других объектов.

Структура и поведение схожих объектов определяют общий для них класс. *Класс* – это множество объектов, связанных общностью свойств, поведения, связей и семантики. Любой объект является экземпляром класса. Определение классов и объектов – одна из самых сложных задач объектно-ориентированного проектирования.

Атрибут – поименованное свойство класса, определяющее диапазон допустимых значений, которые могут принимать экземпляры данного свойства. Атрибуты могут быть скрыты от других классов, это определяет видимость атрибута: public (общий, открытый); private (закрытый, секретный); protected (защищенный).

Определенное воздействие одного объекта на другой с целью вызвать соответствующую реакцию называется операцией или посылкой сообщения. *Операция* – это реализация услуги, которую можно запросить у любого объекта данного класса. Операции реализуют связанное с классом поведение, его обязанности. Описание операции включает четыре части: имя; список параметров; тип возвращаемого значения; видимость.

Понятие полиморфизма может быть интерпретировано, как способность класса принадлежать более чем одному типу. *Полиморфизм* – это способность скрывать множество различных реализаций под единственным общим интерфейсом. Интерфейс – это совокупность операций, определяющих набор услуг класса или компонента. Интерфейс не определяет внутреннюю структуру, все его операции открыты.

Наследование – это построение новых классов на основе существующих с возможностью добавления или переопределения свойств (атрибутов) и поведения (операций).

Компонент – это относительно независимая и замещаемая часть системы, выполняющая четко определенную функцию в контексте заданной архитектуры. Виды компонентов: компонент исходного кода; компонент времени выполнения; исполняемый компонент.

Большинство современных методов ООАП базируются на использовании языка UML. Унифицированный язык моделирования UML (Unified Modeling Language) представляет собой язык для определения, представления, проектирования и документирования программных систем, организационно-экономических систем, технических систем и дру-

гих систем различной природы. UML содержит стандартный набор диаграмм и нотаций самых разнообразных видов.

UML – это преемник того поколения методов ООАП, которые появились в конце 1980-х и начале 1990-х годов и главными в разработке UML являются цели:

1. предоставить пользователям готовый к использованию выразительный язык визуального моделирования, позволяющий разрабатывать осмысленные модели и обсуждать их;
2. предусмотреть механизмы расширяемости и специализации для расширения за счет использования стереотипов;
3. обеспечить независимость от конкретных языков программирования и процессов разработки.
4. обеспечить формальную основу для понимания этого языка моделирования (язык должен быть одновременно точным и доступным для понимания, без лишнего формализма);
5. стимулировать рост рынка объектно-ориентированных инструментальных средств;
6. интегрировать лучший практический опыт.

UML применяется практически всеми крупнейшими компаниями – производителями ПО (Microsoft, Oracle, IBM, Hewlett-Packard, Sybase, Sun Microsystems и др.). Кроме того, практически все мировые производители CASE-средств, помимо IBM Rational Software и Eclipse, поддерживают UML в своих продуктах (Oracle Designer, Together Control Center (Borland), AllFusion Component Modeler (Computer Associates), Microsoft Visual Modeler и др.). Полное описание UML можно найти на сайтах http://www.omg.org, http://www.rational.com, . http://www.eclipse.org

Стандарт UML версии 1.1, принятый OMG в 1997 г., содержит следующий набор диаграмм:

Структурные (structural) модели:

- диаграммы классов (class diagrams) – для моделирования статической структуры классов системы и связей между ними;
- диаграммы компонентов (component diagrams) – для моделирования иерархии компонентов (подсистем) системы;
- диаграммы размещения (deployment diagrams) – для моделирования физической архитектуры системы.

Модели поведения (behavioral):

- диаграммы вариантов использования (use case diagrams) – для моделирования функциональных требований к системе (в виде сценариев взаимодействия пользователей с системой);

- диаграммы взаимодействия (interaction diagrams):
- диаграммы последовательности (sequence diagrams) и кооперативные диаграммы (collaboration diagrams) – для моделирования процесса обмена сообщениями между объектами;
- диаграммы состояний (statechart diagrams) – для моделирования поведения объектов системы при переходе из одного состояния в другое;
- диаграммы деятельности (activity diagrams) – для моделирования поведения системы в рамках различных вариантов использования, или потоков управления.

В UML 2.0 введены новые типы диаграмм, которых ранее не было: диаграммы обзора взаимодействия, временные диаграммы и диаграммы составных структур.

Класс в UML изображается в виде прямоугольника, состоящего из трех частей. В верхней части помещается название класса, в средней – свойства объектов класса, в нижней – действия, которые можно выполнять с объектами данного класса (методы).

Между элементами объектной модели существуют различные виды *связей,* которые представляются в модели UML:

- Отношение между классами типа "содержит" (contain) или "состоит из" называется агрегацией, или включением. Такое отношение в терминал языка UML изображается линией с ромбиком на стороне того класса, который выступает в качестве владельца, или контейнера. Необязательное название отношения записывается посередине линии. Например, если аквариум наполнен водой и в нем плавают рыбки, то можно сказать, что аквариум агрегирует в себе воду и рыбок как показано на рис. 13. Агрегация – наиболее сильный тип связи между целым и его частями.

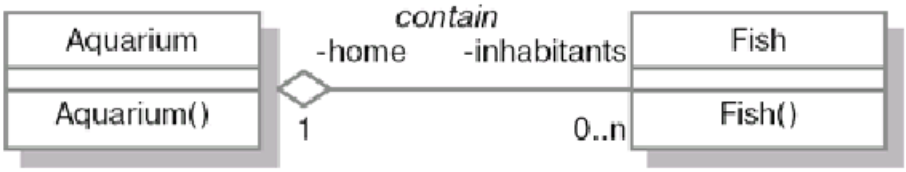


Рис. 13. Пример отношения агрегации между классами

Связи характеризуются: направлением; именем и ролевыми именами участников связи; мощностью. Число объектов, участвующих в отношении, записывается рядом с именем роли. Запись "0..n" означает "от нуля до бесконечности". Приняты также обозначения:

- "1..n" – от единицы до бесконечности;
- "0" – ноль;
- "1" – один;

    - "n" – фиксированное количество;
    - "0..1" – ноль или один.
- Если объекты одного класса ссылаются на один или более объектов другого класса, но ни в ту, ни в другую сторону отношение между объектами не носит характера "владения", или контейнеризации, такое отношение называют ассоциацией (association). Отношение ассоциации изображается так же, как и отношение агрегации, но линия, связывающая классы, простая, без ромбика. Ассоциация – это семантическая связь между классами.
- Зависимость – связь между двумя элементами модели, при которой изменения в спецификации одного элемента могут повлечь за собой изменения в другом элементе.
- Обобщение – связь «тип – подтип».

Диаграммы вариантов использования являются основным элементом разработки и планирования проекта системы и показывают взаимодействия между вариантами использования и действующими лицами, отражая функциональные требования к системе с точки зрения пользователя. Цели моделирования вариантов использования включают выделение требований пользователя таким образом, который может быть одинаково понятен пользователям и разработчикам, позволяя обнаруживать объекты в реальном мире, определять свойства и методы для каждого объекта и устанавливать основу для разработки плана тестирования и руководства пользователя, а также сформировать формализованное описание сценария работы системы как функциональных требований в самом общем виде.

Вариант использования представляет собой последовательность действий (транзакций), выполняемых системой в ответ на событие, инициируемое некоторым внешним объектом (действующим лицом). Вариант использования описывает типичное взаимодействие между пользователем и системой и отражает представление о поведении системы с точки зрения пользователя. В простейшем случае вариант использования определяется в процессе обсуждения с пользователем или представителем заказчика тех функций, которые хотелось бы реализовать, или целей, которые он преследует по отношению к разрабатываемой системе.

Диаграмма вариантов использования является самым общим представлением функциональных требований к системе. Моделирование вариантов использования разбивает полные функциональные возможности системы, определяя все уместные способы использования системы.

Действующее лицо (actor) – это роль, которую пользователь играет по отношению к системе. Действующие лица представляют собой роли, а не конкретных людей или наименования работ. Несмотря на то, что на диаграммах вариантов использования они изображаются в виде стилизованных человеческих фигурок, действующее лицо может также быть внешней системой, которой необходима некоторая информация от данной системы. Показывать на диаграмме действующих лиц следует только в том случае, когда им действительно необходимы некоторые варианты использования.

Один из методов выделения вариантов использования состоит в том, чтобы начать с идентификации актеров, и затем определить бизнес процессы каждого актера, которые он инициирует или в которых участвует. При определении вариантов использования каждый актер способен дать аналитику возможность полностью документировать функциональные возможности системы. Моделирование вариантов использования разбивает полные функциональные возможности системы, определяя все уместные способы использования системы.

Действующие лица делятся на три основных типа – пользователи системы, другие системы, взаимодействующие с данной, и время. Время становится действующим лицом, если от него зависит запуск каких-либо событий в системе.

Рекомендуется, чтобы аналитик рассмотрел:

- Главные задачи актера.
- Требуемый тип доступа актер к информации системы (читать или модифицировать).
- Информацию об изменениях за пределами системы, которую актер должен передавать в систему.
- Информация, которую система должна передавать актеру относительно изменений за пределами системы, известных системе.

Для наглядного представления вариантов использования в качестве основных элементов процесса разработки ПО применяются диаграммы вариантов использования. На рис.14 показан пример такой диаграммы для банкомата (Automated Teller Machine, ATM).

На унифицированном языке моделирования (UML) овалы представляют варианты использования, а фигуры представляют актеров.

На диаграмме показаны два действующих лица: клиент и кредитная система. Существует также шесть основных действий, выполняемых моделируемой системой: перевести деньги, сделать вклад, снять деньги со счета, показать баланс, изменить идентификационный код и осуществить оплату.

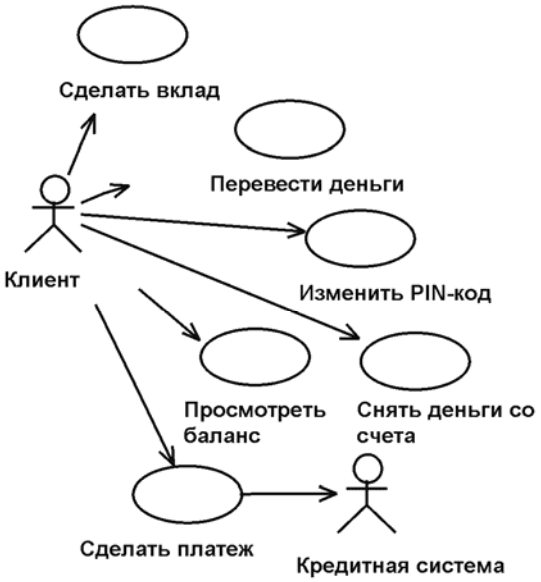


Рис. 14. Пример диаграммы вариантов использования

Для последующего проектирования системы требуются более конкретные детали, которые описываются в документе, называемом "сценарием варианта использования" или "потоком событий" (flow of events). Расширенные варианты использования используются для получения глубокого понимания процессов и требований. В расширенном варианте использования, в дополнение к описанию высокого уровня, записываются цель и обычная или нормальная последовательность действий. Сценарий подробно документирует процесс взаимодействия действующего лица с системой, реализуемого в рамках варианта использования. Основной поток событий описывает нормальный ход событий (при отсутствии отклонений). Альтернативные потоки описывают отклонения от нормального хода событий (ошибочные ситуации) и их обработку.

Альтернативные поведение (наличная оплата, неразрешенная транзакция кредитной карточки) вообще не включаются в вариант использования, потому что цель – понимание основных требований, а не деталей. Легче читать стандартный вариант использования без отвлечения на специальные случаи, и разработчик должен сосредоточиться на наиболее общем случае. Дополнительные детали очерчены и описаны позже.

Достоинства модели вариантов использования заключаются в том, что она:

- определяет пользователей и границы системы – контекст системы;

- определяет системный интерфейс;
- удобна для общения пользователей с разработчиками;
- используется для написания тестов, обеспечивающих проверку работоспособности каждого варианта использования;
- является основой для написания пользовательской документации;
- хорошо вписывается в любые методы проектирования (как объектно-ориентированные, так и структурные).

Диаграммы взаимодействия описывают поведение взаимодействующих групп объектов (в рамках варианта использования или некоторой операции класса). Как только процедуры идентифицированы, изображается диаграмма взаимодействия объектов. Взаимодействие объектов имеет отношение к идентификации работы объектов в рамках системы для удовлетворения требования. Диаграммы взаимодействия представляют объекты и показывают как они связываются друг с другом и взаимодействуют. Как правило, диаграмма взаимодействия охватывает поведение объектов в рамках только одного потока событий варианта использования. На такой диаграмме отображается ряд объектов и те сообщения, которыми они обмениваются между собой.

Сообщение (message) – это средство, с помощью которого объект отправитель запрашивает у объекта получателя выполнение одной из его операций.

Информационное (informative) сообщение – это сообщение, снабжающее объект-получатель некоторой информацией для обновления его состояния.

Сообщение-запрос (interrogative) – это сообщение, запрашивающее выдачу некоторой информации об объекте-получателе.

Императивное (imperative) сообщение – это сообщение, запрашивающее у объекта-получателя выполнение некоторых действий.

Существует два вида диаграмм взаимодействия: диаграммы последовательности (sequence diagrams) и кооперативные диаграммы (collaboration diagrams).

Диаграммы последовательности отражают поток событий, происходящих в рамках варианта использования. Например, вариант использования «Снять деньги» предусматривает несколько возможных последовательностей: снятие денег, попытка снять деньги, не имея их достаточного количества на счету, попытка снять деньги по неправильному идентификационному номеру и некоторые другие. Нормальный сценарий снятия денег со счета (при отсутствии неправильного идентификационного номера или недостатка денег на счете) показан рис. 15.

Эта диаграмма последовательности показывает поток событий в рамках варианта использования «Снять деньги». Все действующие лица показаны в верхней части диаграммы; в приведенном выше примере изображено действующее лицо Клиент. Объекты, требуемые системе длявыполнения варианта использования «Снять деньги», также представлены в верхней части диаграммы. Стрелки соответствуют сообщениям, передаваемым между действующим лицом и объектом или между объектами для выполнения требуемых функций.

На диаграмме последовательности объект изображается в виде прямоугольника, от которого вниз проведена пунктирная вертикальная линия. Эта линия называется линией жизни (lifeline) объекта. Она представляет собой фрагмент жизненного цикла объекта в процессе взаимодействия.

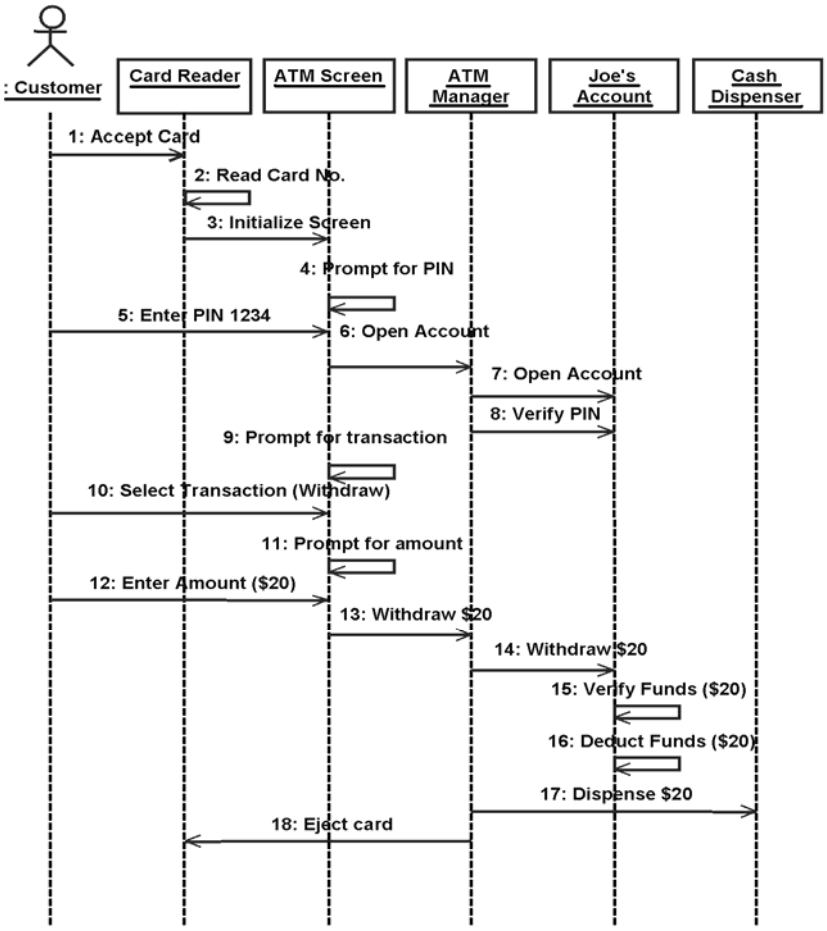


Рис. 15. Диаграмма последовательности снятия клиентом денег со счета

Каждое сообщение представляется в виде стрелки между линиями жизни двух объектов. Сообщения появляются в том порядке, как они показаны на странице сверху вниз. Каждое сообщение помечается как

минимум именем сообщения; при желании можно добавить также аргументы и некоторую управляющую информацию, и, кроме того, можно показать само-делегирование (self-delegation) – сообщение, которое объект посылает самому себе, при этом стрелка сообщения указывает на ту же самую линию жизни и представляет действия внутри объекта.

Хороший способ первоначального обнаружения некоторых объектов – это изучение имен существительных в потоке событий. Можно также изучать документы, описывающие конкретный сценарий – конкретный экземпляр потока событий. Поток событий для варианта использования «Снять деньги» говорит о человеке, снимающем некоторую сумму денег со счета с помощью банкомата.

Не все объекты появляются в потоке событий. Там, например, может не быть форм для заполнения, но их необходимо показать на диаграмме, чтобы позволить действующему лицу ввести новую информацию в систему или просмотреть её. В потоке событий, скорее всего, также не будет и управляющих объектов (control objects). Эти объекты управляют последовательностью потока в варианте использования.

Кооперативныедиаграммы

Вторым видом диаграммы взаимодействия является кооперативная диаграмма.

Подобно диаграммам последовательности, кооперативные диаграммы (collaborations) отображают поток событий через конкретный сценарий варианта использования. Диаграммы последовательности упорядочены по времени, а кооперативные диаграммы больше внимания фокусируют на связях между объектами. На рис. 16 приведена кооперативная диаграмма, описывающая, как клиент снимает деньги со счета.

Как видно из рисунка, здесь представлена вся та информация, которая была и на диаграмме последовательности, но кооперативная диаграмма по-другому описывает поток событий. Из нее легче понять связи между объектами, однако, труднее представить последовательность событий.

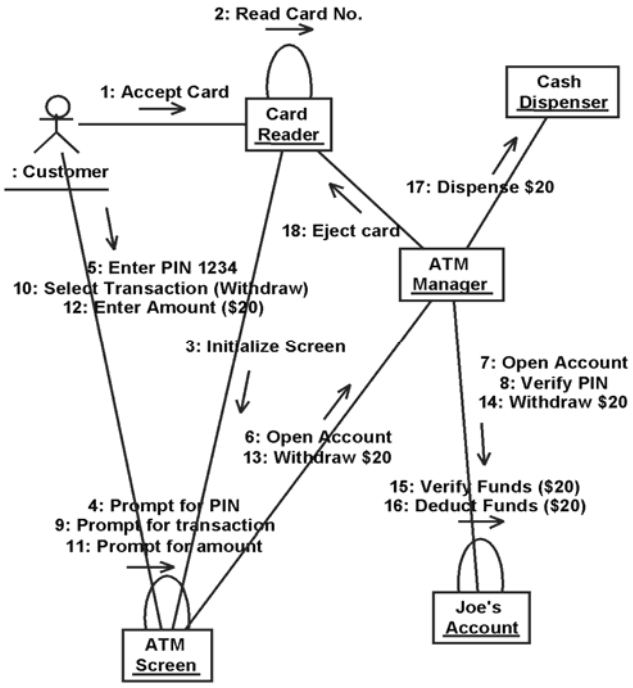


Рис. 16. Кооперативная диаграмма, описывающая процесс снятия клиентом денег со счета

По этой причине часто для какого-либо сценария создают диаграммы обоих типов. Хотя они служат одной и той же цели и содержат одну и ту же информацию, но представляют ее с различных точек зрения.

На кооперативной диаграмме так же, как и на диаграмме последовательности, стрелки обозначают сообщения, обмен которыми осуществляется в рамках данного варианта использования. Их временная последовательность, однако, указывается путем нумерации сообщений.

Диаграмма классов определяет типы классов системы и различного рода статические связи, которые существуют между ними. На диаграммах классов изображаются также атрибуты классов, операции классов и ограничения, которые накладываются на связи между классами. Вид и интерпретация диаграммы классов существенно зависит от точки зрения (уровня абстракции): классы могут представлять сущности предметной области (в процессе анализа) или элементы программной системы (в процессах проектирования и реализации). Пример диаграммы классов представлен на рис. 17.

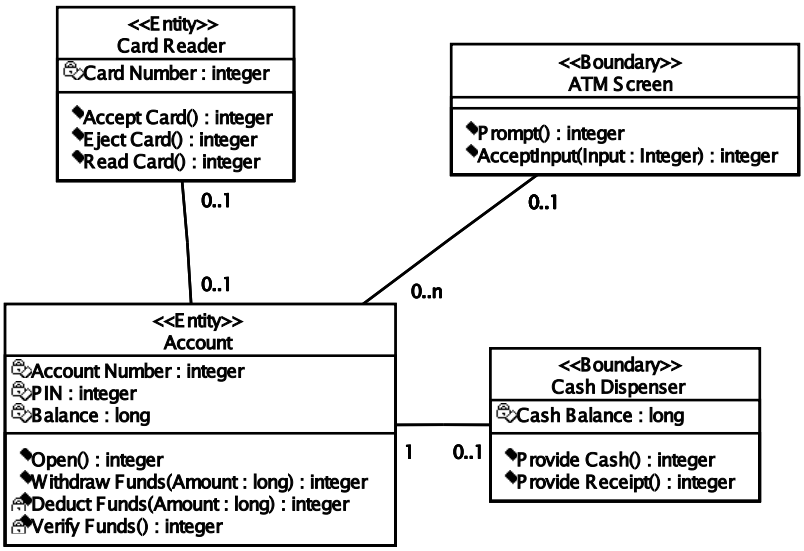


Рис. 17. Пример диаграмма классов

Диаграммы состояний определяют все возможные состояния, в которых может находиться конкретный объект, а также процесс смены состояний объекта в результате наступления некоторых событий. Диаграммы состояний не надо создавать для каждого класса, они применяются только в сложных случаях. Если объект класса может существовать в нескольких состояниях и в каждом из них ведет себя по-разному, для него может потребоваться такая диаграмма. Пример диаграммы состояний представлен на рис. 18.

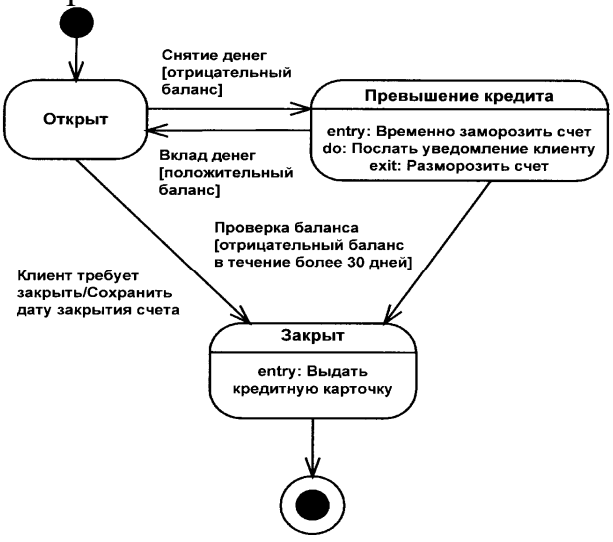


Рис. 18. Пример диаграммы состояний

Диаграммы деятельности, в отличие от большинства других средств UML, заимствуют идеи из нескольких различных методов, в частности, метода моделирования состояний SDL и сетей Петри. Эти диаграммы особенно полезны в описании поведения, включающего большое количество параллельных процессов. Диаграммы деятельности являются также полезными при параллельном программировании, по-

скольку можно графически изобразить все ветви и определить, когда их необходимо синхронизировать.

Диаграммы деятельности можно применять для описания потоков событий в вариантах использования. С помощью текстового описания можно достаточно подробно рассказать о потоке событий, но в сложных и запутанных потоках с множеством альтернативных ветвей будет трудно понять логику событий. Диаграммы деятельности предоставляют ту же информацию, что и текстовое описание потока событий, но в наглядной графической форме.

Диаграммы компонентов моделируют физический уровень системы. На них изображаются компоненты ПО и связи между ними. На такой диаграмме обычно выделяют два типа компонентов: исполняемые компоненты и библиотеки кода.

Каждый класс модели (или подсистема) преобразуется в компонент исходного кода. Между отдельными компонентами изображают зависимости, соответствующие зависимостям на этапе компиляции или выполнения программы.

Диаграммы компонентов применяются теми участниками проекта, кто отвечает за компиляцию и сборку системы. Они нужны там, где начинается генерация кода.

Диаграмма размещения отражает физические взаимосвязи между программными и аппаратными компонентами системы. Она является хорошим средством для того, чтобы показать размещение объектов и компонентов в распределенной системе.

Диаграмма размещения показывает физическое расположение сети и местонахождение в ней различных компонентов. Ее основными элементами являются узел (вычислительный ресурс) и соединение – канал взаимодействия узлов (сеть).

Диаграмма размещения используется менеджером проекта, пользователями, архитектором системы и эксплуатационным персоналом, чтобы представить физическое размещение системы и расположение ее отдельных подсистем. Пример диаграммы размещения приведен на рис. 19.

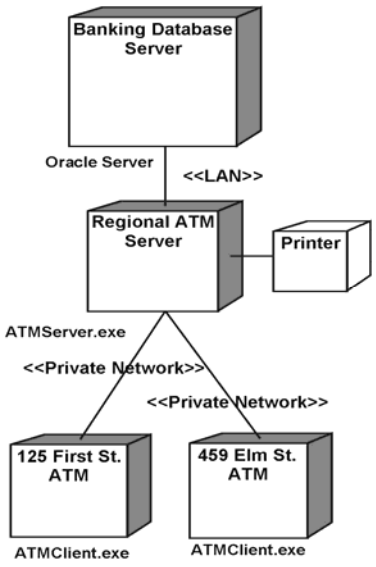


Рис. 19. Пример диаграммы размещения

UML обладает механизмами расширения, предназначенными для того, чтобы разработчики могли адаптировать язык моделирования к своим конкретным нуждам, не меняя при этом его метамодель. Наличие механизмов расширения принципиально отличает UML от таких средств моделирования, как IDEF0, IDEF1X, IDEF3, DFD и ERM.

Сравнительный анализ структурного и объектно-ориентированного подходов

Гради Буч сформулировал главное достоинство объектно-ориентированного подхода (ООП) следующим образом: объектно-ориентированные системы более открыты и легче поддаются внесению изменений, поскольку их конструкция базируется на устойчивых формах. Это дает возможность системе развиваться постепенно и не приводит к полной ее переработке даже в случае существенных изменений исходных требований.

Он отметил также ряд следующих преимуществ ООП:

объектная декомпозиция дает возможность создавать программные системы меньшего размера путем использования общих механизмов, обеспечивающих необходимую экономию выразительных средств. Использование ООП существенно повышает уровень унификации разработки и пригодность для повторного использования не только ПО, но и проектов, что, в конце концов, ведет к сборочному созданию ПО. Системы зачастую получаются более компактными, чем их не объектно-ориентированные эквиваленты, что означает не только уменьшение

объема программного кода, но и удешевление проекта за счет использования предыдущих разработок;

- объектная декомпозиция уменьшает риск создания сложных систем ПО, которые предполагают эволюционный путь развития на базе относительно небольших подсистем. Процесс интеграции системы растягивается на все время разработки, а не превращается в единовременное событие;
- объектная модель вполне естественна, поскольку в первую очередь ориентирована на человеческое восприятие мира, а не на компьютерную реализацию;
- объектная модель позволяет в полной мере использовать выразительные возможности объектных и объектно-ориентированных языков программирования.

К недостаткам ООП относятся некоторое снижение производительности функционирования ПО (которое, однако, по мере роста производительности компьютеров становится все менее критичным) и высокие начальные затраты. Объектная декомпозиция существенно отличается от функциональной, поэтому переход на новую технологию связан как с преодолением психологических трудностей, так и дополнительными финансовыми затратами. При переходе от структурного подхода к объектному, как при всякой смене технологии, необходимо вкладывать деньги в приобретение новых инструментальных средств. Здесь следует учесть расходы на обучение методу, инструментальным средствам и языку программирования. Для некоторых организаций эти обстоятельства могут стать серьезными препятствиями.

Объектно-ориентированный подход не дает немедленной отдачи. Эффект от его применения начинает сказываться после разработки двух-трех проектов и накопления повторно используемых компонентов, отражающих типовые проектные решения в данной области. Переход организации на объектно-ориентированную технологию – это смена мировоззрения, а не просто изучение новых CASE-средств и языков программирования.

Таким образом, структурный подход по-прежнему сохраняет свою значимость и достаточно широко используется на практике. На примере языка UML хорошо видно, что его авторы заимствовали то рациональное, что можно было взять из структурного подхода: элементы функциональной декомпозиции в диаграммах вариантов использования, диаграммы состояний, диаграммы деятельности и др. Очевидно, что в конкретном проекте сложной системы невозможно обойтись только одним способом декомпозиции. Можно начать декомпозицию каким-либо од-

ним способом, а затем, используя полученные результаты, попытаться рассмотреть систему с другой точки зрения.

Основой взаимосвязи между структурным и объектно-ориентированным подходами является общность ряда категорий и понятий обоих подходов (контекстная модель и вариант использования, сущность и класс и др.). Эта взаимосвязь может проявляться в различных формах. Так, одним из возможных вариантов является использование структурного анализа как основы для объектно-ориентированного проектирования. При этом структурный анализ следует прекращать, как только структурные модели начнут отражать не только деятельность организации (бизнес-процессы), а и архитектуру ПО. После выполнения структурного анализа можно различными способами приступить к определению классов и объектов. Так, если взять какую-либо отдельную диаграмму потоков данных, то кандидатами в классы могут быть элементы структур данных.

Другой формой проявления взаимосвязи можно считать интеграцию объектной и реляционной технологий. Реляционные СУБД являются на сегодняшний день основным средством реализации крупномасштабных баз данных и хранилищ данных. Причины этого достаточно очевидны: реляционная технология используется достаточно долго, освоена огромным количеством пользователей и разработчиков, стала промышленным стандартом, в нее вложены значительные средства и создано множество корпоративных БД в самых различных отраслях, реляционная модель проста и имеет строгое математическое основание; существует большое разнообразие промышленных средств проектирования, реализации и эксплуатации реляционных БД. Вследствие этого реляционные БД в основном используются для хранения и поиска объектов в так называемых объектно-реляционных системах.

Взаимосвязь между структурным и объектно-ориентированным подходами достаточно четко просматривается в различных ТЕХНОЛОГИИХ СОЗДАНИЯ ПО.

Вопросы для самопроверки

1. Дайте определение модели системы.
2. В чем состоит основное предназначение визуальной (графической) модели системы?
3. Какой состав моделей рассматривается в методологии структурного анализа и проектирования?
4. Для какого класса ИС успешно используется SADT
   а) Для систем динамически изменяющихся
   б) Любого рода ИС

в) С хорошо определенными регламентами бизнес-процессов

5. Охарактеризуйте модель IDEF0. Что представляет данная модель?
6. Охарактеризуйте моделирование IDEF3.
7. Какие элементы системы моделируются в диаграмме потоков данных? Назовите 3 используемых символа в процессе моделирования DFD.
8. Перечислите этапы создания модели ER-диаграммы.
9. В чем состоят причины возникновения объектно-ориентированного подхода в программировании?
10. Перечислите принципы и понятия объектно-ориентированного подхода анализа и проектирования.
11. Перечислите типы связей, возникающие между классами в объектно-ориентированной модели.
12. В чем состоит назначение диаграммы вариантов использования?
13. Какие два вида диаграмм взаимодействия применяются в в объектно-ориентированной модели?

Методы моделирования бизнес-процессов и спецификации требований

Моделирование бизнес-процессов является важной составной частью проектов по созданию крупномасштабных систем ПО. Отсутствие таких моделей является одной из главных причин неудач многих проектов.

Назначением будущего ПО является, в первую очередь, решение проблем бизнеса. Требования к ПО формируются на основе бизнес-модели, а критерии проектирования систем, прежде всего, основываются на наиболее полном их удовлетворении.

Модели бизнес-процессов являются не просто промежуточным результатом, используемым консультантом для выработки каких-либо рекомендаций и заключений, а представляют собой самостоятельный результат, имеющий большое практическое значение.

На сегодняшний день в моделировании бизнес-процессов преобладает процессный подход. Его основной принцип заключается в структурировании деятельности организации в соответствии с ее бизнес-процессами, а не организационно-штатной структурой. Модель, основанная на организационно-штатной структуре, может продемонстрировать лишь хаос, царящий в организации (о котором в принципе руководству и так известно, иначе оно бы не инициировало соответствующие работы), на ее основе можно только внести предложения об изменении этой структуры. С другой стороны, модель, основанная на бизнес-

процессах, может содержать в себе и организационно структуру предприятия.

Процессный подход может использовать любые из перечисленных выше средств моделирования. Однако в настоящее время наблюдается тенденция интеграции разнообразных методов моделирования и анализа систем, проявляющаяся в форме создания интегрированных средств моделирования. Одним из таких средств является продукт, носящий название **ARIS – Architecture of Integrated Information System**, разработанный германской фирмой IDS Scheer.

Система ARIS представляет собой комплекс средств моделирования и анализа деятельности предприятия. Ее методическую основу составляет совокупность различных методов моделирования, отражающих разные взгляды на исследуемую систему. Одна и та же модель может разрабатываться с использованием нескольких методов, что позволяет использовать ARIS специалистам с различными теоретическими знаниями и настраивать его на работу с системами, имеющими свою специфику.

ARIS поддерживает четыре типа моделей, отражающих различные аспекты исследуемой системы:

- организационные модели, представляющие структуру системы – иерархию организационных подразделений, должностей и конкретных лиц, связи между ними, а также территориальную привязку структурных подразделений;
- функциональные модели, содержащие иерархию целей, стоящих перед организационной структурой управления, с совокупностью деревьев функций, необходимых для достижения поставленных целей;
- информационные модели, отражающие структуру информации, необходимой для реализации всей совокупности функций системы;
- модели управления, представляющие комплексный взгляд на реализацию бизнес-процессов в рамках системы.

Для построения перечисленных типов моделей используются как собственные методы моделирования ARIS, так и различные известные методы и языки моделирования – ERM, UML, OMT и др.

В процессе моделирования каждый аспект деятельности предприятия сначала рассматривается отдельно, а после детальной проработки всех аспектов строится интегрированная модель, отражающая все связи между различными аспектами.

Модели в ARIS представляют собой диаграммы, элементами которых являются разнообразные объекты – "функция", "событие", "структурное подразделение", "документ" и т. п. Между объектами устанавливаются разнообразные связи. Каждому объекту соответствует определенный набор атрибутов, которые позволяют ввести дополнительную информацию о конкретном объекте. Значения атрибутов могут использоваться при имитационном моделировании или для проведения стоимостного анализа.

Основная бизнес-модель ARIS – eEPC (extended Event Driven Process Chain – расширенная модель цепочки процессов, управляемых событиями). По существу, она расширяет возможности IDEF0, IDEF3 и DFD, обладая всеми их достоинствами и недостатками. Применение большого числа различных объектов, связанных различными типами связей, может значительно увеличить размер модели и сделать ее плохо читаемой.

Бизнес-процесс в нотации eEPC представляет собой поток последовательно выполняемых работ (процедур, функций), расположенных в порядке их выполнения. Реальная длительность выполнения процедур в eEPC визуально не отражается. Это приводит к тому, что при создании моделей возможны ситуации, когда на одного исполнителя будет возложено выполнение двух задач одновременно. Используемые при построении модели символы логики позволяют отразить ветвление и слияние бизнес-процесса. Для получения информации о реальной длительности процессов необходимо использовать другие инструменты описания, например графики Ганта в системе MS Project.

Ряд современных методов моделирования бизнес-процессов основан на использовании языка UML. Хотя UML изначально предназначался для моделирования систем ПО, его использование в другой области стало возможным благодаря наличию в UML механизмов расширения (стереотипов).

Среди таких методов наиболее известными являются метод Ericsson-Penker и метод, реализованный в технологии Rational Unified Process (RUP).

Метод Ericsson-Penker представляет интерес, прежде всего, в связи с попыткой применения UML в рамках процессного подхода к моделированию бизнес-процессов. Авторы метода создали свой профиль UML для моделирования бизнес-процессов, введя набор стереотипов, описывающих процессы, ресурсы, правила и цели деятельности организации. Метод использует четыре основные категории бизнес-модели:

Ресурсы – различные объекты, используемые или участвующие в бизнес-процессах (люди, материалы, информация или продукты).

Процессы – виды деятельности, изменяющие состояние ресурсов в соответствии с бизнес-правилами.

Цели – назначение бизнес-процессов Цели могут быть разбиты на подцели и соотнесены с отдельными процессами.

Бизнес-правила – условия или ограничения выполнения процессов (функциональные, поведенческие или структурные). Правила могут быть определены с использованием языка OCL.

Основной диаграммой UML, используемой в данном методе, является диаграмма деятельности. Метод Eriksson-Penker представляет процесс в виде деятельности со стереотипом "process" (основой данного представления является расширение метода IDEF0). Полная бизнес-модель включает множество представлений, подобных представлениям архитектуры ПО. Каждое представление выражено в одной или более диаграммах UML. Диаграммы могут иметь различные типы и изображать процессы, правила, цели и ресурсы во взаимодействиях друг с другом. Метод использует четыре различных представления бизнес-модели:

- концептуальное представление – структура целей и проблем;
- представление процессов – взаимодействие между процессами и ресурсами (в виде набора диаграмм деятельности);
- структурное представление – структура организации и ресурсов (в виде диаграмм классов);
- представление поведения – поведение отдельных ресурсов и детализация процессов (в виде диаграмм деятельности, состояний и взаимодействия).

Методика моделирования RUP предусматривает построение двух моделей:

- модели бизнес-процессов (Business Use Case Model);
- модели бизнес-анализа (Business Analysis Model).

Модель бизнес-процессов представляет собой расширение модели вариантов использования UML за счет введения набора стереотипов – Business Actor (стереотип действующего лица) и Business Use Case (стереотип варианта использования с точки зрения бизнес-процессов). Business Actor (действующее лицо бизнес-процессов) – это некоторая роль, внешняя по отношению к бизнес-процессам организации. Business Use Case определяется как описание последовательности действий (потока событий) в рамках некоторого бизнес-процесса, приносящего ощутимый результат конкретному действующему лицу. Это определение подобно общему определению бизнес-процесса, но имеет более точный смысл. В терминах объектной модели Business Use Case представляет

собой класс, объектами которого являются конкретные потоки событий в рамках описываемого бизнес-процесса.

Описание Business Use Case может сопровождаться целью процесса, которая, так же, как и в методе Eriksson-Penker, моделируется с помощью класса со стереотипом "goal", а дерево целей изображается в виде диаграммы классов.

Для каждого Business Use Case строится модель бизнес-анализа – объектная модель, описывающая реализацию бизнес-процесса в терминах взаимодействующих объектов (бизнес-объектов – Business Object), принадлежащих к двум классам – Business Worker и Business Entity. Business Worker (исполнитель) – класс, представляющий собой абстракцию исполнителя, выполняющего некоторые действия в рамках бизнес-процесса. Исполнители взаимодействуют между собой и манипулируют различными сущностями, участвуя в реализациях сценариев Business Use Case. Business Entity (сущность) является объектом различных действий со стороны исполнителей.

Кроме диаграммы данных классов, модель бизнес-анализа может включать:

- Диаграммы последовательности (и кооперативные диаграммы), описывающие сценарии Business Use Case в виде последовательности обмена сообщениями между объектами-действующими лицами и объектами-исполнителями. Такие диаграммы помогают явно определить в модели обязанности каждого исполнителя в виде набора его операций.
- Диаграммы деятельности, описывающие взаимосвязи между сценариями одного или различных Business Use Case.
- Диаграммы состояний, описывающие поведение отдельных бизнес-объектов.

Методика моделирования Rational Unified Process обладает следующими достоинствами:

- модель бизнес-процессов строится вокруг участников процессов (заинтересованных лиц) и их целей, помогая выявить все потребности клиентов организации. Такой подход в наибольшей степени применим для организаций, работающих в сфере оказания услуг (торговые организации, банки, страховые компании и т. д.);
- моделирование на основе вариантов использования способствует хорошему пониманию бизнес-модели со стороны заказчиков.

Однако следует отметить, что при моделировании деятельности крупной организации, занимающейся как производством продукции,

так и оказанием услуг, необходимо применять различные методики моделирования, поскольку для моделирования производственных процессов более предпочтительным является процессный подход (например, метод Eriksson-Penker).

Спецификация требований к ПО является составной частью процесса управления требованиями. Выявленные в результате применения перечисленных методов требования к ПО оформляются в виде ряда документов и моделей. Так, в технологии RUP функциональные требования к системе моделируются и документируются с помощью вариантов использовании. Стиль их написания зависит от масштаба, количества участников и критичности проекта.

Спецификация требований в RUP не требует обязательного моделирования бизнес-процессов организации, для которых создается ПО, однако, наличие бизнес-моделей существенно упрощает построение системной модели вариантов использования.

Вопросы для самопроверки

1. Какие 4 типа моделей поддерживает ARIS?
2. Дайте характеристику основной бизнес модели ARIS-eEPC.
3. Средствами какого языка выполняется моделирование метода Ericsson-Penker? Дайте характеристику этого метода.
4. Построение каких моделей анализа предусматривает методика моделирования RUP? Охарактеризуйте эти модели.

Методы анализа и проектирования ПО

Целью анализа требований является трансформация функциональных требований к ПО в предварительный системный проект и создание стабильной основы архитектуры системы. В процессе проектирования системный проект "погружается" в среду реализации с учетом всех нефункциональных требований.

Все современные ТС ПО реализуют ту или иную методику анализа и проектирования ПО. Одна из типичных методик ООАП реализована в технологии RUP. Согласно этой методике, объектно-ориентированный анализ включает два вида деятельности: архитектурный анализ и анализ вариантов использования. Архитектурный анализ выполняется архитектором системы и включает в себя:

- утверждение общих стандартов (соглашений) моделирования и документирования системы;
- предварительное выявление архитектурных механизмов (надежности, безопасности и т. д.);

- формирование набора основных абстракций предметной области (классов анализа);
- формирование начального представления архитектурных уровней.

Анализ вариантов использования выполняется проектировщиками и включает в себя:

- идентификацию классов, участвующих в реализации потоков событий варианта использования;
- распределение поведения, реализуемого вариантом использования, между классами (определение обязанностей классов);
- определение атрибутов и ассоциаций классов.

В потоках событий варианта использования выявляются классы трех типов:

- Граничные классы (Boundary) – служат посредниками при взаимодействии внешних объектов с системой. Типы граничных классов: пользовательский интерфейс (обмен информацией с пользователем, без деталей интерфейса – кнопок, списков, окон), системный интерфейс и аппаратный интерфейс (используемые протоколы, без деталей их реализации).
- Классы-сущности (Entity) – представляют собой основные абстракции (понятия) разрабатываемой системы, рассматриваемые в рамках конкретного варианта использования.
- Управляющие классы (Control) – обеспечивают координацию поведения объектов в системе. Примеры управляющих классов: менеджер транзакций, координатор ресурсов, обработчик ошибок.

Классы анализа отражают функциональные требования к системе и моделируют объекты предметной области. Совокупность классов анализа представляет собой начальную концептуальную модель системы

Наиболее важной частью объектно-ориентированного анализа является распределение обязанностей между классами (в виде операций классов). Оно выполняется с помощью диаграмм взаимодействия. При построении диаграмм взаимодействия возникают проблемы правильного распределения обязанностей между классами. Для их решения существует ряд образцов.

Атрибуты классов анализа определяются, исходя из знаний о предметной области и требований к системе. Связи между классами (ассоциации) определяются на основе анализа кооперативных диаграмм, затем анализируются и уточняются.

Целью объектно-ориентированного проектирования является адаптация предварительного системного проекта (набора классов "анализа"), составляющего стабильную основу архитектуры системы, к среде реализации с учетом всех нефункциональных требований.

Объектно-ориентированное проектирование включает два вида деятельности:

- проектирование архитектуры системы;
- проектирование элементов системы.

Проектирование архитектуры системы выполняется архитектором системы и включает в себя:

- идентификацию архитектурных решений и механизмов, необходимых для проектирования системы;
- анализ взаимодействий между классами анализа, выявление подсистем и интерфейсов;
- формирование архитектурных уровней;
- проектирование конфигурации системы.

Проектирование элементов системы включает в себя:

- проектирование классов (детализация классов, уточнение операций и атрибутов, моделирование состояний, уточнение связей между классами);
- проектирование баз данных (в зависимости от типа используемой для хранения данных СУБД – объектной или реляционной).

Вопросы для самопроверки

1. Приведите примеры функциональных и нефункциональных требований. В чем тх отличие?
2. Что включает в себя архитектурный анализ ООАП? Приведите пример выполнения архитектурного анализа применительно к одной из задач банковского сектора.
3. Что включает в себя анализ вариантов использования в ООАП?
4. Что является наиболее важной частью объектно-ориентированного анализа?
5. Какие задачи решаются архитектором системы в рамках проектирования архитектуры? Приведите пример архитектуры системы применительно к одной из задач банковского сектора.
6. Что включает проект элементов системы?

Технологии создания ПО

Технологии создания программного обеспечения (ТС ПО) в общем случае можно описать следующей системой понятий:

ТС ПО – упорядоченная совокупность взаимосвязанных технологических процессов, обеспечивающая поддержку этапов жизненного цикла разработки системы (ЖЦ РС).

Технологический процесс – совокупность взаимосвязанных технологических операций для выполнения отдельных этапов жизненного цикла разработки системы.

Технологическая операция – основная единица работы, выполняемая определенной ролью, которая:

- подразумевает четко определенную ответственность роли;
- дает четко определенный результат (набор рабочих продуктов), базирующийся на определенных исходных данных (другом наборе рабочих продуктов);
- представляет собой единицу работы с жестко определенными границами, которые устанавливаются при планировании проекта.

Рабочий продукт – информационная или материальная сущность, которая создается, модифицируется или используется в некоторой технологической операции (модель, документ, код, тест и т. п.). Рабочий продукт определяет область ответственности роли и является объектом управления конфигурацией.

Роль – определение поведения и обязанностей отдельного лица или группы лиц в среде организации-разработчика ПО, осуществляющих деятельность в рамках некоторого технологического процесса и ответственных за определенные рабочие продукты.

Руководство – практическое руководство по выполнению одной или совокупности технологических операций. Руководства включают методические материалы, инструкции, нормативы, стандарты и критерии оценки качества рабочих продуктов.

Инструментальное средство (CASE-средство) – программное средство, обеспечивающее автоматизированную поддержку деятельности, выполняемой в рамках технологических операций.

Требования, предъявляемые к ТС ПО

Основным требованием, предъявляемым к современным ТС ПО, является их **соответствие стандартам и нормативным документам**, связанным с процессами ЖЦ ПО и оценкой технологической зрелости организаций-разработчиков (ISO 12207, ISO 9000, CMM и др.). Соглас-

но этим нормативам, ТС ПО должна поддерживать следующие процессы:

- управление требованиями;
- анализ и проектирование ПО;
- разработка ПО;
- эксплуатация;
- сопровождение;
- документирование;
- управление конфигурацией и изменениями;
- тестирование;
- управление проектом.

Полнота поддержки процессов ЖЦРС должна поддерживаться комплексом инструментальных средств (CASE-средств).

Соответствие стандартам означает также, в частности, использование общепринятых, стандартных нотаций и соглашений. Для того чтобы проект мог выполняться разными коллективами разработчиков, необходимо использование стандартных методов моделирования и стандартных нотаций, которые должны быть оформлены в виде нормативов до начала процесса проектирования. Несоблюдение проектных стандартов ставит разработчиков в зависимость от фирмы-производителя данного средства, делает затруднительным формальный контроль корректности проектных решений и снижает возможности привлечения дополнительных коллективов разработчиков, смены исполнителей и отчуждения проекта, поскольку число специалистов, знакомых с данным методом (нотацией), может быть ограниченным.

Другим важным требованием является **адаптируемость к условиям применения**, которая достигается за счет поставки технологии в электронном виде вместе с CASE-средствами и библиотеками процессов, шаблонов, методов, моделей и других компонентов, предназначенных для построения ПО того класса систем, на который ориентирована технология. Электронные технологии должны включать средства, обеспечивающие их адаптацию и развитие по результатам выполнения конкретных проектов. Процесс адаптации заключается в удалении ненужных процессов и действий ЖЦ ПО, в изменении неподходящих или в добавлении собственных процессов и действий, а также методик, стандартов и руководств.

Внедрение в технологии создания ПО в организации

При внедрении ТС ПО следует руководствоваться рекомендациями, приведенными в стандартах. Эти рекомендации достаточно акту-

альны и ценны, поскольку отражают опыт, накопленный многими зарубежными пользователями и разработчиками ТС ПО в течение длительного периода их существования.

Термин "внедрение" используется в широком смысле и включает все действия – от оценки первоначальных потребностей до полномасштабного использования ТС ПО в различных подразделениях организации. Процесс внедрения ТС ПО состоит из следующих этапов:

1. Определение потребностей в ТС ПО, характеристик объекта внедрения и проектов создания ПО.
2. Определение требований, предъявляемых к ТС ПО (анализ характеристик объекта внедрения и проектов, обоснование требований к ТС ПО, определение приоритетов требований).
3. Оценка вариантов ТС ПО. Предварительная экспертная оценка заключается в анализе доступных ТС ПО на предмет соответствия требованиям, неудовлетворительные варианты (с точки зрения реализации наиболее приоритетных требований) отвергаются, формируется список претендентов. При детализированной оценке для каждой ТС ПО-претендента формируется ее детальное описание. Источники информации для описания – техническая документация поставщика, доступные данные о реальных внедрениях, результаты выполнения пилотных проектов.
4. Выбор ТС ПО. Производится сравнительный анализ технологий и окончательный выбор ТС ПО с помощью экспертной оценки.
5. Адаптация ТС ПО к условиям применения. Производится формирование конкретной рабочей конфигурации ТС ПО, адаптированной к условиям объекта внедрения.

В процессе внедрения ТС ПО собирается статистика и оценивается эффективность ее внедрения с точки зрения ряда критериев (минимум трудоемкости сопровождения ПО, минимум затрат на сопровождение ПО и др.). При изменении условий объекта внедрения и по результатам анализа эффективности внедрения ТС ПО принимается решение: а) о внесении изменений в рабочую конфигурацию ТС ПО; б) о переходе на новую ТС ПО. В случае перехода повторяются пп. 3–5.

Оценка и выбор ТС ПО

Цель процесса оценки – определение функциональности и качества ТС ПО для последующего выбора. Оценка выполняется в соответствии с конкретными критериями, ее результаты включают как объективные, так и субъективные данные по каждой ТС ПО.

Процессы оценки и выбора тесно взаимосвязаны. По результатам оценки цели выбора и/или критерии выбора и их веса могут потребовать модификации. В таких случаях может понадобиться повторная оценка. Когда анализируются окончательные результаты оценки и к ним применяются критерии выбора, может быть рекомендовано приобретение технологии. Альтернативой может быть отсутствие адекватных технологий, в этом случае рекомендуется разработать новую технологию, модифицировать существующую или отказаться от внедрения.

Процесс выбора включает в себя следующие действия:

- формулировка задач выбора, включая цели, предположения и ограничения;
- выполнение всех необходимых действий по выбору, включая определение и ранжирование критериев, определение технологий-кандидатов, сбор необходимых данных и применение ранжированных критериев к результатам оценки для определения средств с наилучшими показателями;
- выполнение необходимого количества итераций с тем, чтобы выбрать (или отвергнуть) технологии, имеющие сходные показатели.

Типичный процесс оценки и/или выбора может использовать набор критериев различных типов. Каждый критерий должен быть выбран и адаптирован экспертом с учетом особенностей конкретного процесса.

Исходные данные для оценки и выбора – набор параметров (технико-экономических характеристик) ТС ПО:

1. Функциональные характеристики, ориентированные на процессы жизненного цикла ПО (управление проектом, управление требованиями, управление конфигурацией и изменениями, анализ и проектирование ПО и др.).
2. Функциональные характеристики применения (среда функционирования, совместимость с другими ТС ПО, соответствие технологическим стандартам).
3. Характеристики качества (надежность, удобство использования, эффективность, сопровождаемость, переносимость).
4. Общие характеристики (затраты на технологию, лицензионная политика, оценочный эффект от внедрения ТС ПО, инфраструктура, требуемая для внедрения ТС ПО, доступность и качество обучения, сертификация поставщика, поддержка поставщика).

На основе данного набора параметров анализируются и классифицируются существующие ТС ПО. Общий набор критериев, применяемых для оценки ТС ПО, приведен в таблице.

В результате выполненной оценки может оказаться, что ни одна доступная технология не удовлетворяет в нужной мере всем критериям

и не покрывает все потребности проекта. В этом случае может применяться набор средств, позволяющий построить на их базе единую технологическую среду.

Таблица

Критерий	Определение
Минимум трудоемкости создания ПО	Количество человеко-месяцев, затрачиваемых на создание ПО с использованием ТС ПО
Максимум продуктивности	Объем работы (измеряемый в количестве строк кода или функциональных точек), приходящийся на единицу трудоемкости (человеко-месяц) при использовании данной ТС ПО
Максимум качества создаваемого ПО	Количество дефектов в рабочих продуктах при использовании данной ТС ПО
Возврат инвестиций	(Доход от использования ПО – Затраты на создание и сопровождение ПО) / (Затраты на создание и сопровождение ПО)
Минимум затрат на сопровождение ПО	Отношение стоимости сопровождения ПО при использовании данной ТС ПО к совокупным затратам на информационные ТС ПО в организации
Минимум времени внедрения ТС ПО	Временной интервал от начала внедрения ТС ПО до выхода на безубыточный уровень (начало возврата инвестиций в ТС ПО)
Минимум затрат на внедрение ТС ПО	Суммарная стоимость приобретения, обучения и сопровождения ТС ПО
Минимальный срок окупаемости затрат на внедрение ТС ПО	Временной интервал от начала внедрения ТС ПО до полной окупаемости затрат на ее внедрение

Выполнение пилотного проекта

Перед полномасштабным внедрением выбранной ТС ПО в организации выполняется пилотный проект, целью которого является экспериментальная проверка правильности решений, принятых на предыдущих этапах, и подготовка к внедрению.

Пилотный проект позволяет получить важную информацию, необходимую для оценки ТС ПО и его поддержки со стороны поставщика после того, как средство установлено.

Важной функцией пилотного проекта является принятие решения относительно приобретения или отказа от использования ТС ПО. Провал пилотного проекта позволяет избежать более значительных и доро-

гостоящих неудач в дальнейшем, поскольку пилотный проект обычно требует приобретения относительно небольшого количества лицензий и обучения узкого круга специалистов.

Пилотный проект должен обладать следующими характеристиками:

- **Типичность предметной области.** Чтобы облегчить окончательное определение области применения ТС ПО, предметная область пилотного проекта должна быть типичной для обычной деятельности организации. Пилотный проект должен помочь определить любую дополнительную технологию, обучение или поддержку, которые необходимы для перехода от пилотного проекта к широкомасштабному использованию ТС ПО. В рамках этих ограничений пилотный проект должен иметь небольшой, но значимый размер.
- **Масштабируемость.** Результаты, полученные в пилотном проекте, должны показать масштабируемость ТС ПО. Цель – получить четкое представление о масштабах проектов, для которых данная ТС ПО применима.
- **Представительность.** Пилотный проект не должен быть необычным или уникальным для организации. ТС ПО должна использоваться для решения задач, относящихся к предметной области, хорошо понимаемой всей организацией.
- **Критичность.** Пилотный проект должен иметь существенную значимость, чтобы оказаться в центре внимания, но не должен быть критичным для успешной деятельности организации в целом.
- **Авторитетность.** Группа специалистов, участвующих в проекте, должна обладать высоким авторитетом, при этом результаты проекта будут всерьез восприняты остальными сотрудниками организации.
- **Готовность проектной группы.** Проектная группа должна обладать готовностью к нововведениям, технической зрелостью и приемлемым уровнем опыта и знаний в данной ТС ПО и предметной области. С другой стороны, группа должна отражать в миниатюре характеристики организации в целом.

В процессе оценки пилотного проекта организация должна определить свою позицию по следующим трем вопросам:

- Целесообразно ли внедрять ТС ПО?
- Какие конкретные особенности пилотного проекта привели к его успеху (или неудаче)?

- Какие проекты или подразделения в организации могли бы получить выгоду от использования ТС ПО?

  Возможным решением должно быть одно из следующих:
- Внедрить ТС ПО. В этом случае рекомендуемый масштаб внедрения должен быть определен в терминах структурных подразделений и предметной области.
- Выполнить дополнительный пилотный проект. Такой вариант должен рассматриваться только в том случае, если остались конкретные неразрешенные вопросы относительно внедрения ТС ПО в организации.
- Отказаться от ТС ПО. В этом случае причины отказа от конкретной ТС ПО должны быть определены в терминах потребностей организации или критериев, которые остались неудовлетворенными. Перед тем как продолжить деятельность по внедрению ТС ПО, потребности организации должны быть пересмотрены на предмет своей обоснованности.
- Отказаться от использования ТС ПО вообще. Пилотный проект может показать, что организация либо не готова к внедрению ТС ПО, либо автоматизация данного аспекта процесса создания и сопровождения ПО не дает никакого эффекта для организации.

Практическое внедрение ТС ПО

Процесс перехода к практическому использованию ТС ПО начинается с разработки и последующей реализации плана перехода. Этот план может отражать поэтапный подход к переходу – от тщательно выбранного пилотного проекта до проектов с существенно возросшим разнообразием характеристик.

План перехода должен охватывать:

- информацию относительно целей, критериев оценки, графика и возможных рисков, связанных с реализацией плана;
- информацию относительно приобретения, установки и настройки ТС ПО;
- информацию относительно интеграции с существующими средствами, включая как интеграцию средств друг с другом, так и их интеграцию в процессы разработки и эксплуатации ПО, существующие в организации;
- ожидаемые потребности в обучении и ресурсы, используемые в течение и после завершения процесса перехода;
- определение стандартных процедур использования ТС ПО.

План перехода должен определять начальную практику применения и процедуры использования средств. Реальное применение любой ТС ПО в конкретной организации и конкретном проекте невозможно без выработки ряда стандартов (правил, соглашений), которые должны соблюдаться всеми участниками проекта (это особенно актуально при коллективной разработке ПО большим количеством групп специалистов). К таким стандартам относятся следующие:

- руководства по моделированию и проектированию;
- соглашения по присвоению имен;
- процедуры контроля качества и процессов приемки, включая расписание экспертиз и используемые методологии;
- процедуры резервного копирования, защиты мастер-копий и конфигурирования базы данных проекта;
- процедуры интеграции с существующими средствами и базами данных;
- процедуры совместного использования данных и контроля целостности БД;
- стандарты и процедуры обеспечения секретности;
- стандарты документирования.
- стандарт проектирования;
- стандарт оформления проектной документации;
- стандарт интерфейса пользователя.

Для успешного внедрения ТС ПО в организации существенной является последовательность в ее применении. Поскольку большинство систем разрабатываются коллективно, необходимо определить характер будущего использования ТС ПО как отдельными разработчиками, так и группами. Использование стандартных процедур позволит обеспечить плавный переход между отдельными стадиями ЖЦ ПО.

Результатом данного этапа является внедрение ТС ПО в повседневную практику организации. Кроме того, поддержка ТС ПО включается в план текущей поддержки ПО в данной организации.

Вопоосы для самопроверки

1. Какими понятиями описывается ТС ПО?
2. Перечислите процессы, которые должна поддерживать ТС ПО.
3. Перечислите требования, которым должна удовлетворять ТС ПО.
4. Перечислите этапы, которые выполняются в процессе внедрения ТС ПО.
5. Перечислите критерии оценки выбора ТС ПО.
6. Назначение и характеристики пилотного проекта. По каким

вопросам оценивается пилотный процесс?
7. Какие этапы охватывет план перехода к практическому использованию ТС ПО?
8. Какие стандарты рекомендуется соблюдать при коллективной разработке ПО?

Технология Rational Unified Process (IBM Rational Software)

На сегодняшний день практически все ведущие компании – разработчики технологий и программных продуктов (IBM, Oracle, Borland, Computer Associates и др.) располагают развитыми технологиями создания ПО, которые создавались как собственными силами, так и за счет приобретения продуктов и технологий, созданных небольшими специализированными компаниями.

Одна из наиболее совершенных технологий, претендующих на роль мирового корпоративного стандарта – Rational Unified Process (RUP). RUP представляет собой программный продукт, разработанный компанией Rational Software (www.rational.com), которая в настоящее время входит в состав IBM.

RUP в значительной степени соответствует стандартам и нормативным документам, связанным с процессами ЖЦ ПО и оценкой технологической зрелости организаций-разработчиков (ISO 12207, ISO 9000, CMM и др.). Ее основными принципами являются:

1. Итерационный и инкрементный (наращиваемый) подход к созданию ПО.
2. Планирование и управление проектом на основе функциональных требований к системе – вариантов использования.
3. Построение системы на базе архитектуры ПО.

Первый принцип является определяющим. В соответствии с ним разработка системы выполняется в виде нескольких краткосрочных мини-проектов фиксированной длительности (от 2 до 6 недель), называемых итерациями. Каждая итерация включает свои собственные этапы анализа требований, проектирования, реализации, тестирования, интеграции и завершается созданием работающей системы.

Итерационный цикл основывается на постоянном расширении и дополнении системы в процессе нескольких итераций с периодической обратной связью и адаптацией добавляемых модулей к существующему ядру системы. Система постоянно разрастается шаг за шагом, поэтому такой подход называют итерационным и инкрементным.

На рис. 20. показано общее представление RUP в двух измерениях. Горизонтальное измерение представляет время, отражает динамические аспекты процессов и оперирует такими понятиями, как стадии,

итерации и контрольные точки. Вертикальное измерение отражает статические аспекты процессов и оперирует такими понятиями, как виды деятельности (технологические операции), рабочие продукты, исполнители и дисциплины (технологические процессы).

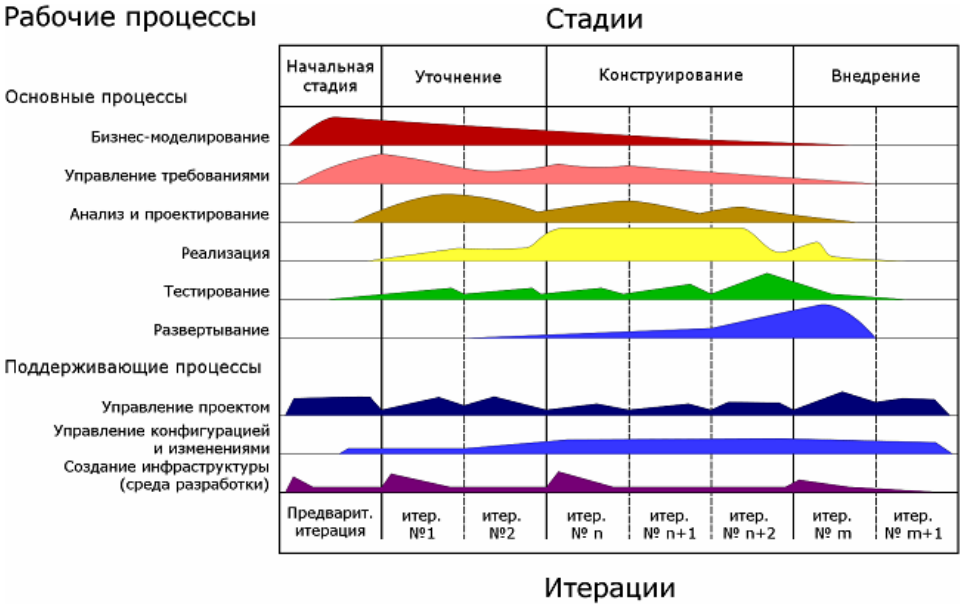


Рис. 20. Общее представление RUP

Согласно RUP, ЖЦ ПО разбивается на отдельные циклы, в каждом из которых создается новое поколение продукта. Каждый цикл, в свою очередь, разбивается на четыре последовательные стадии:

- начальная стадия (inception);
- стадия разработки (elaboration);
- стадия конструирования (construction);
- стадия ввода в действие (transition).

Каждая стадия завершается в четко определенной контрольной точке (milestone). В этот момент времени должны достигаться важные результаты и приниматься критически важные решения о дальнейшей разработке.

Начальная стадия может принимать множество разных форм. Для крупных проектов начальная стадия может вылиться во всестороннее изучение всех возможностей реализации проекта, которое займет месяцы. Во время начальной стадии вырабатывается бизнес-план проекта – определяется, сколько приблизительно он будет стоить и какой доход принесет. Определяются также границы проекта, и выполняется некоторый начальный анализ для оценки размеров проекта.

Результатами начальной стадии являются:

- общее описание системы: основные требования к проекту, его характеристики и ограничения;
- начальная модель вариантов использования (степень готовности – 10–20 %);
- начальный проектный глоссарий (словарь терминов);
- начальный бизнес-план;
- план проекта, отражающий стадии и итерации;
- один или несколько прототипов.

На стадии разработки выявляются более детальные требования к системе, выполняется высокоуровневый анализ предметной области и проектирование для построения базовой архитектуры системы, создается план конструирования и устраняются наиболее рискованные элементы проекта.

Результатами стадии разработки являются:

- модель вариантов использования (завершенная по крайней мере на 80 %), определяющая функциональные требования к системе;
- перечень дополнительных требований, включая требования нефункционального характера и требования, не связанные с конкретными вариантами использования;
- описание базовой архитектуры будущей системы;
- работающий прототип;
- уточненный бизнес-план;
- план разработки всего проекта, отражающий итерации и критерии оценки для каждой итерации.

Самым важным результатом стадии разработки является описание базовой архитектуры будущей системы. Эта архитектура включает:

- модель предметной области, которая отражает понимание бизнеса и служит отправным пунктом для формирования основных классов предметной области;
- технологическую платформу, определяющую основные элементы технологии реализации системы и их взаимодействие.

Эта архитектура является основой всей дальнейшей разработки, она служит своего рода проектом для последующих стадий. В дальнейшем неизбежны незначительные изменения в деталях архитектуры, однако, серьезные изменения маловероятны.

Стадия разработки занимает около пятой части общей продолжительности проекта. Основными признаками завершения стадии разработки являются два события:

- разработчики в состоянии оценить с достаточно высокой точностью, сколько времени потребуется на реализацию каждого варианта использования;
- идентифицированы все наиболее серьезные риски, и степень понимания наиболее важных из них такова, что известно, как справиться с ними.

Сущность планирования заключается в определении последовательности итераций конструирования и вариантов использования, реализуемых на каждой итерации. Итерации на стадии конструирования являются одновременно инкрементными и повторяющимися:

- итерации являются инкрементными в соответствии с той функцией, которую они выполняют. Каждая итерация добавляет очередные конструкции к вариантам использования, реализованным во время предыдущих итераций;
- итерации являются повторяющимися по отношению к разрабатываемому коду. На каждой итерации некоторая часть существующего кода переписывается с целью сделать его более гибким.

Результатом стадии конструирования является продукт, готовый к передаче конечным пользователям. Как минимум, он содержит следующее:

- ПО, интегрированное на требуемых платформах;
- руководства пользователя;
- описание текущей реализации.

Назначением стадии ввода в действие является передача готового продукта в распоряжение пользователей. Данная стадия включает:

- бета-тестирование, позволяющее убедиться, что новая система соответствует ожиданиям пользователей;
- параллельное функционирование с существующей (legacy) системой, которая подлежит постепенной замене;
- конвертирование баз данных;
- оптимизацию производительности;
- обучение пользователей и специалистов службы сопровождения.

Статический аспект RUP представлен четырьмя основными элементами:

- роли;
- виды деятельности;
- рабочие продукты;
- дисциплины.

Понятие "роль" (role) определяет поведение и ответственность личности или группы личностей, составляющих проектную команду. Одна личность может играть в проекте много различных ролей.

Под видом деятельности конкретного исполнителя понимается единица выполняемой им работы. Вид деятельности (activity) соответствует понятию технологической операции. Он имеет четко определенную цель, обычно выражаемую в терминах получения или модификации некоторых рабочих продуктов (artifacts), таких, как модель, элемент модели, документ, исходный код или план. Каждый вид деятельности связано с конкретной ролью. Продолжительность вида деятельности составляет от нескольких часов до нескольких дней, он обычно выполняется одним исполнителем и порождает только один или весьма небольшое количество рабочих продуктов. Любой вид деятельности должен являться элементом процесса планирования. Примерами видов деятельности могут быть планирование итерации, определение вариантов использования и действующих лиц, выполнение теста на производительность. Каждый вид деятельности сопровождается набором руководств (guidelines), представляющих собой методики выполнения технологических операций.

Дисциплина (discipline) соответствует понятию технологического процесса и представляет собой последовательность действий, приводящую к получению значимого результата.

В рамках RUP определены шесть основных дисциплин:

- построение бизнес-моделей;
- определение требований;
- анализ и проектирование;
- реализация;
- тестирование;
- развертывание;

и три вспомогательных:

- управление конфигурацией и изменениями;
- управление проектом;
- создание инфраструктуры.

RUP как продукт входит в состав комплекса Rational Suite, причем каждая из перечисленных выше дисциплин поддерживается определенным инструментальным средством комплекса. Физическая реализация RUP представляет собой Web-сайт, включающий следующие компоненты:

- описание всех элементов динамического и статического аспекта RUP;

- навигатор по всем элементам RUP, глоссарий и средство быстрого обучения технологии;
- руководства для всех участников проектной команды, охватывающие весь жизненный цикл ПО. Руководства представлены в двух видах: для осмысления процесса на верхнем уровне, и в виде подробных наставлений по повседневной деятельности;
- наставления по использованию инструментальных средств, входящих в состав Rational Suite;
- примеры и шаблоны проектных решений для Rational Rose;
- шаблоны проектной документации для SoDa;
- шаблоны в формате Microsoft Word, предназначенные для поддержки документации по всем процессам и действиям жизненного цикла ПО;
- планы в формате Microsoft Project, отражающие итерационный характер разработки ПО.

Адаптация RUP к потребностям конкретной организации или проекта обеспечивается с помощью Rational Process Workbench (RPW) – специального набора инструментов и шаблонов для настройки и публикации Web-сайтов на основе RUP. RPW поддерживает три основные функции моделирования технологических процессов:

- определение процесса;
- описание процесса;
- представление процесса.

Библиотека элементов процесса содержит текстовую информацию о каждом элементе в модели процесса, все текстовые страницы RUP, а RPW – необходимые шаблоны для создания новых страниц описания. RPW генерирует описание процессов, включающее текст и графику, в виде Web-сайта, соединяя модели процессов и библиотеку описаний в единое целое.

RUP опирается на интегрированный комплекс инструментальных средств Rational Suite. Он существует в следующих вариантах:

- Rational Suite AnalystStudio – предназначен для определения и управления полным набором требований к разрабатываемой системе;
- Rational Suite DevelopmentStudio – предназначен для проектирования и реализации ПО;
- Rational Suite TestStudio – представляет собой набор продуктов, предназначенных для автоматического тестирования приложений;
- Rational Suite Enterprise – обеспечивает поддержку полного жизненного цикла ПО и предназначен как для менеджеров проекта,

так и отдельных разработчиков, выполняющих несколько функциональных ролей в команде разработчиков.

В состав Rational Suite, кроме самой технологии RUP как продукта, входят следующие компоненты:

- Rational Rose – средство визуального моделирования (анализа и проектирования), использующее язык UML;
- Rational XDE – средство анализа и проектирования, интегрируемое с платформами MS Visual Studio .NET и IBM WebSphere Studio Application Developer;
- Rational Requisite Pro – средство управления требованиями, предназначенное для организации совместной работы группы разработчиков. Оно позволяет команде разработчиков создавать, структурировать, устанавливать приоритеты, отслеживать, контролировать изменения требований, возникающих на любом этапе разработки компонентов приложения;
- Rational Rapid Developer – средство быстрой разработки приложений на платформе Java 2 Enterprise Edition;
- Rational ClearCase – средство управления конфигурацией ПО;
- Rational SoDA – средство автоматической генерации проектной документации;
- Rational ClearQuest – средство для управления изменениями и отслеживания дефектов в проекте на основе средств e-mail и Web;
- Rational Quantify – средство количественного определения узких мест, влияющих на общую эффективность работы программы;
- Rational Purify – средство для локализации трудно обнаруживаемых ошибок времени выполнения программы;
- Rational PureCoverage – средство идентификации участков кода, пропущенных при тестировании;
- Rational TestManager – средство планирования функционального и нагрузочного тестирования;
- Rational Robot – средство записи и воспроизведения тестовых сценариев;
- Rational TestFactory – средство тестирования надежности;
- Rational Quality Architect – средство генерации кода для тестирования.

Одно из основных инструментальных средств комплекса Rational Rose представляет собой семейство объектно-ориентированных CASE-средств и предназначено для автоматизации процессов анализа и проектирования ПО, а также для генерации кодов на различных языках и выпуска проектной документации. Rational Rose реализует процесс объектно-ориентированного анализа и проектирования ПО, описанный в

RUP. В основе работы Rational Rose лежит построение диаграмм и спецификаций UML, определяющих архитектуру системы, ее статические и динамические аспекты. В составе Rational Rose можно выделить шесть основных структурных компонентов: репозиторий, графический интерфейс пользователя, средства просмотра проекта (браузер), средства контроля проекта, средства сбора статистики и генератор документов. К ним добавляются генераторы кодов для каждого поддерживаемого языка, состав которых меняется от версии к версии.

Репозиторий представляет собой базу данных проекта. Браузер обеспечивает "навигацию" по проекту, в том числе перемещение по иерархиям классов и подсистем, переключение от одного вида диаграмм к другому и т. д. Средства контроля и сбора статистики дают возможность находить и устранять ошибки по мере развития проекта, а не после завершения его описания. Генератор отчетов формирует тексты выходных документов на основе содержащейся в репозитории информации.

Средства автоматической генерации кода, используя информацию, содержащуюся в диаграммах классов и компонентов, формируют файлы описаний классов. Создаваемый таким образом скелет программы может быть уточнен путем прямого программирования на соответствующем языке (основные языки, поддерживаемые Rational Rose – C++ и Java).

В результате разработки проекта с помощью Rational Rose формируются следующие документы:

- диаграммы UML, в совокупности представляющие собой модель разрабатываемой программной системы;
- спецификации классов, объектов, атрибутов и операций;
- заготовки текстов программ.

Тексты программ являются заготовками для последующей работы программистов. Состав информации, включаемой в программные файлы, определяется либо по умолчанию, либо по усмотрению пользователя. В дальнейшем эти исходные тексты развиваются программистами в полноценные программы.

Инструментальное средство Rational XDE представляет собой развитие возможностей Rational Rose в части синхронизации модели и кода (исключающей необходимость прямой и обратной генерации кода). Rational XDE обеспечивает:

- синхронизацию между кодом и моделью;
- отображение элементов кода Java и C# в UML;
- автоматическую синхронизацию с настраиваемым разрешением конфликтов;

- одновременное отображение кода и модели;
- постоянную синхронизацию модели UML, как части проекта Java или C#.

Вопросы для самопроверки

1. Дайте характеристику каждому из 5 уровней CMM.
2. Перечислите основные принципы методологии RUP.
3. Перечислите и охарактеризуйте стадии методологии RUP.
4. Какие результаты формируются на начальной стадии?
5. Какие результаты формируются на стадии разработки?
6. Что включает описание базовой архитектуры будущей системы?
7. Перечислите основные признаки завершения стадии разработки методологии RUP.
8. Прокомментируйте утверждение « Итерации на стадии конструирования являются одновременно инкрементными и повторяющимися».
9. Перечислите содержание результатов стадии конструирования методологии RUP.
10. Перечислите и охарактеризуйте четыре основные элемента статического аспекта RUP.
11. Перечислите компоненты физической реализации RUP в виде Web-сайта.

Базовые технологии Java для создания систем телеобработки

Разработка информационной телекоммуникационной системы основывается на большом количестве современных технологий, обеспечивающих работу взаимодействующих приложений, каждая из которых заслуживает не только упоминания, но и серьезного изучения для создания эффективных систем телеобработки данных. К числу таких технологий, следует отнести технологию Java Enterprise Edition. Последующий материал посвящен рассмотрению этой технологии создания распределенных корпоративных информационных систем.

Введение в JDBC

Приложение при работе с данными, расположенными в СУБД, поддерживающими SQL, должно обеспечивать выполнения следующих задач:

- Хранение и обновление базы данных.
- Поиск данных, хранимых в базе данных и представление их в виде, удобном для пользователя.

Sun Microsystems включила в состав Java Development Kit (JDK) Java Database Connectivity (JDBC) API для обеспечения возможности создания приложений, взаимодействующих с базами данных.

Архитектура JDBC

Java – приложения непосредственно не могут связываться с базой данных для доступа к данным и получения результатов запросов, поскольку база данных может интерпретировать только SQL – операторы, а не операторы языка Java. По этой причине, необходим механизм перевода Java-утверждений в SQL-операторы. Архитектура JDBC обеспечивает механизм перевода такого рода.

Архитектура JDBC может быть представлена двумя уровнями:

- Уровень JDBC-приложения: представляет собой Java-приложение, которое использует JDBC API для взаимодействия с драйверами JDBC. Драйвер JDBC представляет собой программу, которую Java-приложение использует для доступа к базам данных. Менеджер JDBC драйверов JDBC API связывает Java-приложение с драйвером.
- Уровень драйвера JDBC: действует как интерфейс между Java-приложением и базой данных. Этот уровень содержит, например, драйверы SQL-сервера или Oracle, которые обеспечивает взаимодействие с базой данных. Драйвер посылает запрос Java-приложения к базе данных и после обработки запроса, база данных посылает ответ обратно драйверу. Драйвер преобразует его и посылает к JDBC API и тот, в свою очередь, пересылает ответ в Java-приложение. Рис. 21 представляет архитектуру JDBC.

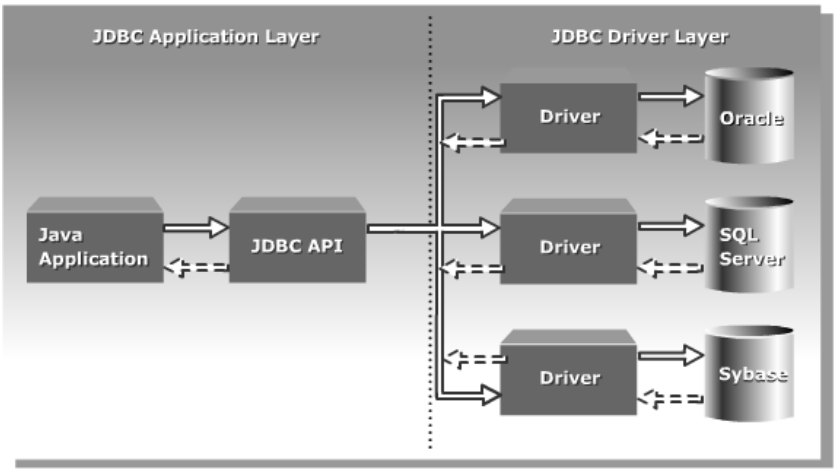


Рис. 21. Архитектура JDBC

Драйверы JDBC

При разработке JDBC-приложения должны использоваться драйверы JDBC, чтобы конвертировать запросы в форму, которую конкретная база данных может интерпретировать. Драйвер JDBC также извлекает результаты SQL-утверждений и конвертирует результат в объекты класса эквивалентные JDBC API, которые использует Java-приложение. JDBC поддерживает четыре типа драйверов:

- Мост JDBC-ODBC драйвер
- Native-API Частично-Java драйвер
- JDBC-Net Pure-Java драйвер
- Native- Protocol Pure-Java драйвер

Мост JDBC-ODBC драйвер

Мост JDBC-ODBC драйвер называется драйвером первого типа. Мост JDBC-ODBC драйвер конвертирует JDBC-запросы в запросы открытого интерфейса доступа к базам данных (ODBC). Мост JDBC-ODBC драйвер дает возможность Java-приложению использовать любую базу данных, которую поддерживает ODBC драйвер, поскольку Java-приложение не может взаимодействовать непосредственно с ODBC драйвером. По этой причине, приложение использует мост JDBC-ODBC драйвер, который работает как интерфейс между приложением и драйвером ODBC. Чтобы использовать Мост JDBC-ODBC драйвер необходимо установить драйвер ODBC на клиентском компьютере. Рис. 22 представляет работу Мост JDBC-ODBC драйвера:

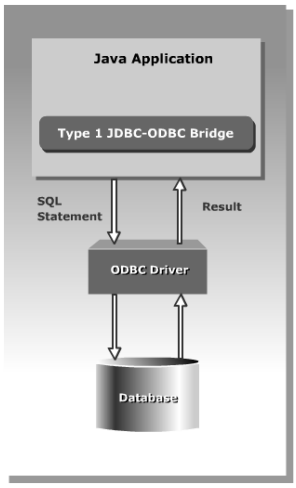


Рис. 22. Мост JDBC-ODBC драйвер

Native-API частично-Java Driver

Native -API Частично-Java драйвер называется драйвером типа 2. Он использует собственные локальные библиотеки для доступа к базам данных, поставляемые производителями баз данных. Драйвер JDBC преобразует запросы JDBC в собственные методы запросов, которые поступают на собственный локальный интерфейс уровня запроса Call Level Interface (CLI). Этот интерфейс включает функции доступа к базам данных, написанные на C. Чтобы использовать драйвер типа 2, CLI должен быть загружен на клиентском компьютере. В противоположность Мост JDBC-ODBC драйвера, Native-API частично-Java драйвер не имеет ODBC промежуточного уровня. В результате, этот драйвер имеет лучшие рабочие характеристики, чем Мост JDBC-ODBC драйвер. Этот драйвер обычно используется для сетевых приложений. Рис. 23 представляет работу Native -API частично-Java драйвера:

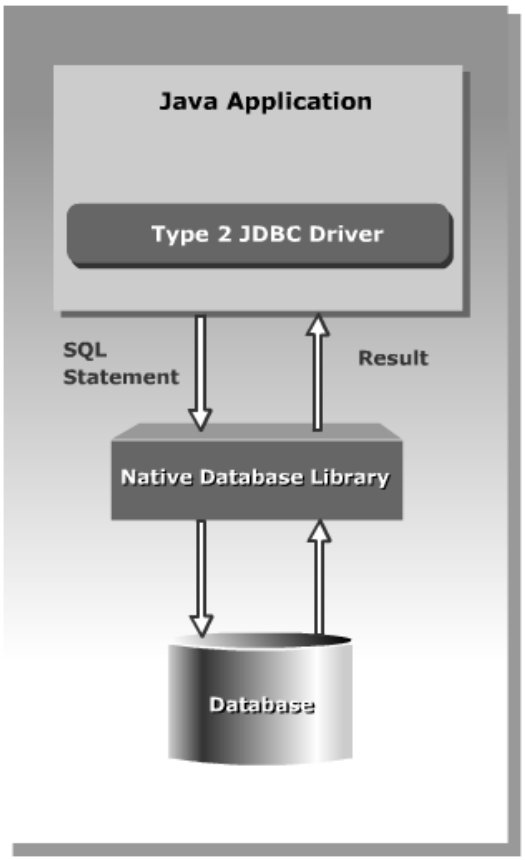


Рис. 23. Native -API частично-Java драйвер

JDBC-Net Pure-Java драйвер

JDBC-Net Pure-Java драйвер является драйвером типа 3. Он может использоваться в Веб-приложениях при взаимодействии апплетов с ба-

зами данных. Этот драйвер состоит из клиентской и серверной частей. Клиентская часть содержит только функции, а серверная часть содержит методы Java и свои собственные. Java-приложение посылает JDBC-запросы к клиентской части JDBC-Net Pure-Java драйверу, который, в свою очередь переводит JDBC-запросы в запросы базы данных. Запросы базы данных передаются серверной части JDBC-Net Pure-Java драйвера, который пересылает запрос базе данных. Когда используется JDBC-Net Pure-Java драйвер, собственные CLI библиотеки загружаются на сервер. Рис. 24 представляет работу JDBC-Net Pure-Java драйвера.

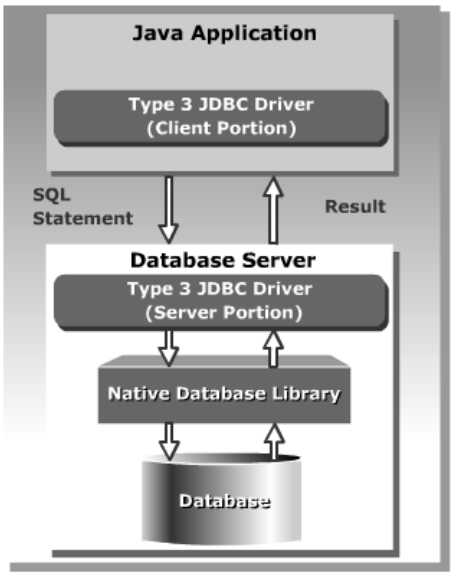


Рис. 24. JDBC-Net Pure-Java драйвер

Native-Protocol Pure-Java драйвер

Native-Protocol Pure-Java драйвер называется драйвером типа 4. Это драйвер Java непосредственно использующий сетевые протоколы, специфицированные производителем. В противоположность другим JDBC драйверам, не требуется установки, какой бы то не было библиотеки от производителя, чтобы использовать драйвер типа 4. Драйвер типа 4 поддерживают технологии прямого доступа для различных баз данных, таких как MS SQL, AS/400 и DB2. Этот драйвер обычно используется для корпоративных приложений. Рис. 25 представляет работу Native Protocol Pure-Java драйвера.

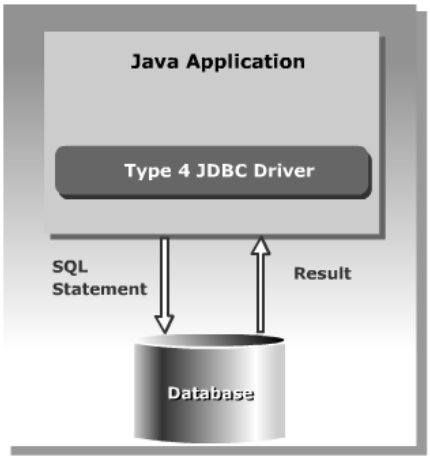


Рис.25. Native-Protocol Pure-Java драйвер

Использование JDBC API

Драйверы баз данных и JDBC API используются при разработке Java-приложения для извлечения или сохранения данных в базе данных. Классы JDBC API и интерфейсы доступны в пакетах java.sql и javax.sql. Классы и интерфейсы выполняют множество задач, таких как установление и закрытие связи с базой данных, передача запроса к базе данных, извлечение и обновление данных базы данных. Обычно используются следующие классы и интерфейсы в JDBC API:

- Класс DriverManager: загружает драйвер для базы данных.
- Интерфейс Driver: представляет драйвер базы данных. Все типы JDBC драйверов должны реализовать интерфейс Driver.
- Интерфейс Connection: позволяет установить связь между Java-приложением и базой данных.
- Интерфейс Statement: позволяет выполнить SQL-операторы.
- Интерфейс ResultSet: представляет информацию, извлеченную из базы данных.
- Класс SQLException: представляет информацию об исключительных ситуациях, которые происходят при взаимодействии с базами данных.

Чтобы запросить информацию из базы данных и отобразить результат, используя Java-приложения, необходимо:

- Загрузить драйвер
- Соединиться с базой данных
- Создать и выполнить JDBC-операторы
- Предусмотреть обработку исключительных ситуаций

Загрузка драйвера

Первым шагом разработки JDBC-приложения является загрузка и регистрация требуемого драйвера с использованием менеджера драйверов. Драйвер загружается и регистрируется двумя способами:

- программно:
  - используя метод forName()
  - используя метод registerDriver()
- вручную:
  - устанавливая системные свойства

Использование метода forName()

Метод forName() находится в классе java.lang.Class. Метод forName() загружает JDBC-драйвер и регистрирует драйвер, посредством менеджера драйверов. Синтаксис загрузки JDBC-драйвера следующий:

```
Class.forName("<driver_name>");
```

Можно загрузить Мост JDBC-ODBC драйвер, используя вызов следующего метода:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Использование метода registerDriver()

Для загрузки JDBC-драйвер может быть создан экземпляр класса Driver. Синтаксис объявления экземпляра класса Driver следующий:

```
Driver d=new <driver name>;
```

Для создания экземпляра класса Driver используется утверждение:

```
Driver d=new sun.jdbc.odbc.JdbcOdbcDriver();
```

После создания объекта класса Driver, вызывается метод registerDriver(), чтобы зарегистрировать объект с помощью менеджера драйверов. Для регистрации драйвер Мост JDBC-ODBC, используется вызов метода `registerDriver()`:

```
DriverManager.registerDriver(d);
```

Установка системных свойств

Драйвер может быть также загружен путем установки системных свойств для JDBC-драйверов. Имя драйвера добавляется к системным свойствам jdbc.drivers, чтобы загрузить JDBC-драйвер. Вы используете командную опцию –D для установки системного свойства в командной строке командной строке:

```
java –D jdbc.drivers=sun.jdbc.odbc.JdbcOdbcDriver SampleApplication
```

В этой команде jdbc.drivers – имя свойства, а sun.jdbc.odbc.JdbcOdbcDriver – значение, которое устанавливается для свойства.

После того, как Вы загрузили драйвер, вы должны установить связь с базой данных.

Связь с базой данных

Для установления связи Java-приложения с базой данных создается объект интерфейса Connection. В Java-приложении может быть создано несколько объектов Connection для доступа и извлечения данных из нескольких баз данных. Для создания объекта Connection класс DriverManager предоставляет метод getConnection(). Метод getConnection() – перезагружаемый метод, который имеет следующие три формы:

- `Connection getConnection (String <url>)`: принимает JDBC URL базы данных, к которой Вам нужно получить доступ как параметр. Может использоваться следующий фрагмент кода, для связи с базой данных, используя метод `getConnection()`:

```
String url = "jdbc:odbc:MyDataSource";
Connection con = DriverManager.getConnection(url);
```

Синтаксис параметра URL для JDBC для метода getConnection():

```
<protocol>:<subprotocol>:<subname>
```

и имеет следующие три компонента:

- **`protocol`**: указывает имя протокола, который используется для доступа к базе данных. Для JDBC, имя протокола доступа всегда jdbc.
- **`subprotocol`**: указывает механизм извлечения данных из базы данных. Например, если Вы используете Мост JDBC-ODBC драйвер, то имя подпротокола – odbc.
- **`subname`**: указывает имя источника данных (Data Source Name) (DSN), которое содержит информацию о базе данных, такую как имя базы данных, местоположение сервера базы данных, имя пользователя и пароль доступа к серверу баз данных.

- `Connection getConnection (String <url>, String <username>, String <password>)`: принимает JDBC url базы данных, имя пользователя и пароль авторизации пользователя базы данных. Можно использовать следующий фрагмент кода для соединения с базой данных:

```
String url = "jdbc:odbc:MyDataSource";
```

```
Connection con = DriverManager.getConnection
(url,"NewUser","NewPassword");
```

- **Connection getConnection (String <url>,Properties <properties>)**: принимает в качестве параметра JDBC url базы данных и объект java.util.Properties. Можно задать имя пользователя и пароль в свойствах объекта, используя метод setProperty(). Можно использовать следующий фрагмент кода для установления связи с базой данных:

```
String url = "jdbc:odbc:MyDataSource";
Properties p = new Properties();
p.setProperty("user","NewUser");
p.setProperty("password","NewPassword");
Connection con = DriverManager.getConnection(url,p);
```

В этом фрагменте **p** – ссылка на объект класса **Properties(Property)**. Имя пользователя и свойства пароля устанавливаются методом **setProperty()**.

После создания связи нужно написать операторы JDBC, которые следует выполнить.

Создание и выполнение утверждений JDBC

Необходимо создать объект **Statement** , чтобы послать запросы и извлечь результаты из базы данных. Объект **Connection** содержит метод **createStatement()**, чтобы создать объект **Statement**. Для создания объекта **Statement** может использоваться следующий фрагмент кода:

```
Connection con = DriverManager.getConnection
("jdbc:odbc:MyDataSource","NewUser","NewPassword");
Statement stmt = con.createStatement();
```

Для выполнения запроса к базе данных можно использовать статические SQL –операторы, которые не содержат параметров времени выполнения, а также можно выполнять SQL –операторы, используя объект **Statement**, который содержит следующие методы:

- **ResultSet executeQuery(String str)**: выполняет SQL-оператор и возвращает объект типа ResultSet. Этот объект предоставляет методы доступа к базе данных из объекта resultset. Метод executeQuery() используется, когда Вам нужно извлечь данные из таблицы базы данных, используя, оператор SELECT. Синтаксис использования метода **executeQuery()** следующий:

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(<SQL statement>);
```

где **stmt** это ссылка на объект интерфейса **Statement** . Метод **executeQuery()** выполняет SQL –оператор и результат, полученный из базы данных, сохраняется в поле **rs** объекта **ResultSet**.

- **`int executeUpdate(String str)`**: выполняет SQL-оператор и возвращает номер строки данных, который был получен после обработки SQL-оператора. Когда Вам нужно модифицировать данные в таблице базы данных, используя операторы языка манипулирования данными (Data Manipulation Language) (DML), INSERT, DELETE и UPDATE, можно использовать метод **`executeUpdate()`**. Синтаксис использования метода **`executeUpdate()`** следующий:

```
Statement stmt = con.createStatement();
int count = stmt.executeUpdate(<SQL statement>);
```

Здесь метод **`executeUpdate()`** выполняет SQL-оператор и номер назначенной строки базы данных сохраняется в целочисленную переменную **`count`**.

- **`boolean execute(String str)`**: выполняет SQL-оператор и возвращает булевское значение. Этот метод используется, когда тип SQL-оператора – как параметра, не известен или когда выполненный оператор возвращает результирующий набор данных или обновленный счетчик. Метод **`execute()`** возвращает значение истина, если результат SQL-оператора – объект типа **`ResultSet`** и ложь, если это обновленный счетчик. Синтаксис метода **`execute()`**:

```
Statement stmt = con.createStatement();
stmt.execute(<SQL statement>);
```

Можно использовать операторы INSERT, UPDATE и DELETE в Java-приложениях, чтобы модифицировать, хранимые в таблицах базы данных. Также используются операторы языка определения данных (Data Definition Language) (DDL) CREATE, ALTER и DROP в Java-приложениях, чтобы определять или изменять структуру объектов базы данных. Следующие различные операции выполняются в Java-приложении:

- Табличные запросы
- Вставка строк в таблицу
- Обновление строк в таблице
- Удаление строк из таблицы
- Создание таблицы
- Изменение и удаление в таблице

Табличные запросы

Для получения данных из таблицы применяется оператор SELECT, используя метод executeQuery(), который возвращает выход-

ные данные в форме объекта ResultSet. Можно использовать следующий фрагмент кода для получения данных из таблицы authors:

```
String str = "SELECT * FROM authors";
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(str);
```

В этом фрагменте кода **str** содержит оператор SELECT, который извлекает данные из таблицы **authors**. Результат сохраняется в **rs** объекта **ResultSet**.

Когда Вам нужно извлечь записи из таблицы, условия поиска определяются в утверждении WHERE оператора SELECT. Следующий фрагмент кода может использоваться для извлечения записей из таблицы authors:

```
String str = "SELECT * FROM authors WHERE city='Tomsk'";
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(str);
```

В этом фрагменте метод **executeQuery()** извлекает записи из таблицы **authors**, определенного города.

Вставка записей в таблицу

Вы можете добавить строки в существующую таблицу, используя, оператор INSERT. Метод **executeUpdate()** позволяет добавлять строки в таблицу. Следующий фрагмент кода может использоваться для вставки записи в таблицу authors:

```
      String str = " INSERT INTO authors (au_id, au_lname,
au_fname, address, city, state, contract) VALUES ('998-72-3568',
'Ringer','Albert','801 826-0752 67 Seventh Av.', 'Salt Lake
City','UT','1')";
      Statement stmt = con.createStatement();
      int count = stmt.executeUpdate(str);
```

В этом фрагменте кода, **str** содержит оператор INSERT , который необходимо выполнить в базе данных. Объект **stmt** интерфейса **Statement**, выполняет оператор INSERT, используя метод **executeUpdate()** и возвращает номер вставленной в таблицу записи в переменной **count**.

Обновление записей в таблице

Вы можете модифицировать существующую информацию в таблице, используя, оператор UPDATE, используя следующий фрагмент кода:

```
      String str = "UPDATE authors SET address='10932 Second Av.'
WHERE au_id='998-72-3568'";
      Statement stmt = con.createStatement();
      int count = stmt.executeUpdate(str);
```

В этом фрагменте, **str** содержит оператор **UPDATE**, который Вам нужно выполнить в базе данных. Объект **Statement** выполняет этот опе-

ратор, используя метод **executeUpdate()** и возвращает в переменной **count** номер строки в модифицированной в таблице.

Удаление строки из таблицы

Вы можете удалить существующую информацию из таблицы, используя, оператор DELETE. Вы можете использовать следующий фрагмент кода, чтобы удалить строку из таблицы authors:

```
String str = "DELETE FROM authors WHERE au_id='998-72-
3568'";
Statement stmt = con.createStatement();
int count = stmt.executeUpdate(str);
```

В этом фрагменте кода, **str** содержит оператор DELETE, который Вам нужно послать к базе данных. Объект **Statement** выполняет этот оператор, используя метод **executeUpdate()** и возвращает в переменной **count** номер строки, удаленной из таблицы.

Создание таблицы

Оператор CREATE TABLE используется для создания и определения структуры таблицы базы данных, используя следующий фрагмент кода:

```
String str="CREATE TABLE MyProduct"
+" (p_id INTEGER,"
+"p_name VARCHAR(25),"
+"rate FLOAT,"
+"unit_msr CHAR(6))";
Statement stmt=con.createStatement();
stmt.execute(str);
```

В этом фрагменте кода, строка **str** содержит оператор CREATE TABLE, чтобы создать таблицу **MyProduct**. Метод **execute()** используется в процессе выполнения оператора CREATE TABLE.

Изменение и удаление в таблице

DDL предоставляет оператор ALTER для изменения структуры таблицы. Например, этот оператор используется, чтобы добавить новый столбец в таблицу, изменить тип данных и ширину существующего столбца, а также добавить ограничение в столбец. Следующий фрагмент кода использует оператор ALTER TABLE для добавления в таблицу столбца MyProduct:

```
String str="ALTER TABLE MyProduct "
+"ADD quantity INTEGER";
Statement stmt=con.createStatement();
stmt.execute(str);
```

DDL предоставляет оператор DROP, чтобы удалить объект из базы данных. Вы используете оператор DROP TABLE для удаления таблицы из базы данных. Следующий фрагмент кода используется для удаления таблицы MyProduct в Java-приложении:

```
String str="DROP TABLE MyProduct";
Statement stmt=con.createStatement();
stmt.execute(str);
```

Обработка исключительных ситуаций SQL

Когда Вы создается JDBC-приложение, необходимо обрабатывать исключительные ситуации. Выполняющиеся методы вызывают SQLException, когда происходит ошибка во времени выполнения SQL-операторов.

Пакет java.sql обеспечивает класс SQLException, который является производным от класса java.lang.Exception. SQLException вызывается различными методами JDBC API и позволяет определить причину ошибок, которые появляются впроцессе взаимодействия Java-приложения с базой данных. Можно перехватывть SQLException в Java-приложении, используя блок try-catch обработки исключительных ситуаций. Класс SQLException предоставляет следующую информацию об ошибках:

- Error message (сообщение об ошибке): это строка, в которой описана ошибка.
- Error code (Код ошибки): это целочисленное значение, которое ассоциировано с ошибкой. Код ошибки специфицирован производителем и зависит от используемой базы данных.
- SQL-state (SQL-состояние): это X/OPEN код ошибки, который идентифицирует ошибку. Различные производители поставляют различные сообщения об ошибках для определения одной и той же ошибки. В результате, ошибка может выдавать различные сообщения об ошибках. X/OPEN код ошибки это стандартное сообщение, связанное с ошибкой, которое может идентифицировать ошибку для многих баз данных.

Класс SQLException содержит различные методы, которые предоставляют информацию об ошибках, а именно:

- `int getErrorCode()`: возвращает код ошибки, ассоциированной с появившейся ошибкой.
- `String getSQLState()`: возвращает X/Open код ошибки.
- `SQLException getNextException()`: возвращает следующее исключение в цепочке исключительных ситуаций.

Следующий фрагмент кода используется для перехвата SQLException:

```
try
{
String str = "DELETE FROM authors WHERE au_id='998-72-
3568'";
Statement stmt = con.createStatement();
int count = stmt.executeUpdate(str);
}
catch(SQLException sqlExceptionObject)
{
System.out.println("Display Error Code");
System.out.println("SQL Exception"+
sqlExceptionObject.getErrorCode());
}
```

В этом фрагменте кода, если оператор DELETE во время выполнения вызывает SQLException, то эта ситуация обрабатывается, используя блок try-catch. Объект **`sqlExceptionObject`** класса **`SQLException`** используется для вызова метода **`getErrorCode()`**.

Доступ к результирующему набору данных

Когда выполняется запрос на получение данных из таблицы в Java-приложении, выходные данные запроса сохраняются в объекте ResultSet в формате таблицы. Объект ResultSet поддерживает курсор, который дает возможность перемещаться по строкам, сохраненным в объекте ResultSet. По умолчанию, объект ResultSet поддерживает курсор, который позволяет перемещаться только в прямом направлении. В итоге, он перемещается от первой строки до последней строки в ResultSet. Вы не можете изменить умолчание объекта ResultSet. Курсор в объекте ResultSet object первоначально устанавливается перед первой строкой.

Типы результирующего набора данных

Вы можете создавать множество типов объекта, в том числе:

- Read only (только для чтения): позволяет только читать строки объекта ResultSet.
- Forward only (только вперед): позволяет перемещать курсор от первой до последней строки только в прямом направлении.
- Scrollable (прокручиваемый): позволяет перемещать курсор в прямом или обратном направлении.
- Updateable (обновляемый): позволяет обновлять строки результирующего набора данных.

Вы можете задать тип объекта ResultSet, используя метод createStatement() интерфейса Connection. Метод createStatement() принимает поля ResultSet как параметры, чтобы создать различные типы объектов ResultSet. Табл. 1 представляет различные поля интерфейса ResultSet, которые можно использовать для создания различных типов результирующего набора данных:

Таблица 1

Поля результирующего набора данных	***Описание***
TYPE_SCROLL_SENTITIVE	Определяет, что курсор объекта ResultSet прокручиваемый и в нем отражаются изменения в данных, сделанные другими пользователями.
TYPE_SCROLL_INSENSITIVE	Определяет, что курсор объекта ResultSet прокручиваемый и не отражаются изменения в данных, сделанные другими пользователями.
TYPE_FORWARD_ONLY	Определяет, что курсор объекта ResultSet перемещается только в прямом направлении от первой строки до последней строки.

Табл. 2 представляет различные поля интерфейса ResultSet, которые можно использовать, чтобы определения различных параллельных режимов работы с результирующим набором данных.

Таблица 2

Поля результирующего набора данных	***Описание***
CONCUR_READ_ONLY	Определяет параллельный режим работы, который не позволяет Вам обновлять объект ResultSet.
CONCUR_UPDATABLE	Определяет параллельный режим работы, который позволяет Вам обновлять объект ResultSet.

Табл. 3 представляет различные поля интерфейса ResultSet, которые можно использовать для определения различных состояний курсора результирующего набора данных.

Таблица 3

Поля результирующего набора данных	***Описание***
HOLD_CURSORS_OVER_COMMIT	Определяет, что объект ResultSet не должен быть закрыт, после того, как данные помещены в базу данных.
CLOSE_CURSORS_AT_COMMIT	Определяет, что объект ResultSet не должен быть закрыт, после того, как данные помещены в базу данных.

Метод createStatement() это перезагружаемый метод, который имеет следующие три прототипа createStatement():

- **createStatement()**: не принимает каких-либо параметров. Этот метод создает по умолчанию объект ResultSet, который позволяет только продвижение вперед.
- **createStatement(int, int)**: принимает два параметра. Первый параметр указывает тип ResultSet, который определяет, может ли курсор результирующего набора данных перемещаться назад. Второй параметр указывает режим параллелизма для результирующего набора данных, который определяет, могут ли данные результирующего набора данных быть обновлены. Этот метод создает объект ResultSet заданного типа и параллелизма.
- **createStatement(int, int, int)**: принимает три параметра. Дополнительно к типу ResultSet и режиму параллелизма, этот метод принимает третий параметр, указывающий, может ли результирующий набор данных закрыт после помещения данных в базе данных. Этот метод создает объект ResultSet с заданным типом, параллелизмом и состоянием.

Методы интерфейса ResultSet

Интерфейс ResultSet содержит различные методы, которые позволяют Вам перемещать курсор по результирующему набору данных. Табл. 4 представляет методы интерфейса ResultSet.

Таблица 4

Метод	***Описание***
boolean first()	Перемещает курсор результирующего набора данных к первой строке результирующего набора данных.
boolean isFirst()	Определяет, указывает ли курсор результирующего набора данных на первую строку результирующего набора данных.
boolean beforeFirst()	Перемещает курсор результирующего набора данных перед первой строкой результирующего набора данных.
boolean last()	Перемещает курсор результирующего набора данных к последней строке результирующего набора данных.
boolean isLast()	Определяет, установлен ли курсор результирующего набора данных на последнюю строку результирующего набора данных.
boolean afterLast()	Перемещает курсор результирующего набора данных после последней строки результирующего набора данных.
boolean isAfterLast()	Определяет, установлен ли курсор результирующего набора данных после последней строки результирующего набора данных.
boolean previous()	Перемещает курсор результирующего набора данных на предыдущую строку результирующего набора данных.
boolean absolute(int i)	Перемещает курсор результирующего набора данных к строке, номер которой специфицирован как параметр.
boolean relative(int i)	Перемещает курсор результирующего набора данных вперед или назад, относительно строки, номер которой специфицирован как параметр. Этот метод принимает или положительные или отрицательные значения как параметр.

Можете создать результирующий набор данных со скроллингом, который обеспечивает перемещение по строкам результирующего набора данных в обоих направлениях. Следующий фрагмент кода используется для создания результирующего набора данных только для чтения со скроллингом:

```
Statement stmt =
con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, Result-
Set.CONCUR_READ_ONLY);
ResultSet rs=stmt.executeQuery ("SELECT * FROM authors");
```

Можно определять местоположение курсора результирующего набора данных, используя методы интерфейса ResultSet. Так, можно использовать следующий фрагмент кода для определения положения курсора перед первой строкой в результирующем наборе данных:

```
if(rs.isBeforeFirst()==true)
System.out.println("Result set cursor is before the first row in
the result set");
```

В представленном коде, `rs` – объект `ResultSet`, который вызывает метод `isBeforeFirst()`.

Можно перемещать отдельные строки, такие как первую или последнюю в результирующем наборе данных, используя методы интерфейса ResultSet. Вы можете использовать следующий фрагмент кода, чтобы переместить курсор результирующего набора данных к первой строке результирующего набора данных:

```
if(rs.first()==true)
System.out.println(rs.getString(1) + ", " + rs.getString(2)+ ", "
+ rs.getString(3));
```

В представленном коде, `rs` – объект the `ResultSet`, который вызывает метод `first()`.

Аналогично, можно перемещать курсор результирующего набора данных к последней строке результирующего набора данных, используя метод `last()`.

Если необходимо переместить курсор на определенную строку в результирующем наборе данных, можно использовать метод absolute(). Например, курсор результирующего набора данных позиционирован на первой строке, и нужно перейти на четвертую строку, следует определить параметр равны1 4 при вызове метода absolute(), как показано в следующем фрагменте кода:

```
System.out.println("Using absolute() method");
rs.absolute(4);
int rowcount = rs.getRow();
System.out.println("rowNum should be 4 " + rowcount);
```

JDBC позволяет создать обновляемый результирующий набор данных, позволяющий модифицировать его строки. Табл. 5 представляет ряд методов, используемых с обновляемым результирующим набором данных:

Таблица 5

Метод	*Описание*
`void updateRow()`	Обновляет строку текущего объекта ResultSet и таблицу базы данных.
`void insertRow()`	Обновляет строку текущего объекта ResultSet и таблицу базы данных.
`void updateString()`	Обновляет определенный столбец, заданный строковым значением.
`void updateInt()`	Обновляет определенный столбец, заданный целым значением.

Вы можете использовать следующий фрагмент кода, чтобы модифицировать авторскую информацию, используя обновляемый результирующий набор данных:

```
Statement stmt = con.createStatement();
stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, Re-
sultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery("SELECT au_id, city, state FROM
authors WHERE au_id='893-72-1158' ");
rs.next();
rs.updateString("state", "NY");
rs.updateString("city", "Columbia");
rs.updateRow();
```

В представленном фрагменте кода, строка, получена из таблицы **authors**, в которой поле **au_id** равно 893-72-1158. В полученной строке, значение столбца **state** изменено на **NY** и значение столбца **city** изменено на **Columbia.**

Запросы и модификация данных, использующие объект PreparedStatement

Интерфейс PreparedStatement является производным от интерфейса Statement и доступен в пакете java.sql. Объект PreparedStatement позволяет передать параметры во время выполнения операторам SQL, чтобы опрашивать и модифицировать данные в таблице.

Объекты PreparedStatement компилируются и подготовляются только однажды посредством JDBC. Будущие обращения объекта не перекомпилируют операторы SQL. Это помогает уменьшить загрузку сервера базы данных и, таким образом, повысить качество рабочих характеристик приложения.

Методы интерфейса PreparedStatement

Интерфейс PreparedStatement наследует следующие методы из интерфейса Statement для выполнения операторов SQL:

- `ResultSet executeQuery()`: выполняет оператор SELECT и возвращает результат в объект ResultSet.
- `int executeUpdate()`: выполняет оператор SQL INSERT, UPDATE или DELETE и возвращает счетчик обработанных записей.
- `boolean execute()`: выполняет оператор SQL и возвращает значение типа boolean.

Представим, что Вы должны извлечь данные об авторе путем передачи id автора во время выполнения. Оператор SQL с параметризованным запросом выглядит следующим образом:

```
SELECT * FROM authors WHERE au_id = ?
```

Чтобы передать такой параметризованный запрос базе данных из приложения, необходимо создать объект PreparedStatement, используя метод prepareStatement() объекта Connection. Можно использовать следующий вызов метода, чтобы подготовить оператор SQL для определения значения во время выполнения:

```
stat=con.prepareStatement("SELECT * FROM authors WHERE au_id =
?");
```

Метод `prepareStatement()` объекта `Connection` получает оператор SQL как параметр. Операторы SQL могут содержать символ '?' как заполнитель, который может быть замещен вводимыми параметрами во время выполнения.

Перед выполнением оператора SQL, специфицированного в объекте PreparedStatement, необходимо установить значение каждого '?' параметра. Это делается вызовом соответствующего метода setXXX(), где XXX это тип данных параметра. Например:

```
stat.setString(1,"1001");
ResultSet result=stat.executeQuery();
```

В этом фрагменте кода, первый параметр метода `setString()` задает индексное значение заполнителя '?', а второй параметр используется для установки '1001' как значение заполнителя '?'. Можно использовать следующий фрагмент кода, когда значение для заполнителя '?' получено из пользовательского интерфейса:

```
stat.setString(1,aid.getText());
ResultSet result=stat.executeQuery();
```

В этом фрагменте кода, метод `setString()` используется, чтобы установить значение заполнителя '?' значением, полученным во время выполнения из текстового поля `aid` пользовательского интерфейса.

Интерфейс PreparedStatement предоставляет различные методы, чтобы установить значение заполнителей для определенных типов данных. Табл. 6 представляет часто используемые методы установки интерфейса PreparedStatement языка Java для параметра, соответствующего `index`, передаваемому как параметр.

Таблица 6

Метод	*Описание*
`void setByte(int index, byte val)`	Устанавливает значение типа byte.
`void setBytes(int index, byte[] val)`	Устанавливает массив типа byte.
`void setBoolean(int index, boolean val)`	Устанавливает значение типа boolean.
`void setDouble(int index, double val)`	Устанавливает значение типа double.
`void setInt(int index, int val)`	Устанавливает значение типа int.
`void setLong(int index, long val)`	Устанавливает значение типа long.
`void setFloat(int index, float val)`	Устанавливает значение типа float.
`void setShort(int index, short val)`	Устанавливает значение типа short.
`void setString(int index, String val)`	Устанавливает значение типа String.

Извлечение записей

Следующий фрагмент кода используется, чтобы извлечь книги, написанные одним автором из таблицы titles, используя объект PreparedStatement:

```
String str = "SELECT * FROM titles WHERE au_id = ?"
PreparedStatement ps= con.prepareStatement(str);
ps.setString(1, "1001");
ResultSet rs=ps.executeQuery();
```

В этом фрагменте кода, переменная `str` содержит оператор SELECT, который располагает одним входным параметром. Метод `setString()` используется для установки значения атрибута `au_id` таблицы `titles`. Оператор SELECT выполняется, используя метод `executeQuery()`, который возвращает объект `ResultSet`.

Вставка строк

Можно использовать следующий фрагмент кода для создания объекта PreparedStatement, который вставляет строку в таблицу authors, определением данных authors во время выполнения:

```
String str = "INSERT INTO authors (au_id, au_fname,
au_lname) VALUES (?, ?, ?)";
PreparedStatement ps = con.prepareStatement(str);
ps.setString(1, "1001");
ps.setString(2, "Abraham");
ps.setString(3, "White");
int rt=ps.executeUpdate();
```

В этом фрагменте кода, переменная **str** сохраняет оператор INSERT, который содержит три входных параметра. Метод **setString()** используется, чтобы установить значения для **au_id**, **au_fname**, и **au_lname** столбцов таблицы **authors**. Оператор INSERT выполняется, используя метод **executeUpdate()**, который возвращает целочисленное значение, определяющее число строк вставленных в таблицу.

Обновление и удаление строк

Вы можете использовать следующий фрагмент кода, чтобы модифицировать **state** на **CA**, где **city** = **Oakland** в таблице **authors**, с использованием объекта PreparedStatement:

```
String str = "UPDATE authors SET state= ? WHERE city= ? ";
PreparedStatement ps = con.prepareStatement(str);
ps.setString(1, "CA");
ps.setString(2, "Oakland");
int rt=ps.executeUpdate();
```

В этом фрагменте кода, два входных параметра **state** и **city** содержат значения для **state** и **city** атрибутов таблицы **authors**, соответственно.

Вы можете использовать следующий фрагмент кода для удаления строки из таблицы authors, где имя автора это Abraham, используя объект PreparedStatement:

```
String str = "DELETE FROM authors WHERE au_fname= ? ";
PreparedStatement ps = con.prepareStatement(str);
ps.setString(1, "Abraham");
int rt=ps.executeUpdate();
```

Вопросы для самопроверки

1. Из каких двух уровней состоит архитектура JDBC?
2. Какое программное обеспечение Java-приложение использует для доступа к базе данных?
3. Какие 4 типа драйверов поддерживает JDBC?
4. В каких пакетах определены классы и интерфейсы JDBC API?
5. Какие имеются два способа загрузки и регистрации драйвера программно?
6. Какой объект устанавливает связь между Java-приложением и базой данных?

7. Какой объект посылает запрос и извлекает результаты из базы данных?
8. Какой объект сохраняет результат, извлеченный из базы данных, когда выполняется оператор SELECT?
9. Объект какого интерфейса позволяет Java-приложению установить связь с базой данных?
10. Какой метод используется, чтобы установить уровень локализации транзакции.
11. Какая из следующих констант устанавливает уровень локализации базы данных, чтобы предотвратить недействительные результаты чтений и разовые чтения?

    TRANSACTION_READ_UNCOMMITTED
    TRANSACTION_READ_COMMITTED
    TRANSACTION_REPEATABLE_READ
    TRANSACTION_SERIALIZABLE

12. Что будет результатом выполнения следующего фрагмента кода?

```
Statement sql2 = con.createStatement();
int result=sql2.executeUpdate("Select * From Publishers");
```

    Будет возбуждаться исключительная ситуация
    Закончится ошибкой компиляции
    Будет возвращаться число строк, извлеченных из таблицы Publishers
    Будет возвращаться null

Для освоения материала предлагается выполнить лабораторную работу № 1 в разделе Лабораторный практикум.

Введение в программирование сетевых сокетов

Основы сетевого взаимодействия

Java был разработан как язык сетевого программирования для обеспечения возможности создания клиент/серверных приложений, которые взаимодействуют друг с другом в сети. Java обеспечивает обширную библиотеку сетевых классов, которые позволяют быстро получать доступ к сетевым ресурсам. Существуют два механизма, предназначенных для общения. Это пользовательский датаграммный протокол (User Datagram Protocol) (UDP) без установления соединения и протокол управления передачей/протокол Интернет (Transmission Control Protocol/Internet Protocol) (TCP/IP), использующий соединение.

Датаграмма – независимое самоопределенное общение, посланное по сети, прибытие которого, время прибытия и содержание не гарантировано. Не гарантируется также и порядок доставки сообщений.

Напротив, общение между компьютерами по протоколу TCP/IP, обеспечивает надежный канал точка-точка путем установления соединения. Все данные, посланные по каналу, получаются в том же порядке, в котором они передавались.

Архитектура клиент/сервер

Большинство разрабатываемых информационных систем используют клиент/серверную модель. Клиент посылает запрос на сервер, который обрабатывает этот запрос клиента и предоставляет запрошенный ресурс. Общение между клиентом и сервером является важным компонентом клиент/серверной модели и происходит обычно через сеть.

Клиент/серверная модель представляет собой архитектуру разработки приложения, разработанную для отделения представления данных от их внутренней обработки и хранения. Клиент запрашивает сервисы, и сервер обслуживает эти запросы. Запрос передается от клиента на сервер через сеть, обработка выполняется на сервере и скрыта от клиента. При этом один сервер может обслуживать несколько клиентов. На рис. 26 представлена клиент/серверная модель, обслуживающая несколько клиентов.

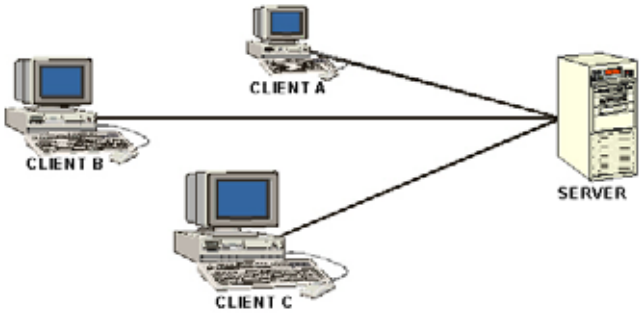


Рис. 26. Несколько клиентов, имеющих доступ к серверу

Сервер и клиент не обязательно являются компонентами оборудования. Это могут быть работающие программы на одной или различных машинах.

Серверная часть клиент/серверного приложения управляет ресурсами, распределенными среди нескольких пользователей, которые получают доступ к серверу вместе с другими клиентами. Лучшим примером демонстрации серверной части клиент/серверной программы может

быть Веб сервер, который посылает HTML страницу через Интернет различным пользователям.

Протоколы

Когда Вы общаетесь с друзьями, Вы соблюдаете некоторые не писаные правила (или протоколы). Например, Вы не разговариваете одновременно и не разговариваете непрерывно. Никто из Вас не мог бы понять, что говорит другой человек, если бы Вы не следовали правилам. Когда Вы говорите, Ваш друг слушает и наоборот. Вы говорите на языке и в темпе, который все понимают.

Когда компьютеры общаются, им также необходимо следовать определенным правилам. Данные посылаются от одной машины на другую в форме пакетов (packets). Правила управляют упаковкой данных в пакеты, скоростью передачи и разборкой данных в их исходную форму. Эти правила называются сетевыми протоколами. Сетевой протокол является набором правил и соглашений, поддерживаемых системами, которые общаются через сеть. Сетевое программное обеспечение обычно реализуется несколькими уровнями протоколов, располагающихся слоями, один над другим.

IP адрес и порт

Сервер Интернет может рассматриваться как набор классов сокетов, которые обеспечивают дополнительные возможности – обычно называемые сервисами. Примерами сервисов являются электронная почта, сетевой удаленный доступ (Telnet) и протокол передачи файлов (FTP). Каждый сервер связан с портом (port). Порт является числовым адресом, через который обрабатывается запрошенный сервис, например, запрос Веб страницы.

Протокол TCP запрашивает два элемента данных: IP адрес и номер порта. Как происходит, что, когда Вы вводите http://www.tpu.ru, Вы получаете домашнюю страницу ТПУ? Интернет протокол (Internet protocol) (IP) обеспечивает логический адрес, называемый IP адресом каждого сетевого устройства. IP адреса, используемые в Интернет, принимают определенный формат. Каждый адрес представляется 32-разрядным числом, состоящим из четырех 8-разрядных чисел, каждое в диапазоне значений от 0 до 255. ТПУ имеет свое зарегистрированное имя, позволяющее www.tpu.ru представляться IP адресом 85.43.64.1.

Если номер порта не указан, используется номер порта по умолчанию для сервиса, примеры которых приведены в табл. 7.

Таблица 7

Номер порта	*Приложение*
21	FTP, которое передает файлы
23	Telnet, которое обеспечивает удаленный вход
25	SMTP, которое доставляет почтовые сообщения
67	BOOTP, которое обеспечивает конфигурацию во время загрузки
80	HTTP, которое передает Веб страницы
109	POP, которое дает возможность пользователям получать доступ к почтовому адресу на удаленной системе

Давайте посмотрим на URL еще раз:

```
http://www.tpu.ru
```

Первый компонент URL (который является, http) означает, что Вы используете протокол передачи гипертекстовых файлов (Hypertext Transmission Protocol) (HTTP) для управления Веб документами. Если файл не задан, большинство Веб серверов конфигурируются таким образом, чтобы представлять файл с именем index.html. Следовательно, IP адрес и порт являются определенными либо явной спецификацией всех частей URL, либо использованием спецификации по умолчанию.

Сокеты

В клиент/серверном приложении сервер обеспечивает сервисы, аналогичные обработке запросов базы данных или модификации данных в базе данных. Взаимодействие, которое происходит между клиентом и сервером должно быть надежным. При этом данные не должны быть потеряны и должны быть доступны клиенту в той же последовательности, в которой сервер их посылал.

TCP обеспечивает надежный канал соединения точка-точка (point-to-point) для клиент/серверных приложений для взаимодействия между собой. Для взаимодействия по протоколу TCP, программы клиента и сервера устанавливают соединение и связывают сокет. Сокеты (англ. socket углубление, гнездо, разъём) – это название программного интерфейса для обеспечения информационного обмена между процессами. Процессы при таком обмене могут исполняться как на одной ЭВМ, так

и на различных ЭВМ, связанных между собой сетью. Сокет – абстрактный объект, представляющий конечную точку соединения и, на сленге современных системных администраторов, означает комбинацию IP-адреса и номера порта. Сокеты используются для управления линией связи между приложениями через сеть. Каждое TCP-соединение может быть однозначно идентифицировано своими двумя конечными точками. Таким способом можно иметь множественные соединения вашего хоста и сервера. Дальнейшее общение между клиентом и сервером происходит через сокет.

Преимущество модели сокетов, использующих TCP, над другими коммуникационными моделями, такими как NetBEUI и Apple Talk, состоит в том, что сервер не находится под воздействием источника клиентских запросов. Он обслуживает все запросы до тех пор, пока клиенты используют протокол TCP/IP. Это означает, что клиентом может быть компьютер любого типа и не ограничивается платформами UNIX, Windows, DOS или Macintosh. Следовательно, все компьютеры в сети, применяющие TCP/IP, могут общаться друг с другом посредством сокетов.

Java способствует упрощению сетевого программирования, обеспечивая различные классы и интерфейсы в пакете java.net.

Классы Java для сетевого программирования

Пакет java.net языка программирования Java содержит классы и интерфейсы, которые обеспечивают поддержку работы в сети. Эти классы и интерфейсы используют протоколы для сетевого программирования.

Классы пакета `java.net`

Сетевые классы содержат методы для выполнения таких задач как открытие и закрытие соединения с удаленной машиной, передача и получение пакетов данных и доступ к ресурсам Веб. Ниже перечислены некоторые из классов, представленных в пакете java.net:

- DatagramPacket: Представляет объект датаграммного пакета (datagram packet), который содержит данные вместе с информацией об адресе источника и адресе назначения, номерах порта источник и назначения. Эта информация используется, чтобы передавать датаграммный пакет от одного компьютера к другому через сеть.

- DatagramSocket: Представляет объект датаграммного сокета, который может посылать и принимать пакеты датаграмм. Пакеты посылаются объектом DatagramSocket, могут прибывать в любом порядке к получателю.
- MulticastSocket: Создает мультикаст (multicast) объект датаграммного сокета, который используется, чтобы посылать и принимать пакеты датаграмм для групп. IP адреса класса D (IP адреса между 224.0.0.0 и 239.255.255.255) используются для создания групп. Когда сообщение посылается на IP адрес класса D, все клиенты, присоединенные к группе, получают сообщение. Этот класс содержит методы joinGroup() и leaveGroup(), которые дают возможность клиентам объединяться и покидать определенную группу.
- InetAddress: Создает объект, который содержит информацию, такую как IP адрес и имя хоста (host name).
- ServerSocket: Создает объект сокета сервера, который слушает запросы клиента. Объекты ServerSocket используют номер порта, чтобы получать запросы клиента.
- Socket: Создает объект сокета клиента, который соединяется с объектом класса ServerSocket, чтобы посылать запросы на сервер.
- URL: Представляет объект, который может размещать файл или существующий ресурс в Интернет.

Пакет java.net также содержит классы обработки исключительных ситуаций, которые управляют сетевыми ошибками во время выполнения. Ниже представлены несколько классов обработки исключительных ситуаций, представленных в пакете java.net:

- BindException: Этот объект исключительных ситуаций генерируется, когда возникает ошибка попытки связаться с сокетом локального адреса или порта. Например, когда не может быть доступен локальный адрес или запрашиваемый порт используется.
- ConnectException: Этот объект исключительных ситуаций генерируется, когда возникает ошибка во время связи сокета с удаленным IP адресом и портом. Например, когда удаленное соединение не может быть установлено.
- MalformedURLException: Этот объект исключительных ситуаций генерируется, когда URL содержит недействительный протокол или если URL не может быть успешно обработана.

- UnknownHostException: Этот объект исключительных ситуаций генерируется, когда IP адрес компьютера хоста не может быть определен или IP адрес не существует.

Класс InetAddress

Класс InetAddress позволяет определять имена хостов и связанные IP адреса, требуемые во время создания сетевых приложений. Объекты InetAddress инициализируются с использованием методов static, объявленных в классе InetAddress. Некоторые из static методов, используемых для инициализации объектов InetAddress следующие:

- public static InetAddress getLocalHost(): Возвращает объект InetAddress, который содержит IP адрес локального компьютера.
- public static InetAddress getByName(String host): Возвращает объект InetAddress, который содержит IP адрес имени хоста, переданный методу как String.
- public static InetAddress[] getAllByName(String host): Возвращает массив объектов InetAddress, который содержит IP адреса для имени хоста, переданный методу как String.

Статические методы getLocalHost(), getByName(), getAllByName() и getByAddress() вызывают исключительную ситуацию UnknownHostException.

Нестатические методы, определенные в классе InetAddress могут быть доступны после инициализации объектов InetAddress. Следующие нестатические методы, определены в классе InetAddress:

- public boolean equals(Object obj): Возвращает истину, если объект InetAddress имеет такой же IP адрес как объект obj, переданный как параметр.
- public byte[] getAddress(): Возвращает IP адрес объекта InetAddress в форме массива byte. Например, если IP адрес объекта InetAddress ipaddress равен 78.140.53.12, то этот метод возвратит следующий массив byte:

```
ipaddress[0] = 78
ipaddress[1] = 140
ipaddress[2] = 53
ipaddress[3] = 12
```

- public String getHostAddress(): Возвращает IP адрес объекта InetAddress как String.
- public String toString(): Возвращает IP адрес хоста как string в формате имя хоста/IP адрес.

Следующий код используется, чтобы инициализировать и использовать объекты InetAddress:

```
import java.net.*;
class InetAddressClass
{
  public static void main(String arg[])
  {
    try
    {
/*  Создается  объект  InetAddress,  используя  getLocalHost()
статический метод класса InetAddress */
      InetAddress address = InetAddress.getLocalHost();
/* Получение IP адреса хоста */
      String addressHost = address.getHostAddress();
      System.out.println(addressHost);
    }
   catch(UnknownHostException e)
   {
      System.out.println("Error");
    }
  }
}
```

В предыдущем коде определяется класс `InetAddressClass`. В методе `main()` класса `InetAddressClass` для объекта `InetAddress` инициализируется `address`, используя статический метод `getLocalHost()`. Объект `InetAddress` запоминает IP адрес компьютера, на котором выполняется программа. Метод getHostAddress() возвращает IP адрес компьютера как `string`. Код выше сохраняется как InetAddressClass.java. Рис. 27 показывает вывод класса InetAddressClass.

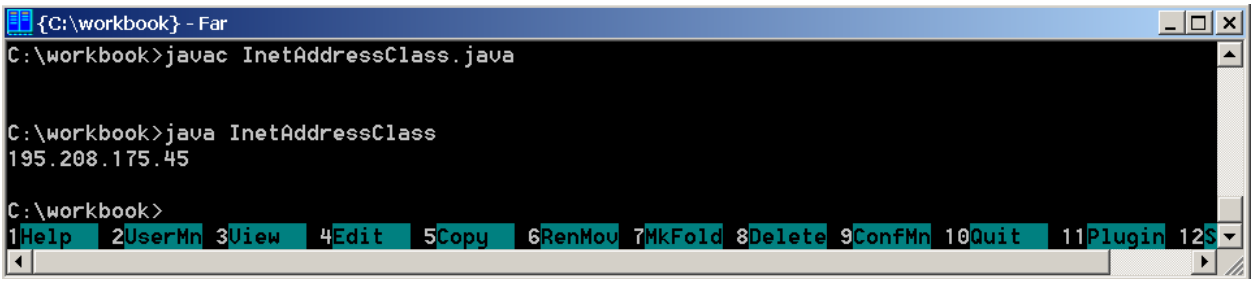


Рис. 27. Вывод класса InetAddressClass

Создание приложения с использованием UDP

Java позволяет разрабатывать сетевые приложения с использованием датаграммных сокетов UDP и сокетов TCP/IP. Сокеты UDP используют протокол UDP для общения через сеть. UDP является быстрым, без установления соединения и ненадежным протоколом. Пакет java.net предлагает следующие два класса, позволяют применять сокеты UDP в приложении Java:

- Класс DatagramPacket
- Класс DatagramSocket

Классы DatagramPacket и DatagramSocket

Объект DatagramPacket является контейнером данных, состоящим из датаграммных пакетов, которые посылаются или принимаются через сеть. Следующие конструкторы используются для инициализации объектов DatagramPacket:

- public DatagramPacket(byte[] buffer, int buffer_length): Создает объект DatagramPacket, который принимает и сохраняет данные в массиве byte. Длина буфера массива byte задается как второй параметр buffer_length.
- public DatagramPacket(byte[] buffer, int buffer_length, InetAddress address, int port): Создает объект DatagramPacket, который посылает пакеты данных заданной длины. Пакеты данных посылаются на компьютер с заданным IP адресом и номером порта, передаваемыми как параметры.

Методы, определенные в классе DatagramPacket, могут быть использованы после инициализации объекта класса DatagramPacket. Табл. 8 представляет существующие методы в классе DatagramPacket.

Таблица 8

Метод	***Описание***
`public InetAddress getAddress()`	Возвращает объект InetAddress, который содержит IP адрес компьютера, на который посылается датаграммный пакет или от которого получается датаграммный пакет
`public byte[] getData()`	Возвращает буферный массив byte, который содержит данные
`public int getLength()`	Возвращает длину буферного массива, который хранит данные
`public int getPort()`	Возвращает номер порта компьютера, на который посылается датаграммный пакет или откуда датаграммный пакет принимается.
`public void setAddress(InetAddress address)`	Устанавливает IP адрес машины, на которую датаграммный пакет должен быть послан
`public void setData(byte[] buffer)`	Устанавливает массив byte как данные для пакета.
`public void setPort(int port)`	Устанавливает номер порта на удаленном хосте
`public void setLength(int length)`	Устанавливает длину буфера

Класс DatagramSocket инкапсулирует функциональность, чтобы управлять объектами DatagramPacket. Объекты DatagramPacket посылают и принимают сохраненные данные, используя объект DatagramSocket. Следующие конструкторы, используются для инициализации объекта DatagramSocket:

- public DatagramSocket(): Создает объект a DatagramSocket и связывает его с любым доступным портом на локальном компьютере.
- public DatagramSocket(int port): Создает объект и связывает его с портом на локальном компьютере, заданным как параметр.
- public DatagramSocket(int port, InetAddress address): Создает объект и связывает его с портом заданного компьютера.

Конструктор класса DatagramSocket вызывает исключительную ситуацию SocketException.

Табл. 9 представляет некоторые из методов, содержащиеся в классе DatagramSocket, которые используются для получения информации от объектов DatagramSocket:

Таблица 9

Метод	***Описание***
`public InetAddress getInetAddress()`	Возвращает объект InetAddress, который содержит IP адрес, с которым объект DatagramSocket соединяется.
`public InetAddress getLocalAddress()`	Возвращает объект InetAddress , который содержит IP адрес локального хоста, к которым объект DatagramSocket соединяется.
`public int getLocalPort()`	Возвращает integer значение, которое представляет порт локального хоста, с которым объект DatagramSocket связывается.
`public void bind(SocketAddress address)`	Связывает объект DatagramSocket с объектом SocketAddress.
`public void close()`	Закрывает объект DatagramSocket.
`public void connect(InetAddress address, int port)`	Соединяет объект DatagramSocket с заданным IP адресом и портом.
`public void disconnect()`	Разъединяет объект DatagramSocket.
`public boolean isBound()`	Возвращает истину, когда объект DatagramSocket связан с портом.
`public boolean isClosed()`	Возвращает истину, когда объект DatagramSocket закрывается.

`public boolean isConnected()`	Возвращает истину, когда объект DatagramSocket соединяется с IP адресом.
`public void receive( DatagramPacket packet)`	Получает датаграммный пакет от текущего объекта DatagramSocket.
`public void send(DatagramPacket packet)`	Передает датаграммный пакет от текущего объекта DatagramSocket.

Сервер UDP

Сервер UDP представляет собой сетевое приложение, которое использует протокол UDP для обслуживания запросов клиентских приложений. Для создания сервера UDP используется объект DatagramSocket, который принимает объекты DatagramPacket от клиентов. Для создания сервера UDP выполняются следующие шаги:

- Создайте сокет, используя объект DatagramSocket.
- Создайте объект класса DatagramPacket и используйте метод receive(), чтобы получить сообщение, посланное клиентом.
- Создайте объект класса DatagramPacket и используйте метод send(), чтобы послать сообщение клиенту.
- Стартуйте сервер, вызывая конструктор класса сервера UDP в методе main().

Можно использовать следующий фрагмент кода для создания объекта DatagramSocket:

```
try
{
   DatagramSocket socket = new DatagramSocket(1501);
}
catch(SocketException se)
{
   System.out.println("Error");
}
```

В предыдущем фрагменте кода объект `socket` класса `DatagramSocket` связывается с портом номер `1501`.

Объект DatagramPacket, который получает датаграммный пакет, содержит буфер для хранения датаграмм. Можно использовать следующий фрагмент кода для создания объект DatagramPacket, который принимает датаграммные пакеты:

```
try
{
DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
socket.receive(packet);
}
catch(Exception e){
```

```
System.out.println("Error");
}
```

В предыдущем фрагменте кода создается объект **packet** класса DatagramPacket, который вызывает метод **receive()** для получения пакет от объекта сокет.

Объект DatagramPacket, посланный получателю отличается от объекта полученных данных. Этот объект DatagramPacket содержит IP адрес и номер порта компьютера, откуда пакет посылается. Можно использовать следующий фрагмент кода, чтобы послать объект DatagramPacket на заданный адрес:

```
try
{
DatagramPacket packet = new DatagramPacket(buffer, length,
address, port);
socket.send(packet);
}
catch(Exception e)
{
System.out.println("Error");
}
```

В предыдущем фрагменте кода, создается новый объект **DatagramPacket**, который принимает четыре следующие параметра:

- **buffer**: Задает буфер, который хранит данные.
- **length**: Задает длину буфера в байтах.
- **address**: Задает адрес, на который датаграмма должна быть отправлена.
- **port**: Задает номер порта, который удаленный компьютер использует для получения датаграммы.

Метод send() класса DatagramSocket посылает адресату объект DatagramPacket.

Запустите сервер UDP, вызывая конструктор класса в методе main(). Можно использовать следующий фрагмент кода, чтобы запустить сервер UDP:

```
public static void main(String args[]) throws Exception
{
/* Запуск сервера */
   new UDPServer();
}
```

В предыдущем фрагменте кода, создается объект класса **UDPServer**, который запускает серверное приложение UDP.

Следующий код создать UDPServer, который отображает полученные сообщения от клиента и посылает клиенту ответные сообщения:

```
import java.io.*;
import java.net.*;
public class UDPServer
{
   /* Объявляются переменные */
```

```
        DatagramSocket socket = null;
        BufferedReader in = null;
        String str = null;
        byte[] buffer ;
        DatagramPacket packet;
        InetAddress address;
        int port;
        /* Конструктор класса UDPServer */
        public UDPServer() throws IOException
              {
         /* Создается объект DatagramSocket, который получает запросы
клиента
          на номер порта 1501 */
           socket = new DatagramSocket(1501);

        /* Вызывается метод call()*/
           call();
        }
        public void call()
              {
          try
          {
           while (true)
           {
                 buffer= new byte[256];
                 /* Инициализируется объект DatagramPacket */
                 packet = new DatagramPacket( buffer, buffer.length);
                 /* Посылается пакет датаграмм, используя метод
receive()
                  класса DatagramSocket */
                 socket.receive(packet);
                 if(packet == null) break;
                 System.out.println("Введите строку для отправки клиенту
");
                 try
           {
           /*Создается входной поток, который считывает данные с консо-
     ли*/
                    in = new BufferedReader(new InputStream-
              Reader(System.in));
                 }
           catch(Exception e)
           {
                    System.out.println("Error : " + e);
                 }
                 str = in.readLine();
                 buffer = str.getBytes();
                 address = packet.getAddress();
                 port = packet.getPort();
                 packet = new DatagramPacket(buffer, buffer.length, ad-
           dress, port);
           /* Посылается датаграммный пакет */
                 socket.send(packet);
     }
     /* Закрывается поток и сокет */
           in.close();
            socket.close();
         }
```

```
        catch(Exception e)
        {
         System.out.println("Error : " + e);
        }
    }
    public static void main(String args[]) throws Exception
    {
/* Запускается сервер */
       new UDPServer();
    }
}
```

В предыдущем коде создается socket объект `DatagramSocket` в конструкторе класса `UDPServer`. После инициализации сокета на порт `1501`, вызывается метод `call()`. Метод `call()` состоит из методов `receive()` и `send()`, которые управляют клиент/серверным взаимодействием. Предыдущий код сохраняется как `UDPServer.java`. Рис. 28 отображает окно консоли UDPServer.

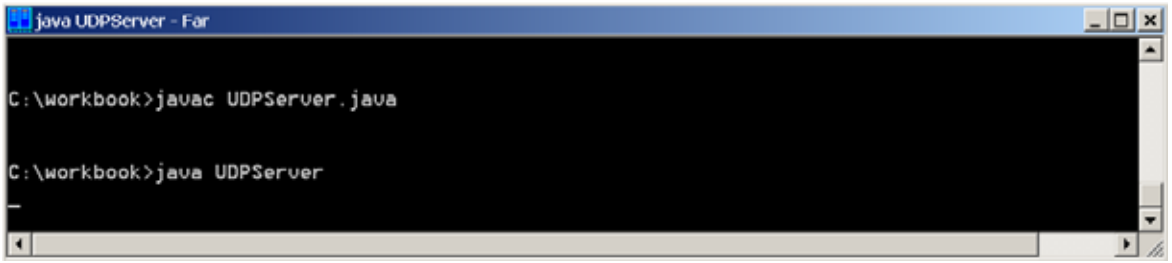


Рис. 28. Окно консоли UDPServer

Клиент UDP

Клиент UDP представляет собой приложение, которое использует протокол UDP, чтобы посылать запросы и получать ответы от приложения сервера. В приложении клиента UDP, Вам необходимо создать объект класса DatagramSocket, который принимает сообщения от сервера UDP. Для создания клиентского приложения UDP необходимо выполнить следующие шаги:

1. Создайте сокет, используя объект DatagramSocket, чтобы установить соединение с сервером.
2. Создайте объект класса DatagramPacket и используйте метод send(), чтобы посылать сообщения на сервер.
3. Создайте объект класса DatagramPacket и используйте метод receive(), чтобы принимать сообщения, посланные сервером.

Следующий фрагмент кода используется, чтобы создать объект DatagramSocket для клиента:

```
try
{
    DatagramSocket socket = new DatagramSocket();
}
catch(Exception e)
```

```
{
System.out.println("Error");
}
```

В предыдущем фрагменте кода, конструктор связывает объект `DatagramSocket` с любым доступным локальным портом, поскольку не указан номера порта в качестве параметра.

Этот объект DatagramPacket содержит IP адрес и номер порта сервера, куда посылается запрос. Следующий фрагмент кода используется для посылки объекта DatagramPacket на заданный сервер:

```
try
{
DatagramPacket packet = new DatagramPacket(buffer, length, address, port);
socket.send(packet);
}
catch(Exception e)
{
System.out.println("Error");
}
```

В предыдущем фрагменте кода создается новый объект `DatagramPacket`, который принимает четыре параметра. Метод `send()` класса `DatagramSocket` посылает объект `DatagramPacket` на сервер.

Создается объект DatagramPacket, который принимает ответ от сервера. Следующий фрагмент кода используется, чтобы создать объект DatagramPacket, который принимает пакеты датаграмм от сервера:

```
try
{
DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
socket.receive(packet);
str = new String(packet.getData());
System.out.println("Принятое сообщение : "+str);

}
catch(Exception e){
System.out.println("Ошибка");
}
```

В предыдущем фрагменте кода создается объект пакета DatagramPacket, который вызывает метод `receive()`. Метод `getData()` принимает данные из пакета и сохраняет сообщение в `string`.

Следующий код используется для создания класса UDPClient, который посылает и принимает сообщения от UDPServer:

```
import java.io.*;
import java.net.*;
public class UDPClient
{
   /* Объявляются переменные */
   static DatagramSocket socket;
   static InetAddress address;
   static byte[] buffer;
   static DatagramPacket packet;
   static String str, str2;
```

```
   static BufferedReader br;
   public static void main(String arg[]) throws Exception{
/* Создается входной поток, который читается с консоли */
      br = new BufferedReader(new InputStreamReader(System.in));
      while(true)
{
      /* Создается новый объект DatagramSocket и связывается с
портом по умолчанию */
            socket = new DatagramSocket();
            address = InetAddress.getByName("127.0.0.1");
            buffer = new byte[256];
            packet = new DatagramPacket(buffer, buffer.length, ad-
      dress, 1501);
      /* Посылается DatagramPacket на сервер */
            socket.send(packet);
            System.out.println("Посылаем запрос");
            packet = new DatagramPacket(buffer, buffer.length);
      /* Принимается DatagramPacket от сервера */
            socket.receive(packet);
      /* Принимаются данные из объекта пакета датаграмм и*/
            str = new String(packet.getData());
            System.out.println("Принятое сообщение : "+str);
            System.out.println("Вы хотите продолжить (Да/Нет) : ");
            str2 = br.readLine();
      /* Выход из цикла while */
            if(str2.equals("Нет")) break;
      }
/* Закрывается объект сокет */
      socket.close();
   }
}
```

В предыдущем коде, цикл `while` метода `main()` в `UDPClient` посылает и принимает запросы от `UDPServer`. Метод `send()` объекта сокета посылает запросы к серверу, а метод `receive()` принимает сообщения от `UDPServer`. Код выше сохраняется как `UDPClient.java`. Рис. 29 отображает вывод `UDPClient`:

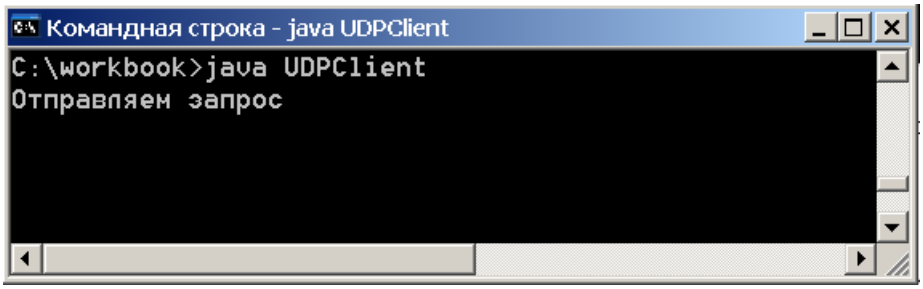


Рис. 29. Вывод UDPClient

На предыдущем рисунке строка «Отправляем запрос» показывает, что клиент послал запрос на UDPServer. UDPServer показывает сообщение «Введите строку для отправки клиенту» в окне консоли после принятия запроса от клиента.

Рис. 30 показывает вывод UDPServer, когда сервер получает запрос от пользователя:

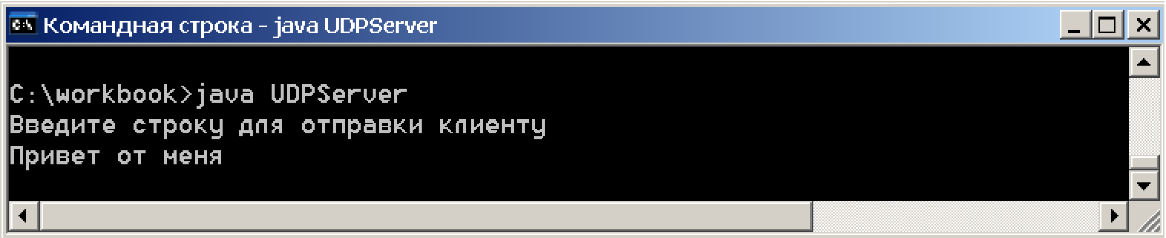


Рис. 30. Получение запроса от клиента

UDPServer принимает строковое сообщение как ввод в окне консоли и посылает строку клиенту. Рис. 31 показывает строку, когда сервер посылает «Привет от меня» клиенту:

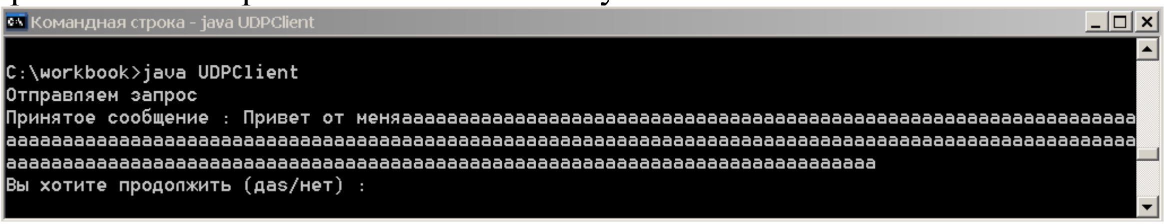


Рис. 31. Получение сообщения от сервера

Когда конечный пользователь вводит «Нет» в окне клиентской консоли завершается процесс UDPClient.

Вопросы для самопроверки

1. Какая модель представляет собой архитектуру разработки приложения, разработанную, чтобы отделять представление данных от их внутренней обработки и хранения?
2. ____________ запрашивает сервисы, и сервер обслуживает эти запросы.
3. ________________ _______________ представляют собой набор правил и соглашений, выполняемых системами, которые общаются через сеть.
4. Интернет протокол (Internet protocol) (IP) приписывает каждому сетевому устройству логический адрес, называемый __ ________.
5. Какая абстракция используются для управления линиями связи между приложениями через сеть
6. Какой протокол обеспечивает надежный канал соединения точка-точка для клиент/серверных приложений для взаимодействия между приложениями. (Протокол управления передачей .
7. Какой пакет языка программирования Java содержит классы и интерфейсы, которые обеспечивают сетевую поддержку?
8. Какие операции обеспечивают методы сетевых классов?
9. Перечислите обычно используемые классы, представленные в

пакете java.net.

10. Перечислите обычно используемые классы исключительных ситуаций, представленные в пакете java.net.
11. Какой класс дает Вам возможность определить имя хоста и связанные IP адреса, требуемые во время создания сетевых приложений?
12. Какой протокол является быстрым, ненадежным протоколом без установления соединения.
13. Какой объект является контейнером данных, содержащим пакеты датаграмм, которые посылаются или принимаются через сеть?
14. Перечислите обычно используемые методы класса DatagramPacket.
15. Какой класс инкапсулирует функциональность для управления объектами DatagramPacket?
16. Перечислите обычно используемые методы класса, представленные в DatagramSocket.

Для освоения материала предлагается выполнить лабораторную работу № 2, часть 1 в разделе Лабораторный практикум.

Создание сетевых приложений с использованием TCP/IP

Java поддерживает классы и методы, которые позволяют устанавливать соединение с удаленным компьютером, используя протокол TCP. В отличие от UDP, TCP является протоколом, ориентированным на установление соединения, которое гарантирует надежную связь между приложениями клиента и сервера. Взаимодействие с использованием протокола TCP, начинается тогда, когда установлено соединение между сокетами клиента и сервера. Сокет сервера слушает запросы на соединение, посланные сокетами клиента и устанавливает соединение. После установления соединения между приложениями клиента и сервера, они могут взаимодействовать друг с другом.

Java упрощает сетевое программирование, путем инкапсуляции функциональности соединения сокета TCP в классы сокета, в которых класс Socket, чтобы создать сокет клиента, а класс ServerSocket, чтобы создать сокет сервера.

Идентификация методов классов `Socket` и `ServerSocket`

Socket является базовым классом и поддерживает протокол TCP. Класс Socket обеспечивает методы для потока ввода/вывода (I/O), кото-

рые способствует удобству чтения и записи в сокет. Этот класс является обязательным для программ, которые выполняют сетевое взаимодействие.

Для создания объектов класса Socket используются следующие конструкторы, определенные в классе Socket:

- public Socket(InetAddress IP_address, int port): Создает объект Socket который соединяет компьютером хоста, заданными в списке параметров как IP_address и port.
- public Socket(String hostname, int port): Инициализирует объект Socket и принимает два параметра, имя хоста или IP адрес, переданные как строка символов, который сервер слушает.

Некоторые полезные методы класса Socket представлены в табл. 10.

Таблица 10

Метод	***Описание***
`public InetAddress getInetAddress()`	Возвращает объект InetAddress, который содержит IP адрес, с которым соединяется объект Socket.
`public InputStream getInputStream()`	Возвращает входной поток для объекта Socket.
`public InetAddress getLocalAddress()`	Возвращает объект InetAddress, который содержит локальный адрес, с которым соединяется объект Socket.
`public int getPort()`	Возвращает удаленный порт, с которым соединяется объект Socket.
`public int getLocalPort()`	Возвращает локальный порт, с которым соединяется объект Socket.
`public OutputStream getOutputStream()`	Возвращает выходной поток объекта Socket.
`void close()`	Закрывает объект Socket.
`public String toString()`	Возвращает IP адрес и номер порта сокета клиента как String.

Конструкторы и метод `close()` класса `Socket` в случае ошибки генерируют `IOException`, которая должна быть перехвачена и обработана.

ServerSocket представляет собой класс, используемый программами сервера для прослушивания запросов клиентов. ServerSocket реально не выполняет сервис, а создает объект Socket от имени клиента. Общение выполняется через созданные объекты.

Используются следующие конструкторы, определенные в классе ServerSocket, чтобы создавать и инициализировать объекты ServerSocket:

- public ServerSocket(int port_number): Создает сокет сервера на заданный порт на локальной машине. Клиентам следует использовать этот порт, чтобы общаться с сервером. Если порт задан как 0, то сокет сервера создается на любой свободный порт локальной машины.
- public ServerSocket(int port, int backlog): Создает сокет сервера на заданный порт на локальной машине. Второй параметр задает максисальное количество соединений клиентов, которые сокет сервера поддерживает на заданном порту.
- public ServerSocket(int port, int backlog, InetAddress bindAddr): Создает сокет сервера на заданный порт. Третий параметр используется, чтобы создать сокет сервера хоста, подключенного к нескольким физическим линиям (multi-homed host). Сокет сервера принимает запросы клиента только с заданных IP адресов.

Полезные методы класса ServerSocket представлены в табл. 11.

Таблица 11

Метод	***Описание***
`public InetAddress getInetAddress()`	Возвращает объект InetAddress, который содержит адрес объекта ServerSocket.
`public int getLocalPort()`	Возвращает номер порта, с которого объект ServerSocket слушает запросы клиента.
`public Socket accept() throws IOException`	Заставляет сокет сервера слушать соединения клиента и принимать его. После установки соединения клиента с сервером метод возвращает сокет клиента.
`public void bind( SocketAddress address) throws IOException`	Связывает объект ServerSocket с заданным адресом (IP адрес и порт). Этот метод вызывает исключительную ситуацию IOException, когда происходит ошибка.
`public void close() throws IOException`	Закрывает объект ServerSocket. Этот метод вызывает исключительную ситуацию IOException, когда происходит ошибка.
`public String toString()`	Возвращает IP адрес и номер порта сокета сервера как String.

Создание сервера TCP/IP

Процедура создания сервера состоит создании объекта ServerSocket, который слушает клиентские запросы с определенного порта. Когда сервер распознает допустимый запрос, объект сокет сервера получает объект Socket, созданный клиентом. Взаимодействие между сервером и клиентом происходит с использованием этого сокета.

Класс ServerSocket пакета java.net используется, чтобы создать сокет, с которого сервер слушает запросы удаленного входа. Класс BufferedInputStream управляет передачей данных от клиента к серверу, а класс PrintStream управляет передачей данных от сервера к клиенту.

Метод accept() ожидает соединения клиента, прослушивая порт, с которым он связан. Когда клиент пытается соединиться с сокетом сервера, метод принимает соединение и возвращает сокет клиента. Позже сокет используется для общения между сервером и его клиентом. Выходной поток этого сокета является входным потоком для соединенного клиента и наоборот. IOException генерируется, если происходит какая-либо ошибка во время установления соединения. Java вынуждает обрабатывать возникающие исключительные ситуации. Чтобы создать приложение сервера сокета TCP, требуется выполнить следующие шаги:

- Создайте объект сокета сервера ServerSocket.
- Прослушивайте запросы клиента на соединение.
- Запустите сервер.
- Создайте поток соединения для запросов клиентов.

1. Создание Server

Класс Server расширяет класс Thread. Объект ServerSocket слешает запросы клиентов. Конструктор класса Server создает объект ServerSocket. Отображается сообщение об ошибке, если возникает исключительная ситуация, когда запускается сервер.

Фрагмент кода для конструктора выглядит следующим образом:

```
public Server()
{
        try
        {
            serverSocket = new ServerSocket(1001);
        }
        catch(IOException e)
        {
            fail(e, " Не могу запустить сервер.");
        }
        System.out.println("Сервер запущен. . .");
        this.start();    // Запускается поток
}
```

В предыдущем фрагменте года определяется общий метод обработки ошибок `fail()`, чтобы обеспечивать обработку всех исключительных ситуаций. Метод принимает два аргумента (объект `Exception` и объект `String`) и выводит сообщение об ошибке. Фрагмент кода для метода `fail()` выглядит следующим образом:

```
public static void fail(Exception e, String str)
{
        System.out.println(str + "." + e);
}
```

2. Прослушивание запроса клиента

Метод run() сервера, аналогичен любому потоку, который применяет интерфейс Runnable и имеет инструкции для потока. В этом случае сервер переходит в бесконечный цикл и слушает запросы клиента. Когда сервер защищает соединение с клиентом, метод accept() класса ServerSocket принимает соединение. Сервер создает для клиента объект класса Connection, определенного пользователем. Объект класса Socket передается конструктору класса Connection. Взаимодействие между клиентом и сервером выполняется через этот сокет. Фрагмент кода для метода run() выглядит следующим образом:

```
public void run()
{
        try
        {
            while(true)
            {
                Socket client = serverSocket.accept();
                Connection con = new Connection(client);
            }
        }
        catch(IOException e)
        {
            fail(e, "Не прослушивается");
        }
}
```

3. Запуск сервера

Фрагмент кода для метода main() приведен ниже.

```
public static void main(String args[])
{
        new Server();
}
```

В предыдущем фрагменте кода, создается объект класса `Server`, который запускает поток.

4. Создание потока соединения

Следующий фрагмент кода описывает класс Connection:

```
      class Connection extends Thread
      {
        protected Socket netClient;
        protected BufferedReader fromClient;
        protected PrintStream toClient;
        public Connection(Socket client)
        {
             netClient = client;
             try
             {
      fromClient = new BufferedReader(new           InputStream-
Reader(netClient.getInputStream()));
      toClient = new PrintStream(netClient.getOutputStream());
             }
             catch(IOException e)
             {
                 try
                   {
                     netClient.close();
                   }
                   catch(IOException e1)
                   {
                     System.err.println("Unable to set up streams"
+ e1);
                     return;
                   }
             }
             this.start();
        }
        public void run()
        {
           String clientMessage;
           try
           {
               for(;;)
               {
                  clientMessage = fromClient.readLine();
                  if(clientMessage == null)
                  break;
                  // Посылает подтверждение клиенту
                  toClient.println("Received");
               }
           }
           catch(IOException e)
           {}
           finally
           {
             try
              {
                netClient.close();
              }
              catch(IOException e)
          {}
         }
       }
      }
```

В предыдущем фрагменте кода класс **Connection** создает объект **BufferedReader** (**fromClient**), который получает ввод от клиента, использующего метод **getInputStream()**. Объект класса **PrintStream** (**toClient**) дает возможность серверу писать клиенту, используя метод **getOutputStream()**. Таким образом, возникают две возможности взаимодействия.

Когда клиент соединяется с сервером, сервер использует метод **readLine()** объекта **fromClien**t, чтобы сохранить сообщение, посланное клиентом в строчной переменной типа String **clientMessage**. Метод **println()** используется, чтобы написать сообщение “Received ” сокету.

Для выхода из системы сервер завершает цикл. Это вызывает блок **finally** для выполнения, закрывая сокет клиента. Закрытие сокета является важным, поскольку сохранение соединения неизбежно приводит к потери памяти сервера. Блок **finally** гарантирует, что соединение закрывается. Отметим, что сервер является многопоточным, и каждый клиент получает свой собственный поток на сервере.

Следующий код используется, чтобы создать класс Server, который принимает запросы соединения клиента и посылает Login строку как ответное сообщение клиенту:

```
import java.io.*;
import java.net.*;
public class Server extends Thread
{
       ServerSocket serverSocket;
       public Server()
       {
       try
       {
            /* Создание объекта ServerSocket, который принимает
      запросы соединения от клиентов от порта 1001*/
            serverSocket = new ServerSocket(1001);
            System.out.println(serverSocket.toString());
       }
       catch(IOException e)
       {
           fail(e, "Не возможно запустить сервер.");
       }
       System.out.println("Сервер запущен . . .");
/* Стартует поток */
       this.start();          }
     public static void fail(Exception e, String str)
     {
      System.out.println(str + "." + e);
     }
      public void run()
      {
      try
      {
          while(true)
          {
```

```
            /* Принимаются запросы от клиентов */
                    Socket client = serverSocket.accept();
                     /* Создается объект соединение, чтобы управлять взаи-
            модействием кдиента с сервером */
                    Connection con = new Connection(client);
                  }
             }
             catch(IOException e)
             {
                 fail(e, "Не прослушивается");
             }
             }
             public static void main(String args[])
             {
      /* Запускается сервер */
             new Server();
             }
      }
      class Connection extends Thread
      {
      /* Declare the variables */
        protected Socket netClient;
        protected BufferedReader fromClient;
        protected PrintStream toClient;
        public Connection(Socket client)
        {
             netClient = client;
             try
             {
      /* Создается входной поток, чтобы принимать данные от клиента */
            fromClient = new BufferedReader(new             InputStream-
      Reader(netClient.getInputStream()));
      /* Создается выходной поток, чтобы посылать данные клиенту */
      toClient = new PrintStream(netClient.getOutputStream());
             }
             catch(IOException e)
              {
                 try
                   {
            /* Закрывается сокет клиента */
                    netClient.close();
                   }
                   catch(IOException e1)
                   {
                     System.err.println("Невозможно установить потоки" +
e1);
                     return;
                   }
              }
      /* Start the thread */
             this.start();
        }
        public void run()
        {
           String login, password;
           try
           {
               for(;;)
```

```
            {
                        toClient.println("Login: ");
        /* Принимается login как ввод от клиента */
                    login = fromClient.readLine();
        /* Завершить соединение, когда Bye вводится как login */
                    if(login == null||login.equals("Bye")){
              System.out.println("Выход");
              return;
                    }else
                    System.out.println(login + " logged in");
                    // Посылается подтверждение клиенту
                        toClient.println("Password: ");
        /* Принимается пароль то клиента */
              password = fromClient.readLine();
            }
          }
          catch(IOException e)
          {}
          finally   {
            try
             {
               netClient.close();
             }
             catch(IOException e)
       {}
      }
   }
}
```

Предыдущий код сохраняется как Server.java. Рис.32 показывает вывод окна консоли сервера.

Рис. 32. Вывод окна консоли сервера Создание клиента TCP/IP

Создание клиента TCP/IP

Первый шаг создания клиента сотстоит в том, чтобы установить соединение между клиентом и сервером. Для этого необходимо создать объект Socket с помощью следующих задач:

1. Создайте сокет клиента, используя объект Socket
2. Читайте и записывйте в сокет
3. Закройте соединение

1. Создание сокета клиента

Сокет клиента создается конструктором класса Socket, который принимает два параметра, IP адрес и номер порта, которые прослушивает сервер. Следующий фрагмент кода используется для создания сокета клиента:

```
try
{
    Socket clientSocket = new Socket("127.0.0.1", 1001);
}
catch(UnknownHostException e)
{
System.err.println("Unidentified hostname ");
System.exit(1);
}
```

В предыдущем фрагменте кода IP адрес равный ‘127.0.0.1’ и порт равный ‘1001’ представляют сокет, на котором сервер прослушивает запросы клиента.

2. Чтение и запись в сокет

После установления соединения между клиентом и сервером, клиент посылает запрос на сервер через сокет. Чтение и запись в сокет аналогичны чтению из файла и записи в файл. Чтобы позволить клиенту общаться с сервером:

- Объявите два объекта по одному из классов PrintStream и BufferedReader. Эти объекты будут использованы для чтения из и записи в сокет socket.

```
PrintStream out = null;       // Запись в сокет
BufferedReader in = null;     // Чтение из сокета
```

- Свяжите объекты PrintStream и BufferedReader с socket.

```
out = new PrintStream(clientSocket.getOutputStream());
in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
```

Методы `getInputStream()` и `getOutputStream()` класса `Socket` дают клиенту возможность общаться с сервером. Метод `getInputStream()` дает возможность объекту `BufferedReader` читать из `socket` и метод `getOutputStream()` дает возможность объекту `PrintStream` записывать в socket.

Объявите еще один объект из класса `BufferdReader` , чтобы связать со стандартныым входом, таким образом, что данные, введенные на клиенте, могут быть переданы на сервер. Следующий код используется для чтения данных из окна консоли:

```
    BufferdReader stdin = new BufferedReader(new InputStream-
Reader(System.in));
    String str;
```

```
while((str = stdin.readLine()).length() != 0)
{
out.println(str);
}
```

Предыдущий фрагмент кода позволяет пользователю принимать данные с клавиатуры. Цикл **while** продолжается пока пользователь не введет символ завершения ввода.

3. Закрытие соединения

Операторы ниже закрывают потоки и соединения с сервером.

```
out.close();
in.close();
stdin.close();
```

Сокет клиента принимает имя пользователя и пароль и разрешает общение. Чтобы завершить соединение, пользователь должен ввести ‘Bye’. Следующий код можно использовать, чтобы создать класс Client:

```
/* Программа для применения простого клиентского сокета */
import java.net.*;
import java.io.*;
public class Client
{
      public static void main(String []args) throws IOException
{
            Socket clientSocket;
            PrintStream out = null;
            BufferedReader in = null;
            try
{
      /* Создается объект сокет, чтобы соединиться с сервером */
                  clientSocket = new Socket("192.168.0.70", 1001);
/* Создается выходной поток, чтобы посылать данные на сервер */
out = new PrintStream(clientSocket.getOutputStream());
/* Создается входной поток, чтобы принимать данные с сервера */
in = new BufferedReader(new InputStream-
Reader(clientSocket.getInputStream()));
}
catch(UnknownHostException e)
{
      System.err.println("Unidentified hostname ");
      System.exit(1);
}
catch(IOException e)
{
      System.err.println("Couldn't get I/O ");
      System.exit(1);
}
/* Создается входной поток, чтобы читать данные из окна консоли */
BufferedReader stdin = new BufferedReader(new InputStream-
Reader((System.in)));
/* Чтение из сокета */
String login = in.readLine();
System.out.println(login);
/* Прием login */
```

```
String logName = stdin.readLine();
out.println(logName);
/* Чтение из сокета */
String password = in.readLine();
System.out.println(password);
/* Прием  password */
String pass = stdin.readLine();
out.println(pass);
String str = in.readLine();
System.out.println(str);
while((str = stdin.readLine()) != null)
{
      out.println(str);
      if(str.equals("Bye"))
      break;
}
out.close();
in.close();
stdin.close();
}
}
```

Представленный выше код сохраняется как Client.java. Когда Вы выполняете приложение Client, отображается сообщение для ввода login. После приема login, на консоли класса Client появляется сообщение для ввода password. Рис. 33 показывает вывод, когда выполняется класс Client, чтобы послать John login и password как вход на Server.

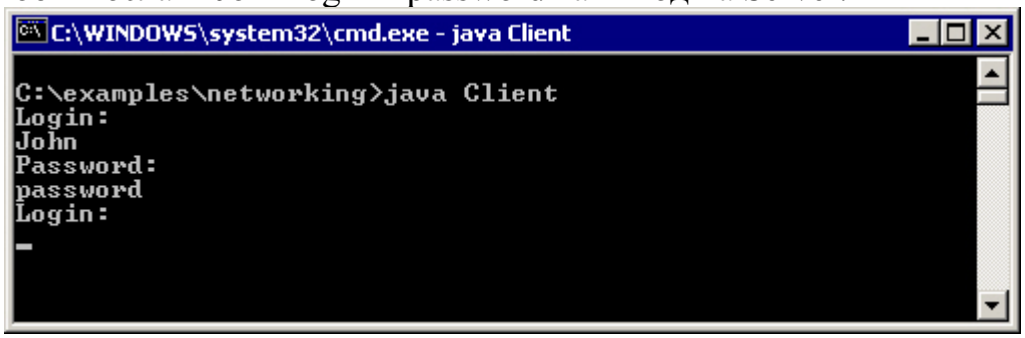


Рис. 33. Запущенный Client

Введенный login в окне консоли класса Client передается классу Server. Рис. 34 отображает класс Server , когда он получает login John от класса Client.

Рис. 34. Получение Login от Client

Когда передается строка Bye как login от клиента на сервер, общение между клиентом и сервером прекращается. Рис. 35 отображает вывод класса Server, когда строка Bye передается как login на сервер от класса Client:

Рис. 35. Client завершает соединение с Server

Вопросы для самопроверки

1. Какой пакет содержит классы для создания сетевых приложений TCP/IP?
2. Какой класс обеспечивает функциональность для создания сокетов для клиента?
3. Какой класс обеспечивает функциональность для создания сокетов для сервера?
4. Перечислите широко используемые методы класса Socket.
5. Перечислите широко используемые методы класса ServerSocket.

Для освоения материала предлагается выполнить лабораторную работу № 2, часть 2 в разделе Лабораторный практикум.

Введение в RMI

Обзор распределенных приложений

В течение последних лет существуют три различных подхода к разработке приложений – традиционный подход, клиент/серверный подход и компонентный подход.

В традиционном подходе единственное приложение управляет логикой представления, логикой обработки и взаимодействием с базой данных (обычно набор файлов), которая размещается также на локальном компьютере. Такие приложения также называются монолитными. Недостатком такого подхода является то, что даже при незначительных изменениях, расширении и развитии, требуемых в приложении, все приложение должно было перекомпилироваться и собираться снова. Это делает очень высокой стоимость изменения и распространения приложения.

С целью устранения недостатков традиционного подхода, была введена клиент/серверная архитектура (также называемая двух-слойная (two-tier) архитектура). В этой архитектуре данные отделяются от клиентской части, хранятся централизованно и предоставляют доступ по технологии клиент/сервера. Логика обработки объединяется с логикой представления либо на стороне клиента, либо на стороне сервера, которая имеет средства для связи с базой данных. Если логика обработки объединяется с логикой представления, то клиент называется толстым клиентом. Если логика обработки объединяется с сервером базы данных, то сервер называется толстым сервером.

Однако клиент/серверная архитектура также обладает рядом недостатков:

- Любое изменение бизнес-логики требует изменений в алгоритмов обработки. Чтобы изменить алгоритмы обработки, вся логика представления или средства связи с базой данных нуждается в изменении, в зависимости от места логики обработки.
- Реализованные приложения, использующие двухслойную архитектуру, могут трудно масштабироваться из-за ограниченного количества доступных для клиента соединителей с базой данных. Запросы соединения, превышающие определенную границу, просто отбрасываются сервером.

Недостатки клиент/серверной архитектуры привели к разработке трехслойной архитектуры, в которой логика представления размещается на стороне клиента, доступ к базе данных контролируется на стороне сервера, и логика обработки находится между двумя слоями. Слой логики обработки относится к серверу приложений (также называемый средним слоем трехслойной компонентной архитектуры). Этот тип архитектуры называется серверо-центрическим, поскольку он дает возможность компонентам приложения выполняться на среднем слое сервера приложений, применяющего правила обработки независимо от интерфейса представления и реализации базы данных. Эти компоненты могут быть разработаны, используя любой язык программирования, который позволяет создание компонентов. Компоненты могут быть централизованы для упрощения разработки, поддержки и развертывания. Так как средний слой управляет логикой обработки, нагрузка распределяется между клиентом, сервером базы данных и сервером, управляющим логикой обработки. Эта архитектура также обеспечивает эффективный доступ к данным. Проблема с ограничением соединений базы данных минимизируется, так как база данных видит только слой логики обработки и не всех ее клиентов. При этом в случае двухслойного приложения соединение базы данных устанавливается заблаговременно и

поддерживается, в то время как в трехслойном приложении (рис. 36) соединение устанавливается только когда требуется доступ к данным и освобождается, как только данные получены или переданы на сервер.

Рис. 36. Распределенное приложение

Приложения, в которых логика представления, логика обработки и база данных размещаются на нескольких компьютерах, называется распределенными (distributed) приложениями.

Метод удаленного вызова

Мы рассмотрели как обеспечивается взаимодействие процессов с использованием сокетов. Еще одним способом взаимодействия является вызов удаленного метода (Remote Method Invocation) (RMI). RMI представляет собой спецификацию, которая дает возможность одной виртуальной Java-машине – Java Virtual Machine (JVM), вызывать методы в объектах, расположенных в другой JVM. Эти две JVM могут быть запущены на различных компьютерах или выполняться в отдельных процессах одного компьютера. RMI применяется на среднем слое трехслойного архитектуры, облегчая возможность программистам вызывать распределенные компоненты в сетевой среде.

Sun Microsystems ввела RMI, как простую альтернативу сложному кодированию, связанному с программированием сокета сервера. Для использования RMI, программисту нет необходимости знать программирование сокета или многопоточное исполнение, и необходимо, единственно, концентрироваться на разработку логики обработки.

Компоненты приложения RMI

Распределенное RMI приложение содержит два компонента:

- RMI-сервер
- RMI-клиент

RMI-сервер содержит объекты, методы которых должны вызываться удаленно. Сервер создает несколько удаленных объектов и делает ссылку на эти объекты в регистре RMI. Регистр RMI является сервисом, который выполняется на сервере RMI. Удаленные объекты, созданные сервером регистрируются в регистре уникальным именем. Клиент получает ссылку на один или несколько удаленных объектов из регистра, просматривая имена объектов. Затем клиент вызывает методы на удаленном объекте(ах), чтобы получить доступ к серверу удаленного объекта(ов). Рис. 37 иллюстрирует функциональность приложений в RMI.

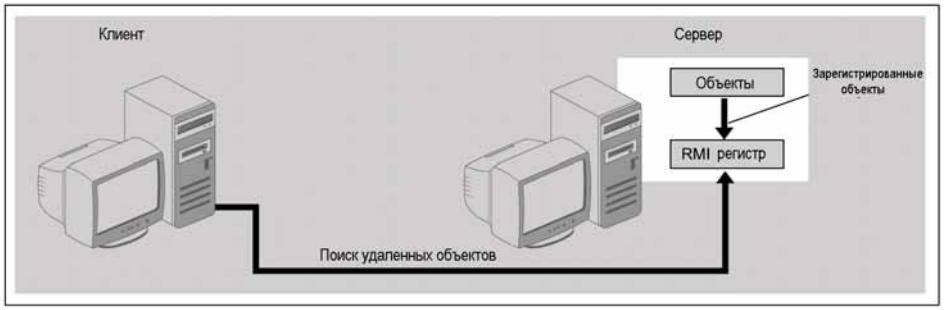


Рис. 37. Функциональность приложений в RMI

Однажды получив ссылку на удаленный объект, в дальнейшем методы в удаленном объекте вызываются точно так же, как методы локального объекта.

Архитектура RMI

Архитектура RMI состоит из трех уровней (рис. 38.):

- Уровень стаб/скелет (Stub/Skeleton)
- Уровень удаленной ссылки (Remote Reference layer)
- Транспортный уровень

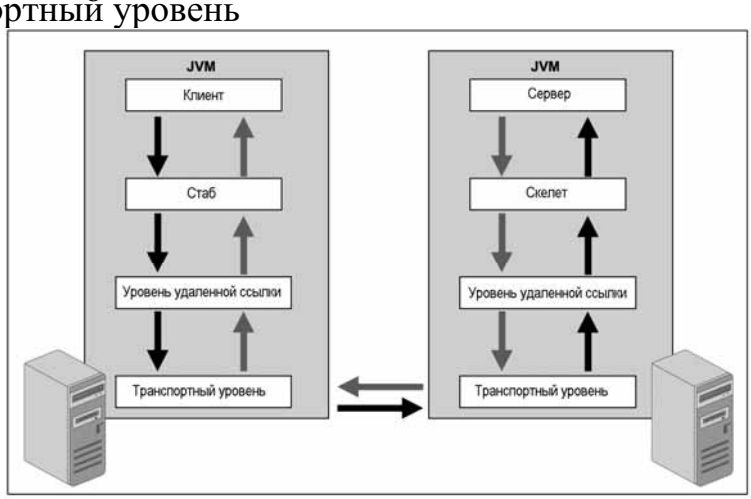


Рис. 38. Архитектура RMI

Уровень стаб/скелет (Stub/Skeleton) слушает вызовы удаленных методов, сделанные клиентом и направляет эти вызовы на удаленные RMI сервисы на сервере. Этот уровень состои из стаб и скелет.

Чтобы вызвать методы удаленного объекта, запрос на клиентской стороне начинается со стаб. Стаб представляет собой прокси (proxy) на стороне клиента, представляющий удаленный объект. К нему обращаются программы, как к любому другому локальному объекту, выполняющемуся на клиенте и он (стаб) обеспечивает методы удаленного объекта. Стаб связывается с удаленным методом объекта через скелет, который реализуется на сервере.

Скелет представляет собой прокси на стороне сервера, который продолжает общение со стаб посредством:

- Чтения параметров для вызова метода
- Выполнения вызова удаленного сервиса объекта реализации
- Получения возвращаемого значения
- Записи возвращенного значения обратно на стаб.

Уровень удаленной ссылки

Уровень удаленной ссылки (Remote Reference Layer) (RRL) интерпретирует и управляет ссылками, сделанными клиентами к удаленному объекту на сервере. Этот уровень представлен как на клиенте, так и на сервере. RRL на стороне клиента принимает запросы для методов от стаб, которые передается, как упакованный поток данных на RRL на стороне сервера. Упаковка представляет собой процесс, в котором параметры, переданные клиентом, преобразуются в формат, который может быть передан по сети. RRL на стороне сервера выполняет распаковку параметров, которые посланы удаленному методу через скелет. Распаковка представляет собой процесс, в котором обработанные ранее упакованные параметры, переданные клиентом RRL клиентской стороны, преобразуются в формат, который скелет понимает. Во время возвращения значений от скелета, данные вновь обрабатываются маршалингом и передаются клиенту через RRL со стороны сервера.

Транспортный уровень

Транспортный уровень представляет собой связь между RRL на стороне сервера и RRL на стороне клиента. Транспортный уровень отвечает за установление нового соединения и управление существующими соединениями. Он также отвечает за управление удаленными объектами, которые размещаются в пространстве адресов транспортного

уровня. Следующие операции разъясняют, каким образом клиент соединяется с сервером:

- При получении запроса от RRL со стороны клиента, транспортный уровень устанавливает соединение сокета с сервером через RRL со стороны сервера.
- Затем, транспортный уровень передает установленное соединение с RRL на стороне клиента и добавляет ссылку на соединение в свою таблицу.

Пакеты RMI

Ниже представлены пакеты RMI, состоящие из различных классов и интерфейсов для создания и запуска различных распределенных приложений, использующих RMI:

- Пакет java.rmi обеспечивает удаленный интерфейс и класс для доступа по удаленным именам, зарегистрированным на сервере и менеджер безопасности для RMI.
- Пакет java.rmi.registry содержит классы и интерфейсы, которые используются удаленным регистром.
- Пакет java.rmi.server содержит классы и интерфейсы, которые применяются удаленными объектами, стабы и скелеты и поддерживают общение RMI.
- Пакет java.rmi.dgc содержит классы и интерфейсы, которые используются RMI распределенным сборщиком мусора (RMI-distributed garbage collector).

Пакет java.rmi

Пакет java.rmi объявляет интерфейс Remote и классы Naming и RMISecurityManager. Он также определяет ряд классов исключительных ситуаций, которые используются с RMI.

Все удаленные объекты должны применять интерфейс Remote. Этот интерфейс не содержит методов и используется для идентификации удаленных объектов. Пакет java.rmi состоит из следующих классов:

- Класс Naming: Содержит статические методы для доступа удаленных объектов через URL. Этот класс поддерживает следующие методы:
  - rebind(): Связывает имя удаленного объекта с заданным URL и обычно используются объектом сервера.
  - unbind(): Удаляет связь между именем объекта и URL.

- lookup(): Возвращает удаленный объект, заданный URL и обычно используется объектом клиента.
- list():Возвращает список URL, которые в настоящее время известны RMI регистру.

- Класс RMISecurityManager: Определяет политику безопасности по умолчанию для удаленного объекта стаб. Политика применяется только для приложений. Апплеты используют класс AppletSecurityManager для RMI. Метод setSecurityManager() класса System используется, чтобы назначить объект RMISecurityManager в качестве менеджера безопасности, который должен быть использован для стаб RMI.

Пакет java.rmi определяет ряд исключительных ситуаций. Класс RemoteException является родительским для всех исключительных ситуаций, которые генерируются во время RMI.

Удаленным объектам, которые доступны локально, нет необходимости вызывать исключительную ситуацию `RemoteException`.

Пакет java.rmi.registry

Пакет java.rmi.registry содержит интерфейсы Registry и RegistryHandler и размещается в регистре. Эти интерфейсы используются, чтобы регистрировать и получать доступ к удаленным объектам по имени. Объекты Remote регистрируются, когда они связываются с процессом регистрации хоста. Процесс регистрации запускается, когда запускается команда start rmiregistry. Команда определяет методы rebind(), unbind(), list() и lookup(), которые используются классом Naming, чтобы связать имена и ссылки URL RMI.

Пакет java.rmi.server

Пакет java.rmi.server реализует интерфейсы и классы, которые поддерживают клиентскую и серверную части RMI. Пакет java.rmi.server состоит из следующих классов и интерфейсов:

- Класс RemoteObject: Реализует интерфейс Remote и обеспечивает удаленную реализацию класса Object. Все объекты, которые применяют удаленные объекты, расширяют класс RemoteObject.
- Класс RemoteServer: Расширяет класс RemoteObject и является общим классом, который подклассом определенных типов реализаций удаленного объекта.
- Класс UnicastRemoteObject: Расширяет класс RemoteServer и обеспечивает реализацию RemoteObject по умолчанию. Клас-

сы, которые реализуют RemoteObject обычно подкласс объекта UnicastRemoteObject. Путем расширения UnicastRemoteObject, подкласс может:

- Использует по умолчанию RMI для общения на основе транспортного сокета.
- Выполняется все время.

- Класс RemoteStub: Обеспечивает абстрактную реализацию стаб на стороне клиента. Статический метод setRef() используется, чтобы связать стаб на стороне клиента с соответствующими удаленными объектами.
- Интерфейс RemoteCall: Обеспечивает методы, которые используются всеми стаб и скелет в RMI.
- Интерфейс Skeleton: Обеспечивает метод, чтобы получать доступ к методам удаленного объекта и этот интерфейс реализуется удаленными скелетом.
- Интерфейс Unreferenced: Дает возможность определить, когда клиент больше не ссылается на удаленный объект. Этот интерфейс реализуется классом Remote.

Пакет java.rmi.dgc

Пакет java.rmi.dgc содержит классы и интерфейсы, которые используются распределенным сборщиком мусора. Серверная сторона распределенного сборщика мусора реализует этот интерфейс. Этот пакет содержит:

- Класс Lease, который создает объекты, используемые для отслеживания ссылок объектов.
- Метод dirty(), который используется, чтобы отмечать, что клиент ссылается на удаленный объект.
- Метод clean(), который используется, чтобы отмечать, что удаленная ссылка завершилась.

Распределенная сборка мусора

Java содержит распределенный механизм сборки мусора, который автоматически удаляет объекты, которые клиенты RMI не используют. RMI использует алгоритм подсчета ссылок для управления ссылками на удаленные объекты. Уровни стаб и скелет используют интерфейс java.rmi.dgc для реализации механизма распределенной сборки мусора. Удаленный объект, который применяет интерфейс java.rmi.server.Unreferenced получает извещение, когда ссылки клиента

больше не существует. Если не существует локальной или удаленной ссылки на удаленный объект, применяться сборка мусора.

Когда какой-либо новый удаленный объект вводится в JVM, выполняющийся объект RMI класса Lease, увеличивает счетчик ссылок на единицу и отмечает удаленный объект как измененный. Аналогично, когда существующий удаленный объект удаляется из JVM, объект класса Lease уменьшает счетчик на единицу и отмечает объект как чистый. В механизме подсчета ссылок удаленная ссылка может быть не активной в JVM в течение выделенного времени. Системное свойство java.rmi.dgc.leaseValue устанавливает выделенное время. Если ссылка не возобновляется соединением к удаленному объекту в течение завершения выделенного времени, RMI перераспределяет память, выделенную удаленному объекту, используя механизм распределенной сборки мусора. Выделенное время устанавливается в миллисекундах и по умолчанию значение выделенного времени равняется 10 минутам.

Создание сервера RMI

Класс сервера RMI содержит объекты, которые вызываются клиентом удаленно. Вам необходимо создать объект в методе main() класса сервера RMI, чтобы создать удаленный объект сервера. Так, например, для создания удаленного объекта класса Hello можно использовать следующий оператор:

```
Hello h = new HelloImpl();
```

Далее, необходимо зарегистрировать объект сервера в регистре самозагрузки перед тем, как объект сервер будет готов принимать запросы от клиента. Вы передаете имя и ссылку объекта сервера в RMI регистр, чтобы зарегистрировать объекты сервера. Имя объекта сервера используется, чтобы получить доступ к объекту стаб, использующему механизм поиска. Вы можете использовать следующий оператор, чтобы зарегистрировать объект сервер в регистре:

```
Naming.rebind("server",h);
```

Метод rebind() принимает два параметра:

- Первый параметр представляет собой стоку URL, которая содержит местоположение и имя удаленного объекта. Если порт не задан, регистр RMI использует по умолчанию порт 1099. Если пользователь должен определить порт, строка URL должна быть изменена на "rmi://ip-address:1234/server".
- Второй параметр является ссылкой на реализацию объекта.

Метод setSecurityManager() класса SecurityManager используется в классе сервера, чтобы установить менеджера безопасности для прило-

жения RMI, так что любой клиент не может вызвать объект сервера. Чтобы установить подлинность клиента RMI client, Вам необходимо создать файл политики безопасности, который содержит все требуемые разрешения. Вы можете использовать следующий код, чтобы создать сервер RMI для распределенного приложения:

```
import java.rmi.*;
import java.rmi.server.*;
public class HelloServer
{
      public static void main(String args[])
      {
            try
            {
                  /* Устанавливается менеджер безопасности */
                  System.setSecurityManager(new     RMISecurityMan-
ager());
/* Создается удаленный объект */
            Hello h = new HelloImpl();
            /* Связывается удаленный объект с регистром RMI */
                  Naming.rebind("server",h);
                  System.out.println("Объект регистрируется.");
System.out.println("Сейчас сервер ожидает запроса клиента...");
            }
            catch(Exception e)
            {
                  System.out.println("Ошибка : "+e);
            }
      }
}
```

В предыдущем коде, метод `rebind()` класса `Naming` вызывает удаленную исключительную ситуацию. В результате, Вы определяете метод `rebind()` внутри блока try-catch.

Политика безопасности

Файл политики безопасности выполнения Java приложений определяет разрешения для кода от различных источников, когда реализуются различные права и представляется объектом Policy. АФайл политики является простым текстом и может быть создан как текстовым редактором так и графической утилитой **Policy Tool**. Создается файл политики с именем java.policy в HOME подкаталоге, который содержит текст, дающий разрешения на требуемые объекты. Следующие шаги создания файла политики безопасности:

1. В командной строке введите **policytool**, чтобы запустить утилиту Policy Tool Java. Окно Policy Tool отображается с сообщение Error (Ошибка). Рис. 39 представляет окно Policy Tool.

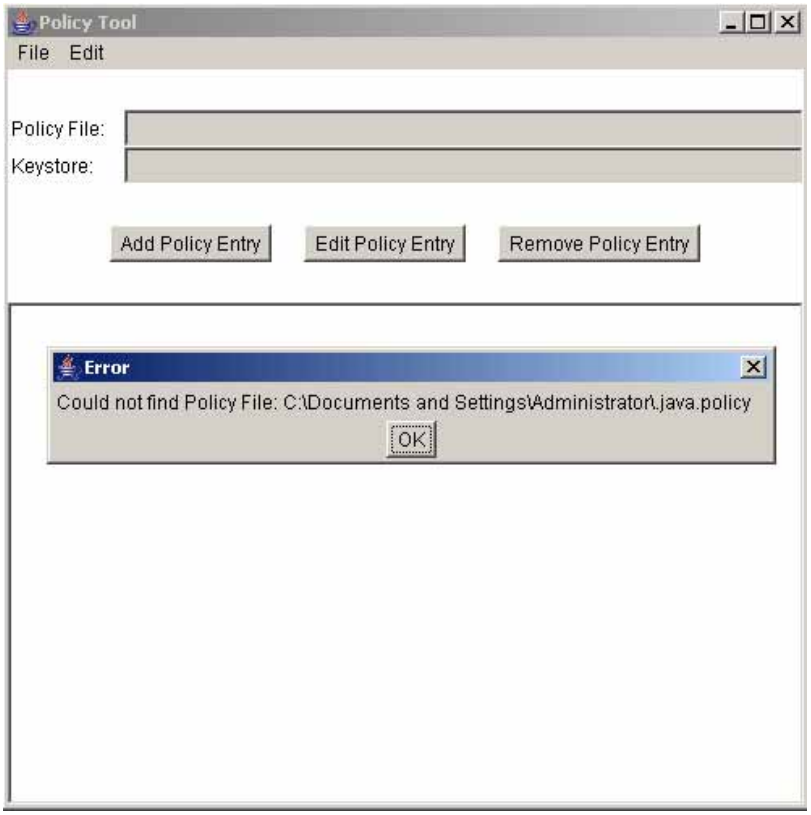


Рис. 39. Инструментальное окно политики

2. Щелкните кнопку OK диалогового окна Error. Затем, щелкните кнопку Add Policy Entry (Добавить политику) для добавления нового файла политики безопасности. Рис. 40 отображает диалоговое окно Policy Entry:

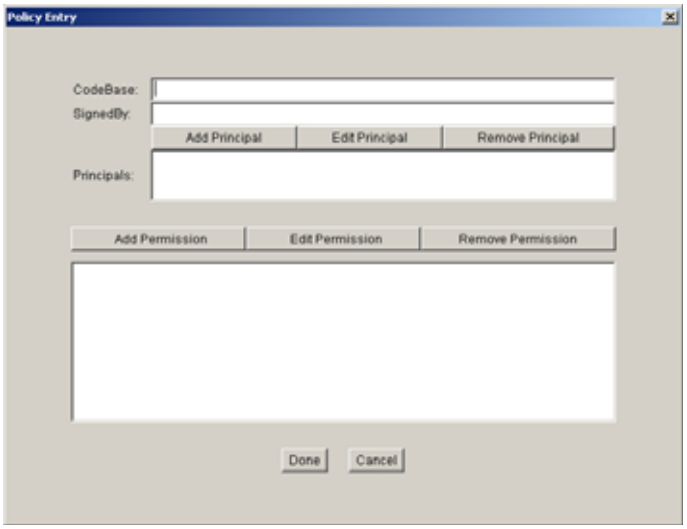


Рис. 40. Диалоговое окно ввода политики

3. Щелкните кнопку Add Permission (Добавить разрешение) чтобы выдать требуемые разрешения на получение доступа к ре-

сурсам сервера клиентам. Рис. 41 отображает пример диалогового окна Permissions (Разрешения):

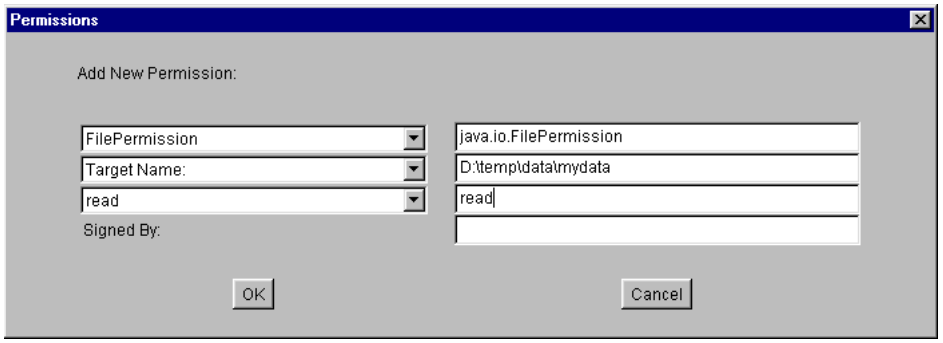


Рис. 41. Диалоговое окно разрешений

4. Выберите AllPermission из списка Permission диалогового окна Permissions
5. Щелкните кнопку OK, чтобы закрыть диалоговое окно Permissions.
6. Щелкните кнопку Done диалогового окна Policy Entry.
7. Выберите команду File->Save As в окне Policy Tool.
8. Сохраните файл политики безопасности как .java.policy в подкаталоге HOME операционной системы. Сообщение подтверждения отображается в окне сообщения Status. Рис. 42 отображает окно сообщения Status.

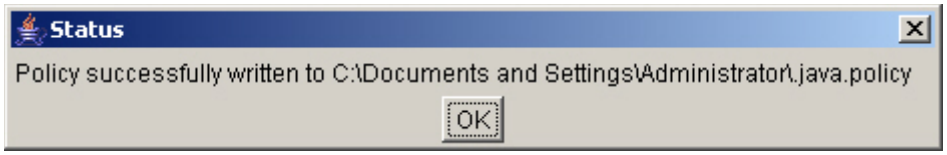


Рис. 42. Окно сообщения Status

9. Щелкните кнопку OK в окне сообщения Status.
10. Выберите команду File->Exit в окне Policy Tool, чтобы закрыть утилиту Java Policy Tool.

Дополнительная информация по управлению политикой безопасности может быть получена на сайте http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/policytool.html.

Это спецификация RMI для установки менеджера безопасности в соответствии с безопасностью сетевого приложения. Однако, приложение RMI может быть также выполнено без установки менеджера безопасности. В этом случае, код не будет подвергаться проверке безопасности перед его выполнением на JVM, и, таким образом, это может привести к сбою безопасности.

Если клиент представляет собой апплет, Вы не должны устанавливать менеджер безопасности, поскольку апплеты используют уже установленный менеджер безопасности на браузере клиента. Однако, если клиентом является приложение, необходимо установить менеджер безопасности как предложено выше.

Создание клиента RMI

Клиент использует объект стаб, чтобы получить доступ к удаленному объекту, который существует на сервере. Имя объекта сервера задается в методе lookup() класса the java.rmi.Naming, чтобы найти объект стаб. Вы можете использовать следующий оператор, чтобы получить доступ к объекту стаб, используя метод lookup():

```
Hello h = (Hello)Naming.lookup("rmi://127.0.0.1/server");
```

В предыдущем операторе метод the `lookup()` вызывает удаленную исключительную ситуацию. В результате метод `lookup()` следует закрыть внутри блока try-catch. Сервер представляет имя объекта сервер. Когда Вы запускаете это приложение на локальном компьютере, то Вы используете localhost вместо IP адреса сервера RMI. Вы можете использовать следующий код, чтобы создать RMI клиент для распределенного приложения:

```
import java.rmi.*;
public class HelloClient
{
      public static void main(String args[])
      {
            try
            {
                  /* Поиск удаленного объекта в регистре RMI */
Hello h = (Hello)Naming.lookup("rmi://127.0.0.1/server");
                  System.out.println("Client: Hello!");
                  System.out.println("Server: " + h.sayHello());
            }
            catch(Exception e)
            {
                  System.out.println("Error : "+e);
            }
      }
}
```

Исходные файлы Java распределенного приложения Hello компилируются стандартным компилятором Java javac, чтобы генерировать файлы class. После компиляции исходных файлов Java, необходимо сгенерировать стаб и скелет, которые общаются между клиентом и сервером.

Выполнение приложения RMI

Необходимо регистрировать объекты сервера в регистре RMI перед тем как запускается приложение. Клиент может вызывать объекты зарегистрированного сервера через сеть. Следующие шаги, чтобы выполнить RMI приложение:

1. Сгенерируйте стаб и скелет удаленного сервисного класса.
2. Запустите регистр RMI.

3. Запустите сервер RMI распределенного приложения.
4. Запустите клиент RMI распределенного приложения.

1. Генерация стаб и скелет

Компилятор RMI **rmic** компилирует класс удаленного сервиса, который реализуют удаленный интерфейс и генерирует стаб и скелет. Стаб дает возможность клиенту общаться с определенным удаленным объектом. Скелет представляет объект клиента, который размещается на удаленном хосте. Следующая команда, чтобы сгенерировать стаб и скелет:

```
rmic [опции] <ClassFile>
```

В предыдущей команде ClassFile представляет имя удаленного сервисного класса. Опции представляет параметры, которые обеспечивают дополнительные возможности RMI компилятора. Табл. 12 представляет различные опции RMI компилятора.

Таблица 12

Опции	***Описание***
`-bootclasspath <path>`	Отменяет размещение файла класса автоматической загрузки.
`-classpath <path>`	Отменяет путь к классу переменных среды по умолчанию.
`-d <directory>`	Задает имя подкаталога, где Вы хотите генерировать стаб и скелет. По умолчанию, стаб и скелет генерируются на текущем устройстве.
`-depend`	Компилирует все файлы, которые связаны с удаленным сервисным классом.
`-extdirs <path>`	Отменяет размещение установленных расширений.
`-g`	Генерирует номера линий и локальные переменные в форме таблицы.
`-keep`	Сохраняет файл '.java', который генерирует стаб и скелет.
`-nowarn`	Не отображает любые предупреждения, когда генерируется стаб и скелет.
`-vcompat`	Создает стаб и скелет, который совместим с ранними версиями протоколов RMI.
`-verbose`	Отображает сообщение, когда удаленный файл сервера скомпилирован rmic.
`-v <version>`	Создает стаб и скелет для заданной версии комплекта разработки Java (Java Development Kit) (JDK).

Команда генерации стаб и скелет для приложения RMI:

```
rmic HelloImpl
```

Два файла, HelloImpl_Stub.class и HelloImpl_Skel.class, генерируются в одном подкаталоге, в котором Вы сохраняете удаленный сервисный класс HelloImpl. Можно тестировать распределенное приложение на локальном компьютере, сохраняя все файлы в одном подкаталоге перед развертыванием его в сети. Чтобы выполнить приложение в сети, Вы должны создать два подкаталога, серверный подкаталог и клиентский подкаталог. В серверном подкаталоге сохраняются файлы:

- Hello.class
- HelloImpl.class
- HelloServer.class
- HelloImpl_Stub.class

В клиентском подкаталоге сохраняются файлы:

- Hello.class
- HelloImpl_Stub.class
- HelloClient.class

2. Запуск регистра RMI

Чтобы запустить регистр RMI на сервере, выполните команду start rmiregistry в командной строке. По умолчанию регистр запускается на порт 1099. Чтобы запустить регистр RMI на другой порт, необходимо задать номер порта в командной строке следующим образом:

```
start rmiregistry 1234
```

Если регистр запускается на отличный порт от 1099, Вы должны будете задать номер порта в строке URL, заданной в методах rebind() и the lookup() класса Naming. Следующая команда для запуска регистра RMI на порт по умолчанию:

```
start rmiregistry
```

Предыдущая команда открывает новое окно Command Prompt в котором выполняется rmiregistry.

Необходимо остановить и перезапустить сервис `rmiregistry`, когда Вы модифицируете удаленный интерфейс.

3. Выполнение сервера RMI

Необходимо запустить сервер, чтобы обслуживать запросы клиента. Следующая команда, чтобы запустить RMI сервер HelloServer:

```
java HelloServer
```

Рис. 43 отображает вывод приложения сервера RMI:

Рис. 43. Выполнение приложения сервера RMI

4. Выполнение клиента RMI

Следующая команда выполнени RMI клиента:

```
java HelloClient
```

Если Вы запускаете клиента с того же самого компьютера, задайте команду в отдельном командном окне.

Рис. 44 отображает вывод приложения клиента:

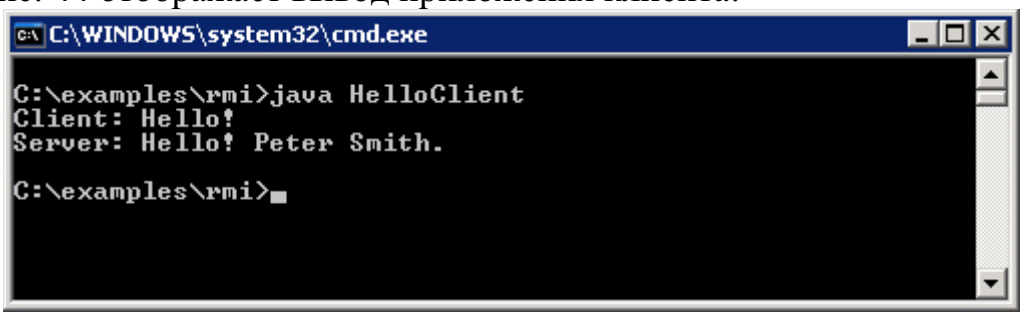


Рис. 44 Выполнение приложения RMI клиента

Клиент RMI вызывает метод sayHello(), чтобы отобразить сообщение, полученное от сервера RMI. Рис. 45 представляет общение между RMI-клиентом и RMI-сервером:

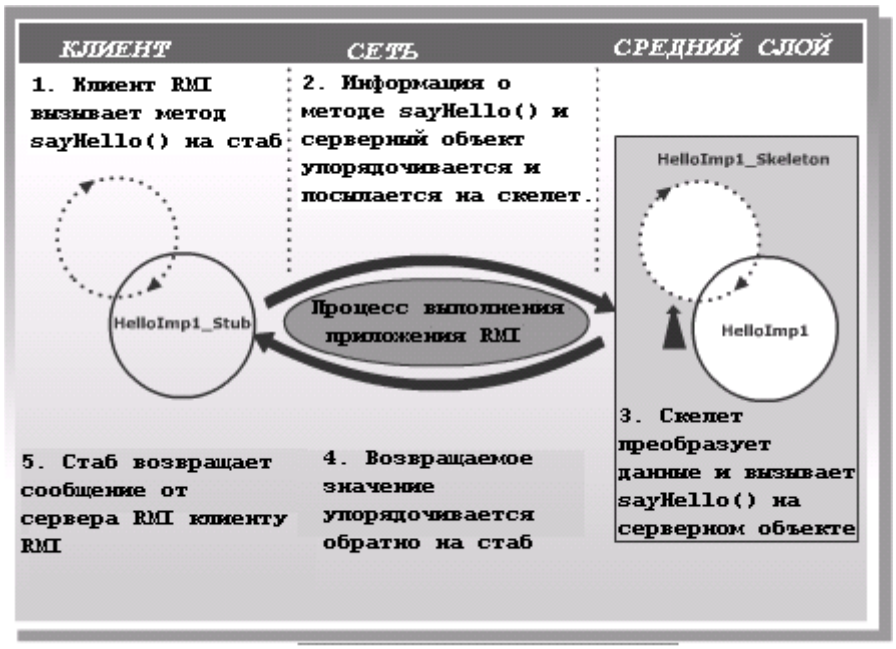


Рис. 45. Работа распределенного приложения, используя RMI

Передача параметров в RMI

Следующие два метода используются для передачи параметров при вызове удаленного метода в сети:

- Передача по значению
- Передача по ссылке

Передача по значению

Метод передачи по значению копирует исходные данные как параметры удаленному методу. В результате, изменения, сделанные удаленным методом отражаются только в копии исходных данных. Метод передачи по значению не позволяет удаленному методу вызывать изменения исходных данных.

Передача по ссылке

Метод передачи по ссылке передает исходные данные как параметры удаленному методу. В результате, любые изменения, сделанные удаленным методом в переменной переданного параметра отражаются в исходных значениях данных.

Введение в JNDI

Интерфейс имен и каталогов Java (Java Naming and Directory Interface) (JNDI) представляет собой API, который обеспечивает сервисы имен и каталогов для приложений Java. JNDI также обеспечивает информацию о приложениях, сервисах имен, сети и конечных пользователях.

Сервис имен дает Вам возможность связать имя с объектом. Иное приложение может использовать сервис имен, чтобы найти объект по имени этого объекта. Сервис имен отображает дружественные пользователю имена в объект, такие как адрес или идентификатор. Например, сервер доменных имен (Domain Name System) (DNS) отображает имя машины в уникальный адрес, такой как 195.208.175.45.

Сервис каталогов представляет собой расширение сервиса имен, который дает возможность связи имен с объектами. Сервис каталогов также дает возможность объектам получать атрибуты. В результате можно ссылаться на объект с помощью их имен или атрибутов.

Использование JNDI в RMI

Сервис провайдер регистра RMI дает возможность приложениям JNDI получать доступ к удаленным объектам. Сервис провайдер JNDI связывает контекст наименования с объектом, который зарегистрирован в регистре RMI. Когда содержание наименования связывается с зарегистрированным объектом, Вы можете получать доступ к удаленным объектам из любых мест. Клиент RMI не требует имя хоста или порта регистра, чтобы получить доступ, чтобы получить доступ к удаленному объекту, если Вы используете JNDI в RMI. Рис. 46 отображает как связать контекст наименования с объектом, используя JNDI.

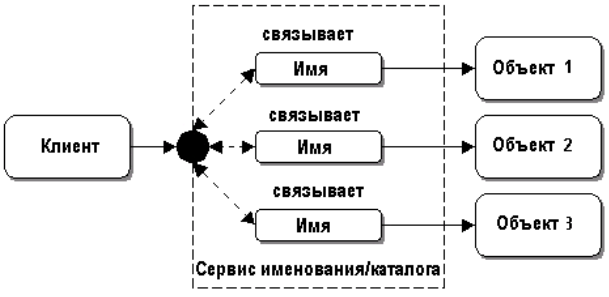


Рис. 46. Связывание имени с объектами

Провайдер регистра RMI использует следующие свойства среды, чтобы связать контекст наименования с зарегистрированным объектом в регистре RMI:

- java.naming.factory.initial: Задает имя класса для провайдера сервисов имени и каталога. Это свойство выбирает регистр провайдера сервиса как начального контекста для связывания.
- java.naming.provider.url: Задает размещение регистра в процессе запуска регистра RMI. Вы передаете местоположение как универсальный локатор информационного ресурса RMI (Uniform Rersource Locator) (URL).
- java.naming.factory.state: Возвращает список имен классов для класса state factory, который представляет состояние объекта.
- java.naming.factory.object: Возвращает список имен классов объекта класса factory, который преобразовывает объект чтения из регистра.

Архитектура JNDI

Архитектура JNDI содержит Java API приложения, менеджеры именования и каталога и интерфейс сервис провайдера (Service Provider Interface) (SPI). Приложения Java используют JNDI API, чтобы получить доступ к различным сервисам именования и каталога. SPI состоит из различных провайдеров сервиса, таких как облегченный протокол службы каталога (Lightweight Directory Access Protocol) (LDAP), стандартная архитектура брокера объектных запросов (Common Object Request Broker Architecture) (CORBA), RMI, DNS, сетевая информационная служба (Network Information Service) (NIS) и сервис каталога Novell (Novell Directory Services) (NDS). SPI дает возможность сервисам именования и каталога позволять приложению Java использовать JNDI API, чтобы получать доступ к их сервисам. Рис. 47 отображает архитектуру JNDI:

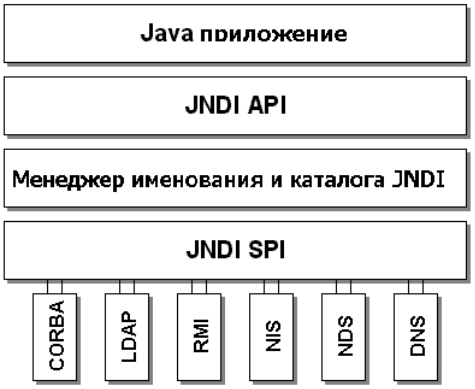


Рис. 47. Архитектура JNDI

JNDI API состоит из различных классов JNDI и провайдеров сервиса. JDK версии 4 или позже и сервер приложений Java EE включает следующих трех провайдеров сервиса для именования и каталога:

- CORBA: Дает возможность объектам общаться друг с другом не смотря на незнание какой язык и операционная система используются.
- LDAP: Дает Вам возможность получать доступ к информационным подкаталогам. LDAP содержит стандарты X.500 и поддерживает протокол управления передачей/интернет протокол (Transfer Control Protocol/Internet Protocol) (TCP/IP), который необходим для общения в сети. X.500 является стандартным протоколом, полученным Институтом инженеров по электротехнике и электронике (Institute of Electrical and Electronics Engineers) (IEEE), чтобы разрабатывать глобальные электронные каталоги. В результате любой в Интернет может полцчить доступ к каталогу.
- RMI: Дает возможность объектам Java общаться с другими объектами удаленно. RMI работает только с объектами, созданнми с использованием Java.
- NIS: Обеспечивает широкомасштабную информацию о пользователях, файлах, принтерах, машин и сетей. Вы встречаетесь с NIS, когда работая с системами, используете операционную систему Solaris. Существуют, однако, и другие системы, такие как Linux и другие операционные системы Unix, которые поддерживают NIS.
- NDS: Обеспечивает информацию о сетевых сервисах, таких как принтеры и файлы. NDS в основном находится в средах, в которых Novell обеспечивает основное сетевое программное обеспечение.
- DNS: Идентифицирует хосты в сети, выполняя трансляцию между именем хоста и адресом интернет.

Вопросы для самопроверки

1. Дайте определение распределенным приложениям.
2. На каком слое реализуется RMI в трехслойной модели архитектурного построения.
3. Что по существу позволяет RMI клиенту одной JVM по отношению к другой JVM.
4. Из каких 3-х уровней состоит архитектура RMI?
5. Из каких четырех пакетов состоит RMI? Дайте характеристику

этих пакетов.

6. Что представляет собой и где размещается регистр RMI?
7. Где объявляются методы, которые могут быть вызваны удаленно клиентом.
8. Какие операции должен выполнить удаленный сервисный класс для сервера RMI?
9. Какой компонент обеспечивает для Java-приложения менеджер безопасности.
10. Какие шаги необходимо выполнить для создания и выполнения приложения RMI?

Для освоения материала предлагается выполнить лабораторную работу № 3, в разделе Лабораторный практикум.

Технология и архитектура JavaEE

В предыдущих разделах мы рассмотрели базовые средства Java для создания серверных приложений, которые обеспечивют клиент/серверное взаимодействие-современную архитектуру распределенных приложений. Для сокращения стоимости и эффективности проектирования и разработки распределенных систем фирма Sun Microsystems разработала компонентную архитектуру Java 2 Enterprise Edition (J2EE). Последняя спецификация платформы Sun Microsystems на основе JDK 5.0 носит название JavaEE и предлагает многоуровневую распределенную модель приложений, многократно используемые компоненты, унифицированную модель безопасности, гибкое управление транзакциями, поддержку WEB-сервисов посредством интегрированного обмена данными на основе языка Extensible Markup Language (XML), открытых стандартов и протоколов.

Такое платформенно – независимое решение не связано с каким-либо продуктом и application program interface (API) одного поставщика, а является открытой спецификацией, которую могут реализовать различные разработчики, обеспечивая наилучший уровень удовлетворения технологических и бизнес требований. При этом JavaEE представляет архитектуру для разработки мобильных приложений, которые могут выполняться на различных операционных системах, таких как Windows, UNIX и MacOS. Разработанное JavaEE приложение может быть выполнено под управлением любого совместимого со спецификаций JavaEE сервера без каких-либо изменений.

Приложения JavaEE состоят из компонентов, которые представляют собой отдельные функциональные единицы программного обес-

печения, которые собираются в приложение JavaEE вместе с классами и файлами, и, которые взаимодействуют с другими компонентами. Спецификация определяет следующие JavaEE компоненты:

- Клиентские приложения и апплеты, которые выполняются на клиентской части приложения.
- Java Servlet Java Server Pages выполняются на сервере
- Компоненты Enterprise JavaBeans (EJB) являются бизнес компонентами, которые выполнятся на сервере.

JavaEE компоненты написаны на языке Java и компилируются также как и любые программы на языке Java. Отличием является то, что компоненты JavaEE объединяются в архитектуру стандартного пакета (архивы EAR,WAR, JAR) приложения JavaEE, проверяется на соответствие спецификации JavaEE и разворачивается на сервере JavaEE, где выполняется и контролируется.

Сервер JavaEE обеспечивает определенные сервисы в форме контейнеров (Container) для каждого типа компонентов. Поскольку нет необходимости разрабатывать эти сервисы самостоятельно, вы концентрируетесь исключительно на решении бизнес проблем. Контейнер представляет собой интерфейс между компонентом и определенной функциональностью низкого уровня, поддерживающей компоненты. Перед выполнением любой компонент распределенного приложения должен быть собран в модуль JavaEE и размещен в соответствующий контейнер.

JavaEE включает компоненты, которые могут быть использованы для разработки приложений как уровня представления так и бизнес логики. Стандартная спецификация определяет API для управления бизнес транзакциями, использования элементов безопасности, инфраструктуру инструментальных средств для поддержки среды выполнения приложения, а также средств для внутренней и внешней интеграции. Процесс сборки приложений вовлекает задание установок контейнера как для каждого из компонентов приложении JavaEE так и для самого приложения. Установки контейнера определяют поддержку, обеспечиваемую сервером JavaEE, включающим такие сервисы как безопасность, управление транзакциями, просмотр Java Naming and Directory Interface (JNDI) и удаленную связность. На рис. 48 представлена архитектура многоуровневого приложения, на котором выделены уровни презентации, реализации бизнес логики и база данных отделены.

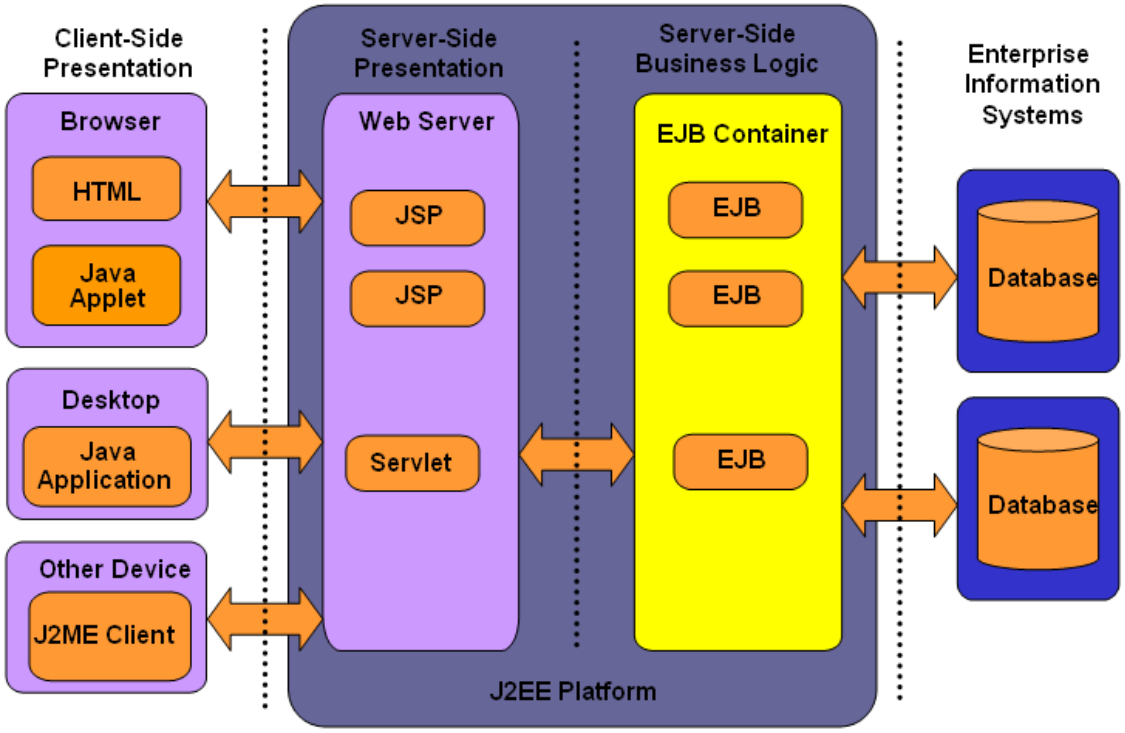


Рис. 48 Архитектура многоуровневого приложения JavaEE
Ниже приводится список основных технологий JavaEE.

Приложения и транзакции

- Components – Servlet, JavaServer Pages (JSP), Java Server Faces (JSF), и EJB являются компонентами, выполняемыми на стороне сервера и использующими для определения логики презентации и бизнес логики.
- HTTP – Клиентское API обеспечивается пакетом java.net package, а серверное API HTTP определяется с использованием servlet и JSP.
- HTTPS – Тоже протокол HTTP выполняется поверх протокола SSL теми же клиентскими и серверными API.
- JavaMail – Почтовый API обеспечивает интерфейс уровня приложений для компонентов приложений для передачи электронной почты через Интернет.
- Java Transaction (JTA) API – JTA API предназначено для разграничения границ транзакций между контейнером и приложением при выполнении распределенного приложения с транзакциями.
- Java Naming and Directory Interface (JNDI) JNDI API обеспечивает интнрфейс уровня приложения для доступа к сервисам имен и подкаталогов, а также к интерфейсу провайдера сервиса.

- JavaBeans Activation Framework (JAF) JAF обеспечивает API для обработки данных различных типов Multipurpose Internet Mail Extension (MIME), созданных в различных форматах и на разных языках.

Сервисы безопасности

- Java Authentication and Authorization Service (JAAS) – Контекст входа для аутентификации и авторизации обслуженного потребителя Login context for authenticating and authorizing the serviced requester.
- Java Authorization Service Provider Contract for Container (JACC) Устанавливает связь между сервером приложений Java EE и провайдером сервиса авторизации.
- Java Secure Socket Extension (JSSE) API для Secure Socket Layer, который обеспечивает безопасную сессию для конфиденциальности данных, целостности данных и сервера аутентификации.
- Java Cryptography Architecture (JCA) Базовый API для доступа и разработки криптографической функциональности.
- Java Crypto Services (JCE) Криптографическое API с расширенными функциями для поддержки множества провайдеров сервера криптографии.
- CertPath или Certification Path – API для создания, построения и поверки достоверности путей цифровых сертификатов.
- Java Generic Security Services Application Program Interface (JGSS) – API для унифицированного доступа к различным механизмам безопасности, включая Kerberos, который является строительным блоком для однократного предъявление пароля (single sign-on) и кодирования данных.

Интеграция и интероперабельность

- Java Message Service (JMS) – JMS обеспечивает надежный обмен сообщениями для сервисов по принципам точка-точка публикация-подписка.
- Remote Method Invocation over the Internet Inter-ORB Protocol (RMIIIOP) – API позволяет обрабатывать удаленные вызовы Java , используя RMI поверх IIOP, который может получить доступ к объектам CORBA или сервисам непосредственно из приложения Java RMI.

- Java Interface Description Language (IDL) – Java IDL позволяет приложениям Java EE работать как клиенту CORBA для вызова внешних объектов CORBA, используя протокол IIOP.
- JDBC API – JDBC – API обеспечивает связность с системой управления базы данных, которая включает соединение (connection), пул соединений и распределенные сервисы баз данных.
- Архитектура Java EE Connector – Архитектура Connector представляет собой интерфейс провайдера сервиса, который позволяет ресурсам, соединенным с Enterprise Information Systems (EIS) или действующим системам соединяться с любыми компонентами сервисов Java
- Web Services (WS) – Средство включает поддержку API для синхронных Web-сервисов (Java API for XML-based RPC или JAX-RPC), асинхронных Web-сервисов (SOAP с присоединенными API для Java или SAAJ), и доступ к сервисам регистра XML (Java API for XML Registries или JAXR). JAXP обеспечивает стандартный способ анализа документа XML и их изменения, используя таблицу стилей (stylesheet). Java EE 5.0 добавляет более простую и расширенную поддержку для Web сервисов введением JAX-WS 2.0 (наследник JAX-RPC) и JAXB 2.0.

Управление

- Java Management Extensions (JMX) JMX API перехватывает события приложений и исключительные ситуации на уровне системы управления приложения и диагностики. На рис. 49 представлена архитектура JavaEE

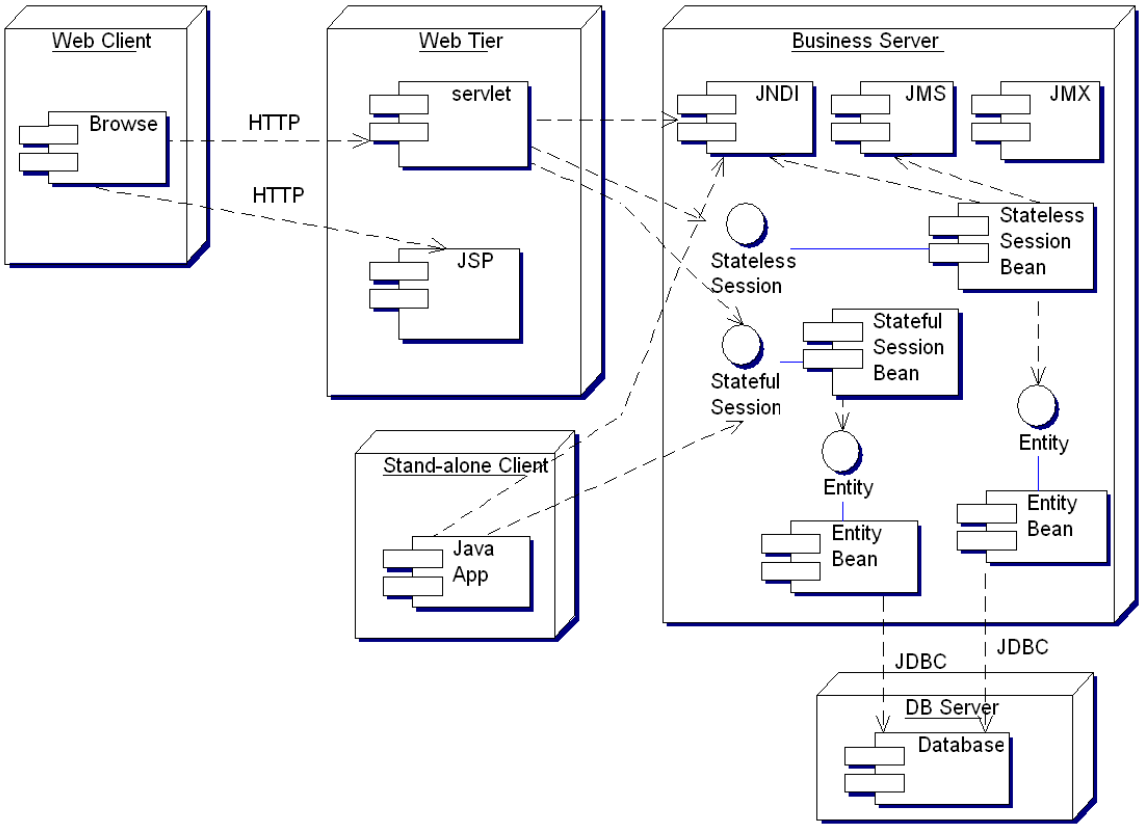


Рис. 49. Архитектура JavaEE

JavaEE является открытой спецификацией, которая реализуется как в составе уже упомяеутых коммерческих разработок Oracle, Web Sphere (IBM), Web Logic, так и большим количеством открытых систем, таких как Apache, Jboss, Jetty и ряда других. В пособии используется популярная серверная реализация спецификации JavaEE фирмы Red Hat, которая назваетя Jboss (www.jboss.org) и свободно распространяемая интегрировнная среда разработки (Integrated Development Environment) Eclipse (www.eclipse.org).

Очевидно, что в рамках предлагаемого пособия не представляется возможным рассмотреть детально, а тем паче апробывать все упомянутые компоненты архитектуры JavaEE. Полную информацию можно получить из списка литературы, приведенного в конце пособия и на сайте www.sun.com, однако, основные технологии JavaEE, необходимые для использования при создания распределенных приложений, такие как JDBC, программирование сокетов, RMI, JSP, Servlet, Web services будут представлены в последующем материале.

Введение в сервлеты Java

Понятие сервлета

Сервлеты – это серверные программы, написанные на Java. В отличие от CGI-скриптов код инициализации сервлета выполняется только один раз. Кроме того, обрабаека каждого клиентского запроса выполняется в отдельном потоке на сервере. Это предотвращает создание лишних процессов, увеличивая, таким образом, производительность сервера. При этом сервлеты наследуют все свойства языка программирования Java. Например, подобно всем стандартным классам Java, сервлет не зависит от платформы и может использоваться в различных операционных системах. Также сервлет может пользоваться большими возможностями Java, такими как работа в сети, многопоточность, JDBC, локализация и RMI.

Технология Java Servlet

Сервлеты позволяют расширить функциональность серверного приложения для формирования динамического содержимого в приложении. Кроме всех наследуемых свойств Java, сервлеты имеют следующие черты:

- *Безопасность*: Веб-контейнер обеспечивает среду выполнения для сервлета. Сервлеты наследуют параметры безопасности, предоставляемые веб-контейнером. Это позволяет разработчикам сосредоточиться на функциональности сервлета и передать проблемы безопасности веб-контейнеру для обработки.
- *Управление сеансом*: Это механизм отслеживания состояния пользователя при многократных запросах. Сеанс сохраняет идентичность и состояние клиента при многократных запросах.
- *Устойчивость объекта*: Сервлеты помогают увеличить производительность сервера, предотвращая частый доступ к диску. Например, если клиент заходит на сайт банка для проверки балланса или получения ссуды, его номер счета должен проверяться на каждой этапе в базе данных. Вместо каждой проверки номера счета в базе данных, сервлеты сохраняют номер счета в памяти до тех пор, пока пользователь не покинет веб-сайт.

Чтобы получить доступ к сервлету, сначала необходимо откомпилировать сервлет, а затем развернуть файл класса сервлета на a Java-совместимом сервере приложений, таком, например, JBOSS Application

Server. Сервер приложений JavaEE предоставляет веб-контейнер для управления различными компонентами веб-приложения, в том числе, HTML-страница и сервлеты. Веб-контейнер обеспечивает среду исполнения для запуска сервлета. Веб-контейнеры предоставляют следующие сервисы, требуемые веб-приложениям:

- Управление различными стадиями времени жизни сервлета, такими как инициализация экземпляра сервлета, обработка клиентского запроса и удаление экземпляра сервлета.
- Определение ограничений безопасности, которые предписывают получение доступа к развернутым сервлетам авторизованным пользователям.
- Управление транзакциями, когда сервлету нужно записать и прочитать данные из базы данных.
- Создание и удаление экземпляров сервлетов из пула экземпляров для обслуживания многократных запросов.

Работа сервлетов

Когда клиент отправляет запрос серверу приложений JavaEE для конкретного сервлета, сервер приложений JavaEE передает запрос веб-контейнеру, который проверяет, существует ли экземпляр требуемого сервлета. Если экземпляр сервлета существует, то веб-контейнер передает запрос сервлету, который обрабатывает запрос клиента и отправляет назад ответ.

Если экземпляр сервлета не существует, веб-контейнер находит и загружает класс сервлета. Затем веб-контейнер создает экземпляр сервлета и инициализирует его. После инициализации экземпляр сервлета начинает обработку запроса. Веб-контейнер передает ответ, сгенерированный сервлетом, клиенту. Рис. 50 показывает взаимодействие между сервером приложений JavaEE и веб-контейнером:

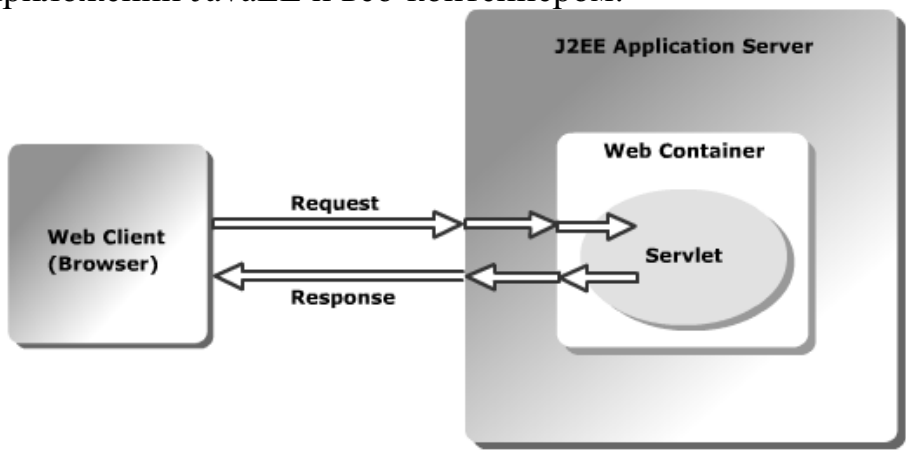


Рис. 50 Взаимодействие между сервером приложений и веб-контейнером при обработке клиентского запроса

Рис. 51 показывает этапы, выполняемые веб-контейнером при первом получении запроса от клиента.

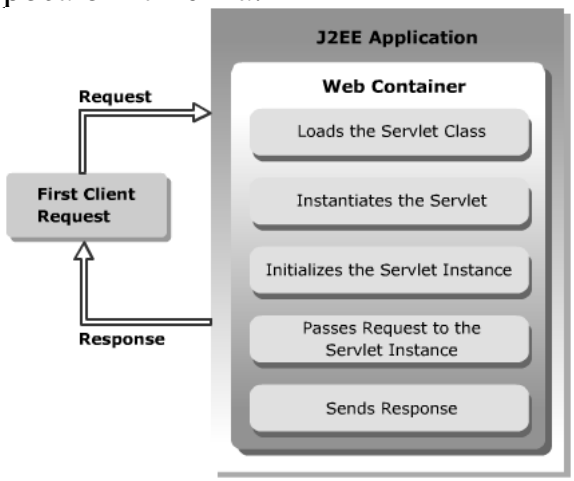


Рис. 51. Этапы, выполняемые веб-контейнером

Иерархия класов сервлетов и методы жизненного цикла

Веб-контейнер управляет сервлетом, вызывая различные методы жизненного цикла. Эти методы определены в Servlet API. Это коллекция классов и интерфейсов, которую можно использовать для разработки сервлета. Эти классы и интерфейсы находятся в пакетах javax.servlet и javax.servlet.http.

Иерархия класса Servlet

Интерфейс Servlet является корневым интерфейсом иерархии сервлетов. Всем сервлетам необходимо явно или неявно реализовывать интерфейс Servlet. Класс GenericServlet из Servlet API реализует интерфейс Servlet. Кроме интерфейса Servlet, класс GenericServlet реализует интерфейс ServletConfig из Servlet API и интерфейс Serializable из стандартного пакета java.io. Объект интерфейса ServletConfig используется веб-контейнером для передачи конфигурационной информации сервлету при его инициализации.

Чтобы разработать сервлет, который взаимодействует по протколу HTTP, необходимо расширить класс HttpServlet в сервлете. Класс HttpServlet расширяет класс GenericServlet и предоставляет встроенную функциональность HTTP. Например, HttpServlet предоставляет методы, которые позволяют сервлету обрабатывать запрос клиента, приходящий с помощью определенного метода HTTP. Рис. 52 показывает общий план иерархии интерфейсов и классов в пакетах javax.servlet и javax.servlet.http:

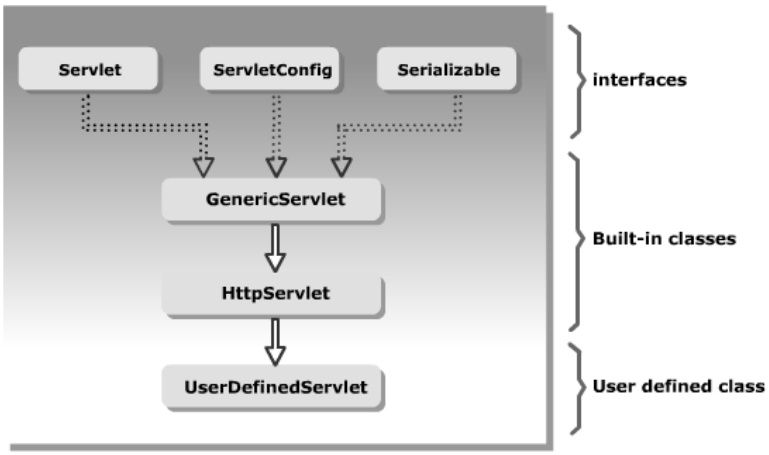


Рис. 52. Общий план иерархии сервлетов

Интерфейс javax.servlet.Servlet

Рис. 53 представляет иерархию объектов класса servlet.

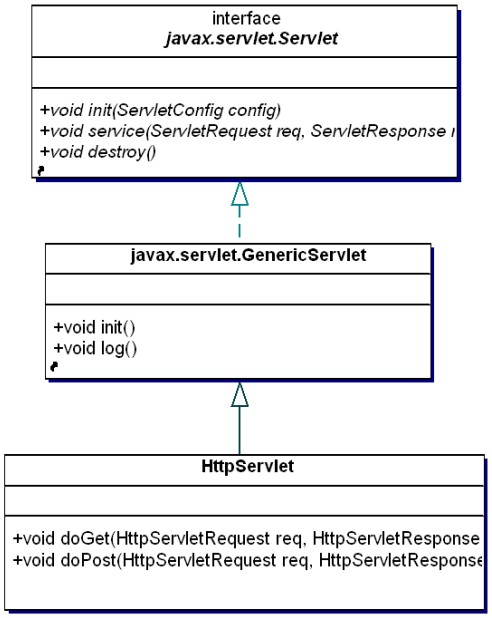


Рис. 53. Иерархия объектов класса servlet

Интерфейс Servlet пакета javax.servlet определяет методы, которые веб-контейнер вызывает для управления жизненным циклом сервлета. Табл. 13 представляет методы интерфейса javax.servlet.Servlet.

Таблица 13

Метод	***Описание***
`public void destroy()`	Веб-контейнер вызывает метод destroy()перед уничтожением экземпляра сервлета.
`public ServletConfig getServletConfig()`	Возвращает объект ServletConfig, который содержит данные о конфигурации, такие как параметры инициализации, чтобы инициализи-

	ровать сервлет.
`public String getServletInfo()`	Возвращает строку, которая содержит информацию о сервлете, такую как автор, версия и авторское право.
`public void init(ServletConfig config) throws ServletException`	Веб-контейнер вызывает этот метод после создания экземпляра сервлета.

Интерфейс javax.servlet.ServletConfig

Интерфейс javax.servlet.ServletConfig реализуется веб-контейнером, чтобы передавать данные о конфигурации сервлету во время инициализации сервлета. Сервлет инициализируется передачей объекта ServletConfig его методу init() веб-контейнером. Объект ServletConfig содержит данные для инициализации и предоставляет доступ к объекту ServletContext.

Параметры инициализации – это пары имен и значений, которые можно использовать для передачи информации сервлету. Например, можно указать URL JDBC как параметр инициализации сервлета. При инициализации сервлет может использовать значение URL, чтобы получить соединение с базой данных. Объект интерфейса ServletContext позволяет сервлету взаимодействовать с веб-контейнером, владеющим этим сервлетом. Табл. 14 представляет ряд методов интерфейса javax.servlet.ServletConfig:

Таблица 14

Метод	***Описание***
`public String getInitParameter(String param)`	Возвращает строку, содержащую значение указанных параметров инициализации или null, если параметр не существует.
`public Enumeration getInitParameterNames()`	Возвращает имена всех параметров инициализации как объект Enumeration, содержащий объекты String. Если параметры инициализации не определены, возвращается пустой Enumeration.
`public ServletContext getServletContext()`	Возвращает объект ServletContext для сервлета, который позволяет взаимодействие с веб-контейнером.

Методы жизненного цикла сервлета

Интерфейс javax.servlet.Servlet определяет методы жизненного цикла сервлета, такие как init(), service() и destroy(). Веб-контейнер вызывает методы сервлета init(), service() и destroy() во время его жизни. Ниже представлена последовательность, в которой веб-контейнер вызывает методы времени жизни сервлета:

1. Веб-контейнер загружает класс сервлета и создает один или более экземпляров класса сервлета.
2. Веб-контейнер вызывает метод init() экземпляра сервлета во время инициализации сервлета. Метод init() вызывается только один раз в жизненном цикле сервлета.
3. Веб-контейнер вызывает метод service(), чтобы разрешить сервлету обработку запроса клиента.
4. Метод service() обрабатывает запрос и возвращает ответ веб-контейнеру.
5. Затем сервлет ожидает, чтобы получить и обработать следующие запросы, как указано в пунктах 3 и 4.
6. Веб-контейнер вызывает метод destroy() перед удалением экземпляра сервлета. Метод destroy() также вызывается только один раз во время жизни сервлета.

Метод init()

Метод init() вызывается во время инициализации жизненного цикла сервлета. Веб-контейнер сначала преобразует запрашиваемый URL в соответствующий сервлет, имеющийся в веб-контейнере, а затем назначает этот сервлет для выполнения. Затем веб-контейнер создает объект интерфейса ServletConfig, который содержит данные конфигурации при запуске, такие как параметры инициализации сервлета. После чего веб-контейнер вызывает метод init() сервлета и передает ему объект ServletConfig.

Метод init() генерирует исключение ServletException, если веб-контейнер не может инициализировать ресурсы сервлета. Инициализация сервлета завершается до того, как начнут приниматься любые клиентские запросы. Следующий фрагмент кода демонстрирует метод init():

```
public void init (ServletConfig config) throws ServletException
```

Следующий фрагмент кода представляет сигнатуру метода init() сервлета Java для его инициализации:

```
public class ServletLifeCycle extends HttpServlet
{
```

```
static int count;

public void init (ServletConfig config) throws ServletException
{
            count=0;
}
}
```

Метод service()

Метод service() обрабатывает запросы от клиента. Каждый раз, когда веб-контейнер получает клиентский запрос, он вызывает метод service(). Метод service() вызывается только после того, как завершится инициализация сервлета. При вызове метода service() веб-контейнер передает объект интерфейса ServletRequest и объект интерфейса ServletResponse. Объект ServletRequest содержит информацию о запросе, сделанном клиентом. Объект ServletResponse содержит информацию, возвращаемую сервлетом клиенту. Следующий фрагмент кода демонстрирует метод service():

```
public void service(ServletRequest req, ServletResponse res) throws Ser-
vletException,IOException
```

Метод service() генерирует исключение ServletException, когда возникает исключительная ситуация, которая препятствует нормальной работе сервлета. Метод service() генерирует исключение IOException, когда возникает исключительная ситуация ввода или вывода.

Метод service() отправляет клиентский запрос одному из методов обработки запроса интерфейса HttpServlet, таким как doGet(), doPost(), doHead() или doPut(). Методы обработки запроса принимают объекты HttpServletRequest и HttpServletResponse как параметры от метода service().

Метод `service()` не переопределяется в `HttpServlet`, так как веб-контейнер автоматически вызывает метод `service()`. Функциональность HTTP сервлетов записывается в методах `doGet()` или `doPost()`.

Метод doGet()

Метод doGet() обрабатывает клиентский запрос, используя GET-метод HTTP. GET является типом метода запроса HTTP, который в основном используется для извлечения статических ресурсов. Когда вы вводите адрес URL в строке браузера для просмотра статической веб-страницы, браузер использует метод GET для выполнения запроса. Аналогично, когда вы щелкаете по гиперссылке на веб-странице, чтобы получить доступ к некоторому ресурсу, ваш запрос отправляется с использованием метода GET. Также, используя метод GET, можно от-

правлять данные, введенные пользователем в форму HTML. Данные, отправленные с использованием GET-метода, добавляются как строка запроса к URL. Тип метода HTTP указывается в форме HTML, используя атрибут METHOD тега FORM.

Для обработки клиентских запросов, которые получены с использованием метода GET, необходимо переопределить метод doGet() в вашем классе сервлета. В методе doGet() вы можете извлечь данные от клиента из объекта HttpServletRequest. Вы можете использовать объект HttpServletResponse, чтобы отправить ответ клиенту. Следующий фрагмент программы демонстрирует doGet():

```
protected void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException
```

Метод doPost()

Метод doPost() обрабатывает запросы в сервлете, которые отправлены клиентом, используя метод POST протокола HTTP. Например, если клиент вводит регистрационные данные в форму HTML, данные могут быть отправлены с использованием метода POST. В отличие от метода GET, запрос POST отправляет данные как часть тела запроса HTTP. В результате отправленные данные не видны как часть URL.

Для обработки запросов в сервлете, которые отправляются с использованием метода POST, необходимо переопределить doPost(). В методе doPost() можно обработать запрос и отправить ответ клиенту. Следующий фрагмент кода демонстрирует метод doPost():

```
protected void doPost(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException
```

Метод doHead()

Метод doHead() обрабатывает запросы, отправленные с использованием метода HEAD протокола HTTP. Подобно методу GET, метод HEAD также отправляет запросы серверу. Единственное отличие между методами GET и HEAD заключается в том, что метод HEAD возвращает заголовок ответа, который содержит элементы, такие как Content-Type (Тип содержимого), Content-Length (Длина содержимого) и Last-Modified (Дата последнего изменения). Метод HEAD используется для определения, существует ли запрашиваемый ресурс. Он также используется для получения информации о ресурсе, такой как тип ресурса. Эта информация требуется перед повторным запросом содержимого ресурса. Также можно использовать метод HEAD для определения времени,

когда ресурс был последний раз изменен. Это помогает определить, можно ли использовать ресурс из кэша, или необходимо обновить кэш.

Чтобы обработать запрос, отправленный с помощью метода HEAD, следует переопределить метод doHead() в сервлете. Следующий фрагмент кода демонстрирует использование метода doHead():

```
protected void doHead(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException
```

Метод doPut()

Метод doPut() обрабатывает запросы, отправленные с использованием метода PUT протокола HTTP. Метод PUT позволяет клиенту сохранить информацию на сервере. Например, можно использовать метод PUT для отправки файла с изображением на сервер. Следующий фрагмент кода демонстрирует метод doPut():

```
protected void doPut(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException
```

Метод destroy()

Метод destroy() обозначает конец жизненного цикла сервлета. Веб-контейнер вызывает метод destroy() перед удалением экземпляра сервлета в следующих случаях:

- Период времени, указанный для сервлета, истек. Период времени сервлета – это период, в течение которого сервлет сохраняется в активном состоянии веб-контейнером для обслуживания клиентских запросов.
- Веб-контейнеру нужно выгрузить экземпляры сервлетов для сохранения памяти.
- Веб-контейнер заканчивает работу.

В методе destroy() можно написать код для освобождения ресурсов, занимаемых сервлетом. Например, если сервлет имеет открытое соединение с базой данных, вы можете закрыть соединение с базой данных в методе destroy() сервлета. Метод destroy() также используется для сохранения любой предназначенной для длительного использования информации перед тем, как экземпляр сервлета будет удален. Следующий фрагмент кода демонстрирует метод destroy():

```
public void destroy();
```

Создание сервлета

Для создания сервлета, необходимо выполнить следующие действия:

1. Напишите код сервлета.
2. Откомпилируйте и упакуйте сервлет.
3. Разверните сервлет как приложение JavaEE.
4. Вызовите сервлет из браузера.

Программирование сервлета

Чтобы написать код сервлета, необходимо расширить класс интерфейса HttpServlet. При разработке сервлета нужно включить функциональность для чтения клиентского запроса и отправки ответа.

Чтение и обработка клиентского запроса

После объявления и инициализации различных переменных сервлета, необходимо описать функциональность сервлета. Для этого переопределите методы doGet() и doPost() класса HttpServlet. Методы doGet() и doPost() получают запросы HTTP, используя интерфейс HttpServletRequest. Объект HttpServletRequest содержит информацию о запросе, отправленном клиентом с помощью запроса HTTP. Можно получить значения параметра запроса, вызвав метод getParameter() интерфейса HttpServletRequest. Метод getParameter() интерфейса HttpServletRequest принимает имя параметра запроса как строковое значение и возвращает строку, которая содержит соответствующее значение параметра. Следующий фрагмент программы демонстрирует метод getParameter():

```
String getParameter(String arg);
```

Следующий фрагмент кода показывает, как можно получить имя пользователя, отправленное браузером:

```
String user=req.getParameter("UserName");
```

Отправка ответа клиенту

HTTP-сервлеты для отправки ответа клиенту используют объект интерфейса HttpServletResponse. Чтобы отправить символьные данные клиенту, нужно получить объект java.io.PrintWriter. Можно использовать метод getWriter() интерфейса HttpServletResponse, чтобы получить объект PrintWriter. Следующий фрагмент кода демонстрирует пример использования метода getWriter() для получения объекта PrintWriter:

```
PrintWriter out = response.getWriter();/* Здесь response является
объектом HttpServletResponse. */
```

Можно указать тип содержимого ответа, используя метод setContentType() интерфейса HttpServletResponse как это делается в следующем фрагменте кода сервлета:

```
res.setContentType("text/html");
```

Отправить данные клиенту можно, используя метод println() класса PrintWriter как показывает следующий фрагмент кода:

```
out.println("Welcome "+ user+".");
```

Компиляция и упаковка сервлета

Для формирования файла класса сервлета необходимо откомпилировать подготовленный код сервлета. До компиляции сервлета нужно включить файл j2ee.jar в путь к классам. Вы можете найти файл j2ee.jar в каталоге lib каталога, в котором установлен J2EE 1.4 Sun Application Server. Например, если J2EE 1.4 Application Server установлен на диске C, можно использовать следующую команду, чтобы установить путь к классам:

```
C:\> set classpath= %classpath %;C:\Sun\Appserver\lib\j2ee.jar;
```

При использовании другого сервера приложений, например Jboss от RedHat или WebLogic от компании BEA Systems, следует указывать подключаемые классы соответствующим образом.

После компиляции сервлета его необходимо упаковать с помощью создания дескриптора развертывания. Дескриптор развертывания – это файл XML с именем `web.xml`, который содержит конфигурационную информацию о веб-приложении, для которому он создается. Файл дескриптора развертывания `web.xml` может управлять регистрацией сервлета, соответствиями URL, файлами приветствия, MIME-типами, ограничениями доступа к страницам и работой сервлетов в распределенных средах. Файл дескриптора развертывания не зависит от сервера и упрощает процесс развертывания.

Дескриптор развертывания можно создать с помощью XML-редактора. Код ниже демонстрирует пример дескриптора развертывания:

```
      <?xml version="1.0" encoding="UTF-8" ?>
      - <web-app version="2.4"
xmlns=http://java.sun.com/xml/ns/j2ee
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
      <display-name>SampleApp</display-name>
      <servlet>
      <display-name>SampleServlet</display-name>
      <servlet-name>SampleServlet</servlet-name>
      <servlet-class>SampleServlet</servlet-class>
      </servlet>
      </web-app>
```

В представленном коде первая строка дескриптора развертывания является *прологом*. Пролог устанавливает версию XML и кодировку,

которая используется для кодирования данных. В данном примере используется кодировка UTF-8.

В файле `web.xml` используются следующие атрибуты:

`version`: Атрибут задает версию схемы. Схемы XML используются для подтверждения действительности XML-документа.

`xmlns:` Атрибут задает пространство имен схемы дескриптора развертывания. Все схемы дескрипторов развертывания J2EE используют пространство имен `http://java.sun.com/xml/ns/j2ee`. Пространство имен позволяет схеме или множеству схем быть однозначно идентифицируемыми.

`xsi:schemaLocation:` Этот атрибут задает местоположение схемы. После указания данных о схеме, необходимо внести данные о развертываемом сервлете.

Тег `<displayname>` внутри тега `<web-app>` задает имя веб-компонента, который необходимо развернуть. Тег `<servlet>` содержит информацию о сервлете. Он содержит следующие теги:

`<display-name>:` Отображаемое имя сервлета

`<servlet-name>:` Предназначенное для доступа имя сервлета

`<servlet-class>:` Имя класса сервлета

После создания дескриптора развертывания можно упаковать сервлет. J2EE определяет стандартную структуру упаковки сервлета в приложение J2EE, для того чтобы сделать его переносимым между различными серверами приложений, что позволяет серверам приложений легко размещать и загружать файлы приложений из стандартной структуры каталогов. Чтобы создать упакованную структуру веб-приложения, необходимо создать следующие каталоги:

Корневой каталог: Корневой каталог содержит статические ресурсы, такие как, файлы HTML, файлы JSP и файлы изображений. Например, в корневом каталоге можно разместить файл `LoginPage.html`.

Каталог WEB-INF внутри корневого каталога: Каталог WEB-INF содержит файл дескриптора развертывания приложения `web.xml`, который хранит различные конфигурации веб-приложения.

Каталог classes: Каталог classes размещается внутри каталога WEB-INF и содержит файлы классов приложения. Например, можно разместить файл `SampleServlet.class` сервлета SampleServlet в каталоге classes.

Каталог lib: Каталог lib размещается также внутри каталога WEB-INF и содержит Java Archive files (JAR – Архивные файлы Java) библиотек, которые требуются компонентами приложения.

На рис. 54 представлена стандартная структура упаковки веб-приложения Java.

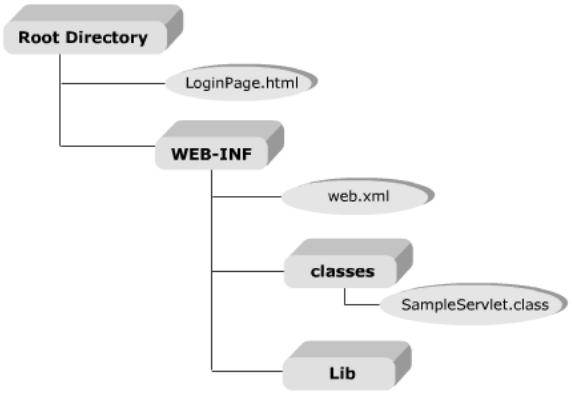


Рис. 54. Стандартная структура упаковки

Эта стандартная структура упаковки хранится в файле Enterprise Archive (EAR), который создается во время процесса развертывания веб-приложения. После размещения специальных файлов приложения в стандартной структуре каталогов, необходимо упаковать приложение в файл веб-архива (Web Archive (WAR)). WAR- файл – это заархивированное J2EE веб-приложение. Для создания WAR-файла с помощью базовых компонентов Java, в командной строке введится следующая команда:

```
jar cvf <имя war-файла>
```

При этом создается файл с расширением war, который может быть развернут на сервере приложений.

Большинство серверов приложений предоставляют утилиты, которые автоматически генерируют дескриптор развертывания и выполняют упаковку. Кроме того, все современные IDE, к которым относятся и предлагаемый вашему вниманию Eclipse, содержит все необходимые средства для создания JavaEE приложения и его развертывания на сервере.

Лабораторный практикум: лабораторная работа № 4, часть 1.

Servlet API и события жизненного цикла

Servlet API

Как обсуждалось ранее, Servlet API – это коллекция классов и интерфейсов Java, которые позволяют создавать сервлеты. Эти классы и интерфейсы Servlet API находятся в пакетах `javax.servlet` и `javax.servlet.http`.

Классы и интерфейсы пакета `javax.servlet` обеспечивают взаимодействие между сервлетом и клиентом. При создании сервлета необходимо реализовать интерфейс `Servlet` непосредственно или расширением класса, который реализует этот интерфейс. Интерфейс `Servlet` определяет различные методы жизненного цикла. Другими широко используемыми интерфейсами пакета `javax.servlet` являются:

- Интерфейс `ServletRequest`
- Интерфейс `ServletResponse`
- Интерфейс `ServletContext`

Рис. 55 показывает обобщенную объектную модель, представляющую иерархию классов пакета `javax.servlet`:

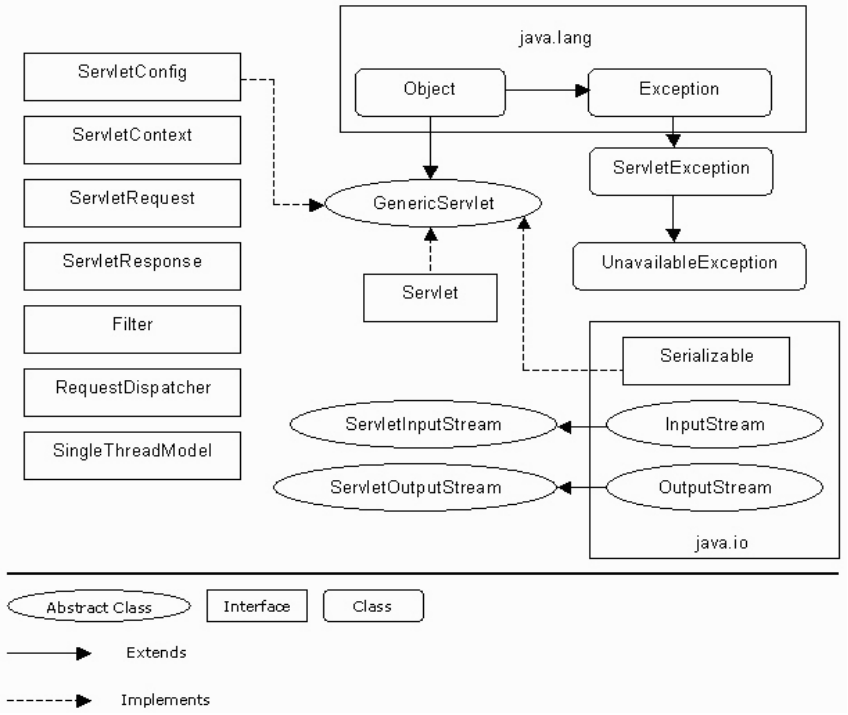


Рис. 55. Обобщенная объектная модель пакета javax.servlet

Интерфейс `ServletRequest`

Интерфейс `ServletRequest` содержит методы для обработки клиентских запросов к сервлету. При вызове сервлета, веб-контейнер передает объекты, которые реализуют интерфейсы `ServletRequest` и `ServletResponse`, методу `service()` сервлета. Табл. 15 описывает различные методы интерфейса `ServletRequest`.

Таблица 15

Метод	Описание
`public String getParameter(String paramName)`	Возвращает объект String, содержащий значение заданного параметра запроса.
`public String[] getParameterValues(String paramName)`	Возвращает массив объектов String, содержащий все значения параметра запроса.
`public Enumeration getParameterNames()`	Возвращает Enumeration, содержащий все имена параметров в виде объектов String, которые содержит запрос к сервлету.
`public String getRemoteHost()`	Возвращает String, содержащий точное имя компьютера, с которого был отправлен запрос.
`public String getRemoteAddr()`	Возвращает String, содержащий IP-адрес компьютера, с которого был отправлен запрос.

Интерфейс `ServletResponse`

Интерфейс `ServletResponse` содержит методы, которые позволяют сервлету реагировать на клиентские запросы. Сервлет может отправить ответ в виде символьных или двоичных данных. Поток `PrintWriter` может быть использован для передачи символьных данных в ответе сервлета, а поток `ServletOutputStream` – для передачи двоичных данных в ответе сервлета. Табл. 16 описывает различные методы интерфейса `ServletResponse`.

Таблица 16

Метод	Описание
`public ServletOutputStream getOutputStream() throws IOException`	Возвращает объект класса `ServletOutputStream`, который представляет выходной поток для передачи двоичных данных в ответе.
`public PrintWriter getWriter() throws IOException`	Возвращает объект класса `PrintWriter`, который сервлет ис-

	пользует для передачи символьных данных в ответе.
public void setContent-Type(String type)	Устанавливает тип Multipurpose Internet Mail Extensions (MIME – Многоцелевые расширения электронной почты) для ответа сервлета. Некоторые из MIME-типов: text/plain, image/jpeg и text/html.

Интерфейс `ServletContext`

Интерфейс `ServletContext` предоставляет информацию сервлетам о среде, в которой они выполняются. Контекст также называют контекстом сервлета или веб-контекстом, и создается веб-контейнером как объект интерфейса `ServletContext`. Этот объект представляет контекст, внутри которого выполняется веб-приложение. Веб-контейнер создает объект `ServletContext` для каждого развернутого веб-приложения. Можно использовать объект `ServletContext` для определения пути к другим файлам веб-приложения, для доступа к другим сервлетам веб-приложения и записи сообщений в журнал сервера приложений. Также можно использовать объект `ServletContext` для установки атрибутов, к которым другие сервлеты вашего приложения могут получать доступ. Табл. 17 описывает методы интерфейса `ServletContext`:

Таблица 17

Метод	***Описание***
public void setAttribute(String, Object)	Связывает объект с именем и сохраняет пару имя/значение как атрибут объекта `ServletContext`. Если атрибут уже существует, то этот метод замещает существующий атрибут.
public Object getAttribute(String attrname)	Возвращает объект, хранимый в объекте `ServletContext` с именем, переданным как параметр.
public Enumeration getAttributeNames()	Возвращает Enumeration объектов String, который содержит имена всех атрибутов контекста.
public String getInitParameter(String pname)	Возвращает значение параметра инициализации с именем, переданным как параметр.

Метод	Описание
`public Enumeration getInitParameterNames()`	Возвращает Enumeration объектов String, который содержит имена всех параметров инициализации.
`public int getMajorVersion()`	Возвращает целое значение, которое определяет номер версии Servlet API, которую поддерживает веб-контейнер. Если ваш веб-контейнер поддерживает Servlet API версии 2.4, этот метод вернет 2.
`public int getMinorVersion()`	Возвращает целое значение, которое определяет номер подверсии Servlet API, которую поддерживает веб-контейнер. Если ваш веб-контейнер поддерживает Servlet API версии 2.4, этот метод вернет 4.

Для использования объекта `ServletContext` необходимо получить объект `ServletContext` в методе `init()` сервлета. Метод `getServletContext()` интерфейса `ServletConfig` позволяет получить объект `ServletContext`. Можно использовать следующий фрагмент кода для получения объекта `ServletContext`:

```
ServletContext ctx;
public void init(ServletConfig cfig)
{
       ctx = cfig.getServletContext();
}
```

После получения объекта `ServletContext` можно установить атрибуты объекта `ServletContext` с помощью метода `setAttribute()`. Так как объект `ServletContext` доступен всем сервлетам веб-приложения, другие сервлеты могут извлекать атрибуты из объекта `ServletContext`, используя метод `getAttribute()`. Например, рассмотрим веб-приложение, в котором сервлет сохраняет URL JDBC как атрибут объекта `ServletContext` для доступа к базе данных. Другие сервлеты приложения могут извлекать этот URL из объекта `ServletContext` для доступа к базе данных.

В следующем примере используются методоы `setAttribute()` и `getAttribute()` интерфейса `ServletContext` для создания сервлета `SettingCntx`, который устанавливает URL JDBC как атрибут контекста:

```
import javax.servlet.*;
import java.io.*;

public class SettingCntx extends GenericServlet
{
```

```
            ServletContext ctx;

            public void init(ServletConfig cfig)
            {
                    /* Получение объекта ServletContext*/
                   ctx = cfig.getServletContext();
            }

            public void service(ServletRequest request, ServletRe-
sponse response) throws ServletException, IOException
            {
                  /* Установка атрибута контекста */
                  ctx.setAttribute("URL","jdbc:odbc:EmployeesDB");
                  /* Получение объекта PrintWriter*/
                  PrintWriter pw = response.getWriter();
                   /* Отправка ответа, что атрибут URL установлен
*/
                  response.setContentType("text/html");
                  pw.println("<B>The JDBC URL has been set as a
context attribute</B>");
            }
      }
```

В приведенной выше программе, сервлет `SettingCntx` получает объект `ServletConext` в методе `init()`. В методе `service()` сервлет использует метод `setAttribute()` для присвоения значения `jdbc:odbc:EmployeesDB` атрибуту `URL`. В конце сервлет использует объект `PrintWriter` для вывода сообщения, что атрибут установлен.

Сервлет `RetrievingCntx` извлекает атрибут `URL` и выводит значение атрибута. Можно использовать следующий код для создания сервлета `RetrievingCntx`, чтобы извлечь и вывести атрибут `URL`:

```
      import javax.servlet.*;
      import java.io.*;

      public class RetrievingCntx extends GenericServlet
      {
            ServletContext ctx;
              String url;

            public void init(ServletConfig cfig)
            {
                  /* Получение объекта ServletContext */
                  ctx = cfig.getServletContext();
            }
            public void service (ServletRequest request, ServletRe-
sponse response) throws ServletException, IOException
            {
                  /* Извлечение атрибута URL */
                  url = (String)ctx.getAttribute("URL");
                  /* Получение объекта PrintWriter */
                  PrintWriter pw = response.getWriter();
                  /* Отправка ответа для вывода значения атрибута
URL */
                  response.setContentType("text/html");
```

```
                    pw.println("<B>The URL value is  </B>:     "+ url
+ "<BR>");
                }
        }
```

В приведенной выше программе, сервлет `RetrievingCntx` получает объект `ServletConext` в методе `init()`. В методе `service()` сервлет использует метод `getAttribute()` для получения значения атрибута `URL`. В конце сервлет использует объект `PrintWriter` для вывода значения атрибута.

При выполнении сервлета `SettingCntx`, сервлет устанавливает URL как атрибут объекта `ServletContext`. Сервлет отправляет ответ, что атрибут установлен, как показано на рис. 56.

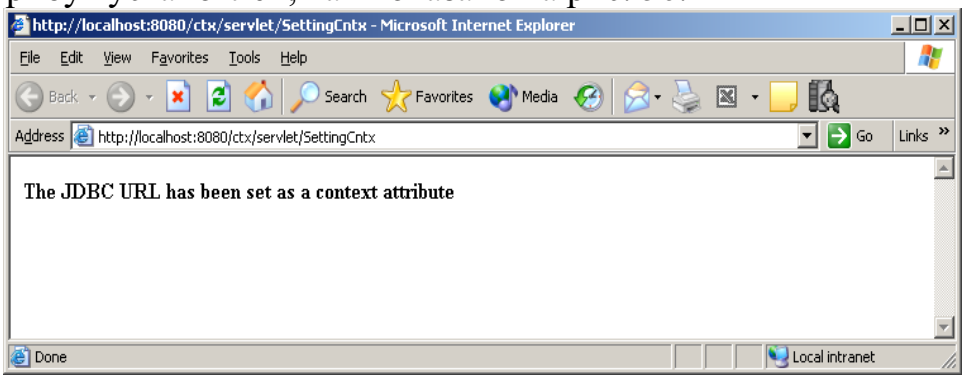


Рис. 56. Результат работы сервлета SettingCntx

В процессе выполнении сервлета `RetrievingCntx` извлекает URL, который был установлен сервлетом `SettingCntx`. Сервлет отправляет ответ для вывода значения URL, как показано на рис. 57.

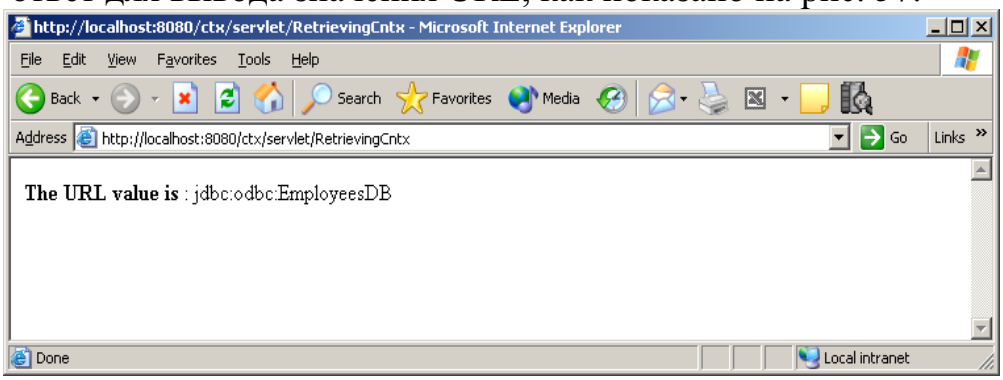


Рис. 57. Результат работы сервлета RetrievingCntx

Параметры инициализации контекста

Параметры инициализации контекста – это пары имя/значение, которые можно задавать во время развертывания веб-приложения. Можно использовать параметры инициализации контекста для предоставления информации, которая доступна для всех сервлетов веб-приложения, например, можно задать ваш адрес электронной почты как

параметр инициализации контекста. Затем вы можете извлечь это значение в других сервлетах вашего приложения, которым необходимо вывести ваш адрес.

При развертывании веб-приложения веб-контейнер считывает параметры инициализации из дескриптора развертывания и инициализирует ими объект `ServletContext`. Можно использовать методы `getInitParameter()` и `getInitParameterNames()` объекта `ServletContext` для получения значений параметров в сервлете.

Элемент `context-param` дескриптора развертывания определяет пару имя/значение параметра инициализации контекста. Следующий фрагмент кода демонстрирует элемент `context-param` дескриптора развертывания, который задает параметр с именем email_id и значением `john_adam@hotmail.com`:

```
<context-param>
      <param-name> email_id</param-name>
      <param-value> john_adam@hotmail.com</param-value>
</context-param>
```

Сервер приложений может поддерживать распределенные контейнеры, которые используют различные Java Virtual Machines (JVMs – Виртуальные машины Java), которые могут быть расположены на различных компьютерах. Веб-приложения, которые разрабатываются для запуска в распределенных контейнерах, называются распределенными веб-приложениями. Можное указать, что приложение является распределенным, используя элемент распределения в дескрипторе развертывания веб-приложения.

Пакет javax.servlet.http

Пакет `javax.servlet.http` является расширением пакета `javax.servlet`. Классы и интерфейсы этого пакета обрабатывают сервлеты, которые работают, используя HTTP. Эти сервлеты также называются HTTP-сервлетами. Вам нужно расширить класс `HttpServlet` для создания HTTP-сервлета. Ниже представлены наиболее часто используемые интерфейсы пакета `javax.servlet.http`:

- Интерфейс `HttpServletRequest`
- Интерфейс `HttpServletResponse`
- Интерфейс `HttpSession`

Рис. 58 представляет обобщенную объектную модель, представляющую иерархию классов пакета `javax.servlet.http`:

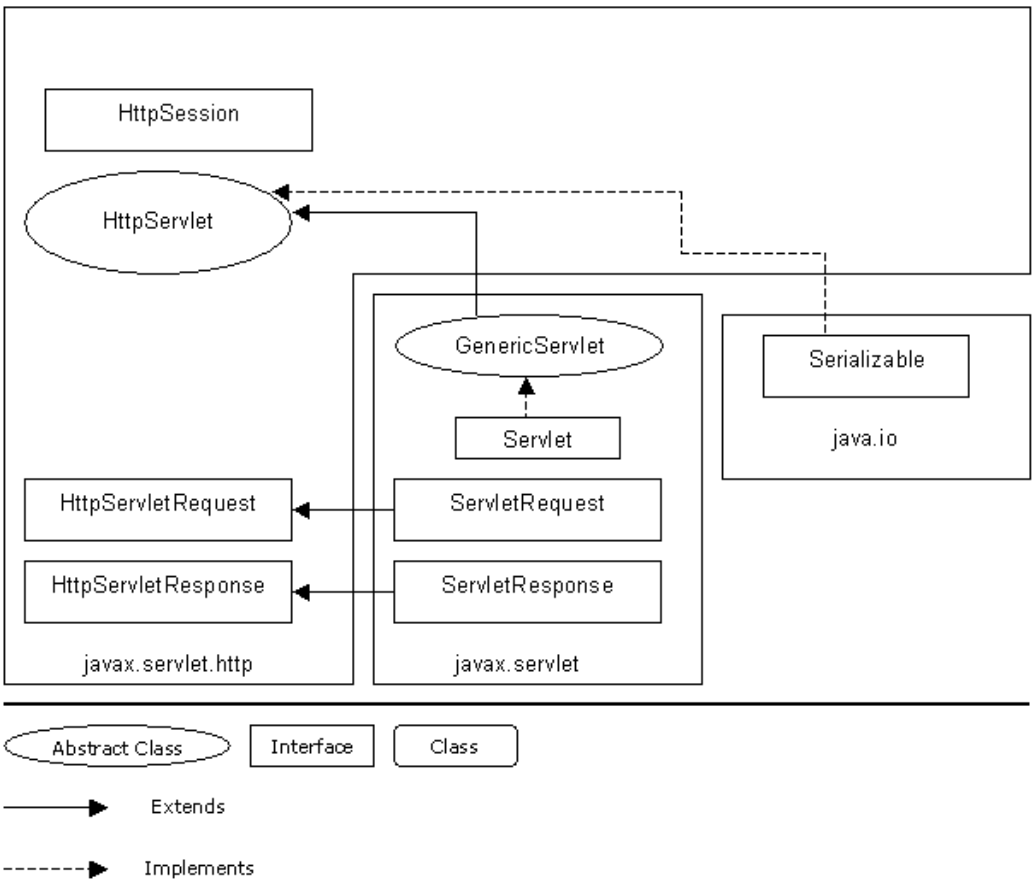


Рис. 58. Обобщенная объектная модель пакета javax.servlet.http

Интерфейс `HttpServletRequest`

Интерфейс `HttpServletRequest` расширяет интерфейс `ServletRequest` для представления информации из запроса, отправленного клиентом HTTP. Он включает поддержку для получения параметров запроса и доступа к информации из заголовка HTTP-запроса.

HTTP-запросы имеют несколько связанных заголовков, предоставляющих дополнительную информацию о клиенте, такую как название и версия браузера, отправляющего запрос. Ниже представлены некоторые из важных заголовков HTTP-запросов:

`Accept`: Определяет MIME-тип, который клиент предпочитает использовать.

`Accept-Language`: Определяет язык, на котором клиент предпочитает получать запрос.

`User-Agent`: Определяет название и версию браузера, отправившего запрос.

Табл. 18 описывает некотоые методы интерфейса `HttpServletRequest`.

Таблица 18

Метод	*Описание*
`public String getHeader(String fieldname)`	Возвращает значение поле заголовка запроса, такое как Cache-Control и Accept-Language, указанное в параметре.
`public Enumeration getHeaders(String sname)`	Возвращает все значения, связанные с конкретным заголовком запроса в виде Enumeration объектов String
`public Enumeration getHeaderNames()`	Возвращает имена всех заголовков запроса, к которым сервлет может получить доступ, в виде Enumeration объектов String.

Сервлеты используют такие методы, как `getHeader()`, `getHeaderNames()` и `getHeaders()` для получения значений из заголовков HTTP-запроса. Можно использовать следующий код для получения информации из заголовка запроса:

```
/* Импорт необходимых пакетов */
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

public class HttpRequestHeaderDemo extends HttpServlet
{
      public void doGet(HttpServletRequest req, HttpServletRe-
sponse res) throws ServletException, IOException
      {
            res.setContentType("text/html");
            PrintWriter out=res.getWriter();
                /* Получение всех имен заголовков в Enumeration */
            Enumeration hnames=req.getHeaderNames();
            out.println("<H3>The request headers are:</H3>");
                /* Проверка, есть ли еще элементы в Enumeration */
            while(hnames.hasMoreElements())
            {
             /* Получение имени заголовка */
                  String hname=(String) hnames.nextElement();
             /* Получение всех значений из заголовка как Enumera-
tion, соответствующих имени заголовка, переданного как параметр
метода */
                  Enumeration hvalues=req.getHeaders(hname);
                  out.println("<BR>");
                  if(hvalues!=null)
                  {
           /* Проверка, есть ли еще элементы в Enumeration */

                        while(hvalues.hasMoreElements())
                        {
              /*Получение значения из заголовка*/
```

```
                                String hvalue=(String) hval-
ues.nextElement();
        /*Отправка имени заголовка и его значения в ответе*/
                                out.println(hname + ": " + hvalue);
                            }
                        }
                    }
                }
            }
```

В приведенной программе, метод `getHeaderNames()` извлекает все имена заголовков, отправленные в запросе, и сохраняет их в объекте Enumeration. Затем сервлет получает значение из заголовков, используя метод `getHeaders()`. В заключение, сервлет отправляет имена заголовков и значения клиенту.

Рис. 59 демонстрирует различные заголовки запроса, которые получает сервлет.

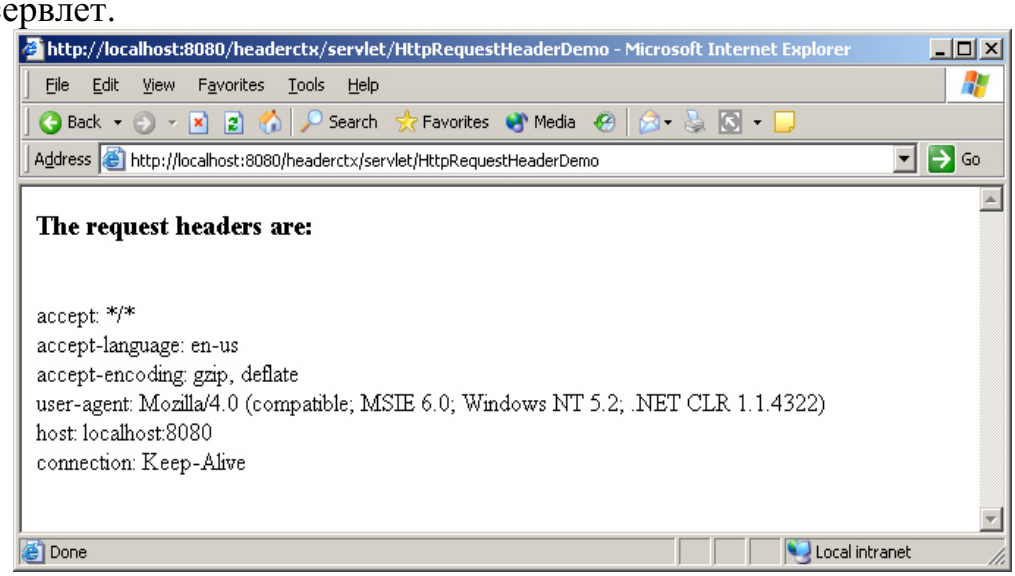


Рис. 59. Результат работы сервлета HttpRequestHeaderDemo

Интерфейс `HttpServletResponse`

Интерфейс `HttpServletResponse` расширяет интерфейс `ServletResponse` и предоставляет методы для обработки ответов, кодов статуса и заголовков ответов для сервлетов, которые взаимодействуют с помощью HTTP. Табл. 19 описывает различные методы интерфейса `HttpServletResponse`.

Таблица 19

Метод	***Описание***
`void setHeader(String hname, String hvalue)`	Устанавливает значение заголовка hname в hvalue. Если заголовок уже установлен, новое значение стирает предыдущее.

Метод	*Описание*
void setIntHeader(String hname, int hvalue)	Устанавливает значение заголовка hname в значение hvalue типа int. Если заголовок уже установлен, новое значение затирает предыдущее.
void setDateHeader(String hname, long datev)	Устанавливает значение заголовка hname в значение datev типа long. Datev представляет количество миллисекунд, прошедшее с полуночи от 1 января 1970.
void addHeader(String hname, String hvalue)	Добавляет заголовок hname со значением hvalue. Этот метод добавляет новый заголовок, если заголовок уже существует.
void addIntHeader(String hname, int hvalue)	Добавляет заголовок hname со значением hvalue типа int.
void addDateHeader(String hname, long datev)	Добавляет заголовок hname со значением, равным дате datev. Значение datev должно быть в миллисекундах, которые прошли с полуночи 1 января 1970.
boolean containsHeader(String hname)	Возвращает true, если заголовок hname уже установлен с конкретным значением, и false, в противном случае.
void sendRedirect(String url)	Перенаправляет запрос указанному URL.

Установка заголовков ответа

Вы можете предоставить дополнительную информацию об ответе, который отправляется клиенту сервером приложений, добавив новые HTTP-заголовки. Ниже представлены некоторые дополнительные заголовки HTTP-ответа, которые может устанавливать сервлет:

`Content-Type`: Определяет MIME-тип данных, отправленных сервлетом, такой как `text/html` и `text/plain`.

`Cache-control`: Определяет информацию для кэширования сервлета. Если значение `Cache-control` установлено в `no-store`, то сервлет не кэшируется броузером или прокси-сервером. Значение `max-age` задает время в секундах, в течение которого сервлет должен находиться в кэше.

`Expires`: Определяет время, когда содержимое сервлета может измениться или когда его данные станут неверными, и что требуется перезагрузка сервлета браузером для вывода обновленных данных.

Сервлеты используют такие методы как `setHeader()` и `setDateHeader()`, для установки значений HTTP-заголовков в объекте ответа. Можно использовать следующий фрагмент кода для установки данных HTTP-заголовке:

```
      public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException
      {
      PrintWriter out = res.getWriter();
      /* Задание значения типа long для хранения времени до обновления*/
      long mseconds;
      java.util.Date date = new java.util.Date();
      mseconds = date.getTime() + 10000;
      /* Установка заголовка Content-Type со значением text/html */
      res.setHeader("Content-Type", "text/html");
      /* Установка заголовка Expires со значением mseconds */
      res.setDateHeader("Expires", mseconds);
      /* Отправка текущих даты и времени в ответе */
      out.println("<HTML><BODY>");
      out.println(new java.util.Date());
      out.println("</BODY></HTML>");
      }
```

В приведенном фрагменте программы первый вызов метода `setHeader()` устанавливает значение поля заголовка `Content-Type` в `text/html`. Затем значение поля заголовка `Expires` устанавливается в mseconds. Переменная mseconds является значением типа long, полученным при добавлении 10000 миллисекунд к количеству миллисекунд, возвращенным методом `getTime()`. Метод `getTime()` класса `java.util.Date` возвращает количество миллисекунд, прошедшее с полночи 1 января 1970. Значение заголовка `Expires` гарантирует, что HTML-данные, сгенерированные сервлетом, которые показывают дату конечному пользователю, устареет через десять секунд. Это значит, что если конечный пользователь не запросит страницу через десять секунд, браузер отправит новый запрос серверу и выведет обновленные данные.

Перенаправление запросов клиентов

Вы можете использовать метод `sendRedirect()` интерфейса `HttpServletResponse` для перенаправления запроса другому URL. Рассмотрим ситуацию, когда на веб-сайте образовательного учреждения предлагается онлайновая регистрация на технические и медицинские курсы. На начальной странице веб-сайта находятся две альтернативные кнопки, для выбора технического или медицинского курса. Если вы выберите радиокнопку с техническим курсом и нажмете кнопку Submit, откроется онлайновая регистрационная форма для технического курса, в противном случае открывается онлайновая регистрационная форма для

медицинского курса. Следующий фрагмент кода показывает использование метода sendRedirect():

```
      public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException
      {
              /* Получение имени выбранной пользователем радиокнопки, от-
правленной как параметр запроса */
            String yourchoice=req.getParameter("chooseoption");
             /* Если выбрана радиокнопка "medical", перенаправление на
страницу medical.html*/
            if(yourchoice.equals("medical"))
            res.sendRedirect("http://localhost:8080/ctx/servlet/medical.
html");
      /* Если выбрана радиокнопка "engineering", перенаправление на
страницу engineering.html*/
            if(yourchoice.equals("engineering"))
            res.sendRedirect("http://localhost:8080/ctx/servlet/engineer
ing.html");}
```

Интерфейс HttpSession

Интерфейс HttpSession содержит методы для сохранения состояния конечного пользователя в веб-приложении. Объект интерфейса HttpSession предоставляет поддержку для отслеживания и управления сеансом конечного пользователя. Табл. 20 описывает различные методы интерфейса HttpSession:

Таблица 20

Метод	*Описание*
public void setAttribute(String name, Object value)	Связывает объект с именем и сохраняет пару name/value как атрибут объекта HttpSession. Если атрибут уже существует, то этот метод замещает существующий атрибут.
public Object getAttribute(String name)	Извлекает объект String, указанный в параметре, из объекта сеанса. Если не найден объект для указанного атрибута, то метод getAttribute() возвращает null.
public Enumeration getAttributeNames()	Возвращает Enumeration, который содержит имена всех объектов, которые связаны с атрибутами объекта сеанса.

API жизненного цикла сервлета

Во время жизненного цикла сервлета возникают различные события, такие как создание контекста сервлета, создание сеанса и добавле-

ние атрибутов в контекст сервлета. Веб-контейнер уведомляет класс-слушатель, когда возникает событие в течение жизненного цикла сервлета и для того, чтобы получить уведомление о событии, классу-слушателю необходимо расширить интерфейс-слушатель Servlet API.

Типы событий

Ниже представлены различные события, которые генерируются во время жизненного цикла сервлета:

- События запроса к сервлету
- События контекста сервлета
- События сеанса HTTP

События запроса к сервлету

Следующие два интерфейса представляют события запроса сервлета, которые имеют отношение к изменениям в объектах запроса, связанных с веб-приложением:

- javax.servlet.ServletRequestEvent
- javax.servlet.ServletRequestAttributeEvent
- Веб-контейнер создает объект ServletRequestEvent при:
- Инициализации объекта запроса, когда приходит запрос
- Удалении объекта запроса

Веб-контейнер создает объект `ServletRequestAttributeEvent`, когда происходит какое-либо изменение в атрибуте запроса к сервлету. Веб-контейнер создает объект `ServletRequestEvent` во время:

- Добавления атрибута к объекту запроса к сервлету
- Удаления атрибута из объекта запроса к сервлету
- Замены атрибута в объекте запроса к сервлету другим атрибутом с таким же именем

События контекста сервлета

События, которые связаны с изменениями в контексте веб-приложения, известны как события контекста сервлета. Следующие два интерфейса представляют события контекста сервлета:

- javax.servlet.ServletContextEvent
- javax.servlet.ServletContextAttributeEvent

Веб-контейнер создает объект `ServletContextEvent` во время:

- Создания объекта ServletContext при инициализационной фазе жизненного цикла сервлета.
- Удаления объекта ServletContext.
- Веб-контейнер генерирует объект ServletContextAttributeEvent, когда происходит какое-либо изменение в атрибуте контекста сервлета веб-приложения. Веб-контейнер создает объект `ServletContexAttributetEvent` при:
- Добавлении атрибута объекту контекста сервлета
- Удалении атрибута у объекта контекста сервлета
- Замене атрибута в объекте контекста сервлета другим атририбутом с таким же именем.

События сеанса HTTP

События сеанса HTTP связаны с изменениями в объекте сеанса сервлета. Следующие интерфейсы представляют события сеанса сервлета:

- javax.servlet.http.HttpSessionEvent
- javax.servlet.http.HttpSessionAttributeEvent
- javax.servlet.http.HttpSessionActivationEvent
- javax.servlet.http.HttpSessionBindingEvent
- Веб-контейнер создает объект HttpSessionEvent при:
- Создании нового сеанса
- Недействительности сеанса
- Истечении срока сеанса

Веб-контейнер генерирует объект `HttpSessionAttributeEvent`, когда возникает какое-либо изменение в объекте атрибута сеанса. Веб-контейнер создает объект `HttpSessionAttributetEvent` при:

- Добавлении атрибута к объекту сеанса.
- Удалении атрибута из объекта сеанса.
- Замене атрибута в объекте сеанса.

Веб-контейнер генерирует объект `HttpSessionBindingEvent`, когда объект сервлета связывается с сеансом или разрывает связь сеанса. Связывание объекта с сеансом значит, что объект сотносится с сеансом. Разрыв связи объекта сервлета значит, что объект не сотносится с сеансом. Веб-контейнер генерирует объект `HttpSessionActivationEvent`, когда сеанс активизируется или деактивируется.

Обработка событий жизненного цикла сервлета

Классы, которые получают уведомления о событиях жизненного цикла сервлета, известны как слушатели событий. Эти классы-слушатели реализуют один или более интерфейсов слушателей событий сервлета, которые определены в Servlet API. Классы-слушатели могут быть логически разделены на следующие категории:

- Слушатели запросов к сервлету
- Слушатели контекста сервлета
- Слушатели сеанса HTTP

Слушатели запросов к сервлету

Слушатели запросов к сервлету – это те классы, которые слушают и обрабатывают события запросов к сервлету. Слушатели запросов к сервлету могут реализовывать следующие интерфейсы для получения уведомления о событиях запроса:

- javax.servlet.ServletRequestListener
- javax.servlet.ServletRequestAttributeListener

Интерфейс ServletRequestListener

Интерфейс `ServletRequestListener` позволяет классу-слушателю получать уведомления от веб-контейнера об изменениях в объекте запроса сервлета. Необходимо реализовать интерфейс `ServletRequestListener` в классе-слушателе для получения уведомлений о событиях запроса. Ниже представлены методы интерфейса `ServletRequestListener`:

`void requestInitialized(ServletRequestEvent e)`: Уведомляет класс-слушатель об инициализации запроса к сервлету, связанного с веб-приложением.

`void requestDestroyed(ServletRequestEvent e)`: Уведомляет сервлет об удалении запроса к сервлету, связанному с веб-приложением.

Интерфейс ServletRequestAttributeListener

Интерфейс `ServletRequestAttributeListener` позволяет классу-слушателю получать уведомления от веб-контейнера об изменениях в атрибуте запроса. Ниже представлены методы интерфейса

ServletRequestAttributeListener, которые можно использовать в классе-слушателе:

void attributeAdded(ServletRequestAttributeEvent srae): Уведомляет класс-слушатель о добавлении атрибута запроса.

void attributeRemoved(ServletRequestAttributeEvent srae): Уведомляет класс-слушатель об удалении атрибута запроса.

void attributeReplaced(ServletRequestAttributeEvent srae): Уведомляет класс-слушатель о замене существующего атрибута запроса новым атрибутом запроса.

Слушатели контекста сервлета

Слушатели контекста сервлета – это те классы-слушатели, которые обрабатывают события контекста сервлета. Слушатели контекста сервлета могут реализовывать следующие интерфейсы для получения уведомлений об изменениях в контексте сервлета:

- javax.servlet.ServletContextListener
- javax.servlet.ServletContextAttributeListener

Интерфейс javax.servlet.ServletContextListener

Интерфейс ServletContextListener позволяет классу-слушателю получать уведомления от веб-контейнера об изменениях в контексте сервлета веб-приложения. Ниже представлены методы интерфейса ServletContextListener:

void contextInitialized(ServletContextEvent ce): Уведомляет класс-слушатель об инициализации контекста сервлета веб-приложения.

void contextDestroyed(ServletContextEvent ce): Уведомляет класс-слушатель об удалении контекста сервлета веб-приложения.

> ***В распределенных приложениях слушатели контекста сервлета не получают уведомления об изменениях в контексте сервлета различных JVM. Это происходит из-за того, что события контекста сервлета не могут распространяться между несколькими JVM.***

Интерфейс javax.servlet.ServletContextAttributeListener

Интерфейс `ServletContextAttributeListener` позволяет классу-слушателю получать уведомления от веб-контейнера об изменениях, имеющих место в атрибутах контекста сервлета. Ниже представлены методы интерфейса `ServletContextAttributeListener`:

`void attributeAdded (ServletContextAttributeEvent cae)`: Уведомляет класс-слушатель о добавлении атрибута в контекст сервлета.

`public void attributeRemoved (ServletContextAttributeEvent cae)`: Уведомляет класс-слушатель об удалении атрибута из контекста сервлета.

`public void attributeReplaced (ServletContextAttributeEvent cae)`: Уведомляет класс-слушатель о замене существующего атрибута контекста сервлета новым атрибутом.

Следующий фрагмент кода демонстрирует использование методов `attributeAdded()`, `attributeReplaced()` и `attributeRemoved()` интерфейса `ServletContextAttributeListener`:

```
/*Вызывается, при добавлении атрибута в контекст сервлета*/
public void attributeAdded(ServletContextAttributeEvent scae)
{
      ServletContext sc=scae.getServletContext();
        /*Имя атрибута добавляется в журнал сервера*/
      sc.log("The Attribute Name:"+scae.getName());
        /*Значение атрибута добавляется в журнал сервера*/
      sc.log("The Attribute Value:"+scae.getValue());
}
/*Вызывается при замене имени атрибута в контексте сервлета*/
public void attributeReplaced(ServletContextAttributeEvent scae)
{
        /*Запись сообщения, что атрибут заменен*/
      ServletContext sc=scae.getServletContext();
      sc.log("The Attribute is replaced in Servlet Context ");

}
/*Вызывается при удалении атрибута из контекста сервлета*/
public void attributeRemoved(ServletContextAttributeEvent scae)
{
        /*Запись сообщения, что атрибут удален*/
      ServletContext sc=scae.getServletContext();
      sc.log("The Attribute is removed from Servlet Context ");
}
```

В распределенных приложениях слушатели атрибута контекста сервлета получают уведомления об изменениях в атрибутах контекста сервлета только от локальной JVM.

Слушатели сеанса HTTP

Классы-слушатели, которые получают уведомления о событиях сеанса HTTP, известны как слушатели сеанса HTTP. Классы-слушатели сеанса HTTP могут реализовывать следующие интерфейсы для получения уведомлений о событиях сеанса:

- javax.servlet.http.HttpSessionListener
- javax.servlet.http.HttpSessionAttributeListener
- javax.servlet.http.HttpSessionActivationListener

Интерфейс javax.servlet.http.HttpSessionListener

Интерфейс `HttpSessionListener` позволяет классу-слушателю получать уведомления от веб-контейнера об изменениях, имеющих место в объекте сеанса, связанного с веб-приложением. Ниже представлены методы интерфейса `HttpSessionListener`:

`void sessionCreated (HttpSessionEvent hse)`: Уведомляет класс-слушатель о создании объекта сеанса.

`void sessionDestroyed (HttpSessionEvent hse)`: Уведомляет класс-слушатель об удалении существующего объекта сеанса.

Интерфейс javax.servlet.http.HttpSessionAttributeListener

Интерфейс `HttpSessionAttributeListener` позволяет классу-слушателю получать уведомления от веб-контейнера об изменении в атрибутах объекта сеанса. Ниже представлены методы интерфейса `HttpSessionAttributeListener`:

`void attributeAdded(HttpSessionBindingEvent sbe)`: Уведомляет класс-слушатель о добавлении атрибута в объект сеанса.

`void attributeRemoved(HttpSessionBindingEvent sbe)`: Уведомляет класс-слушатель об удалении атрибута из объекта сеанса.

`void attributeReplaced(HttpSessionBindingEvent sbe)`: Уведомляет класс-слушатель о замене атрибута сеанса новым атрибутом.

> ***В распределенных приложениях слушатели атрибута контекста сервлета получают уведомления об изменениях в атрибутах сеанса только от локальной JVM.***

Интерфейс javax.servlet.http.HttpSessionActivationListener

Интерфейс `HttpSessionActivationListener` позволяет классу-слушателю получать уведомления от веб-контейнера об изменениях в состоянии объекта сеанса, сопоставленного с сервлетом. Ниже представлены методы интерфейса `HttpSessionActivationListener`:

`void sessionDidActivate(HttpSessionEvent se)`: Уведомляет класс-слушатель об активизации нового сеанса в веб-приложении.

`void sessionWillPassivate(HttpSessionEvent se)`: Уведомляет класс-слушатель о пассивизации существующего сеанса в веб-приложении.

Понятие элементов дескриптора развертывания

Дескриптор развертывания – это файл XML с именем `web.xml`, который содержит конфигурационную информацию о веб-приложении. Существует только один файл `web.xml` для каждого веб-приложения. Элементы файла `web.xml` позволяют задавать параметры инициализации для сервлета, MIME-тип для различных файлов, указывать классы-слушатели и устанавливать шаблон URL для сервлета. Ниже описаны некоторые из часто используемых элементов дескриптора развертывания вместе с примерами их использованием:

`<context-param>`: Задает параметры инициализации контекста сервлета веб-приложения, как демонстрируется в следующем фрагменте кода:

```
<context-param>
<param-name> rmihost </param-name>
<param-value> 192.162.100.4 </param-value>
</context-param>
```

<init-param>: Задает параметр инициализации для сервлета. В отличие от параметра инициализации контекста, который доступен всем сервлетам веб-приложения, этот параметр доступен только сервлету, для которого он объявлен. Следующий фрагмент кода демонстрирует использование элемента init-param:

```
<init-param>
<param-name> title <param-name>
<param-value> This is the First Servlet </param-value>
</init-param>
```

<mime-mapping>: Задает соответствие между расширением файла и MIME-типом, как показано в следующем фрагменте кода:

```
<mime-mapping>
<extension>html</extension>
<mime-type> text/html </mime-type>
</mime-mapping>
```

<servlet-mapping>: Задает соответствие между сервлетом и шаблоном URL, как показано в следующем фрагменте кода:

```
<servlet-mapping>
<servlet-name>MyServlet</servlet-name>
<url-pattern>/test</url-pattern>
</servlet-mapping>
```

После определения данного соответствия в дескрипторе развертывания сервлета, веб-контейнер будет сопоставлять следующий URL сервлету `MyServlet`, как показано ниже:

```
http://localhost:8080/servletctx/test
```

`<session-config>`: Задает сеансовую информацию для сервлета, такую как значение тайм-аута для сеанса, как показано в следующем фрагменте кода:

```
<session-config>
      <session-timeout>30</session-timeout>
</session-config>
```

Данный выше элемент дескриптора развертывания задает, что сеанс сервлета закончится через 30 минут.

`<listener>`: Задает имя класса-слушателя, который реагирует на события времени жизни сервлета, как показано в следующем фрагменте кода:

```
<listener>
<listener-class>ContextListenerHandler</listener-class>
</listener>
```

Управление сеансами servlet

Управление сеансом – это процесс отслеживания действий пользователя на веб-страницах. Например, в онлайновом магазине, пользователь может выбрать продукт и добавить его в магазинную тележку. Когда пользователь переходит на другую страницу, товары в магазинной тележке остаются, так что пользователь может проверить предметы в тележке и затем заказать их.

Отслеживание сеанса также может быть использовано для учета предпочтений пользователя, поэтому управление сеансом является неотъемлемой частью веб-приложения.

Приемы управления сеансом

HTTP является протоколом без сохранения состояния и, поэтому, не может хранить информацию о действиях пользователя на веб-страницах. Однако существуют определенные приемы, которые помогают сохранять информацию о пользователе веб-страниц, использующего протокол HTTP, котрые представлены приемы:

- Скрытое поле формы

- Перезапись URL
- Cookies
- Servlet Session API

Скрытое поле формы

Могут использоваться скрытые поля формы для сохранения информации о сеансе пользователя, когда пользователь взаимодействует с веб-приложением. Скрытое поле формы вставляется в HTML-страницу, и оно невидимо при просмотре из броузера как показано в следующем фрагменте кода в форме HTML-страницы:

```
<HTML>
<FORM METHOD=″GET″ Action=″/servlet/TestServlet″>
<input type="hidden" name="id" value="L123">
- - - - -
- - - -
</HTML>
```

Например, в онлайновом магазине скрытые поля формы используются для сохранения информации о пользователе. Пользователь записывает имя в форму HTML. Эта информация читается сервлетом. Сервлет затем генерирует форму HTML, которая содержит скрытое поле формы и кнопку **Submit**. Когда пользователь нажимает кнопку **Submit**, другой сервлет получает имя пользователя, хранимое в скрытом поле формы, и выводит сообщение с приветствием.

Следующий код используется для создания страницы входа, которая принимает имя пользователя и вызывает сервлет `HiddenServlet`, когда пользователь нажимает кнопку **Login**:

```
<HTML>
<TITLE>ONLINE SHOPPING PORTAL</TITLE>
<BODY>
      <FORM ACTION =
"http://localhost:8080/hidden_field/servlet/HiddenServlet" METHOD = POST
align=CENTER>

      Username: <INPUT TYPE = TEXT NAME = "user" align=CENTER><BR>
            <INPUT TYPE = SUBMIT VALUE = "Login" align=CENTER>
    </FORM>
</BODY>
</HTML>
```

Рис. 60 показывает результат работы файла `Login.html` с именем пользователя **William**, отображаемым в текстовом поле **Username**:

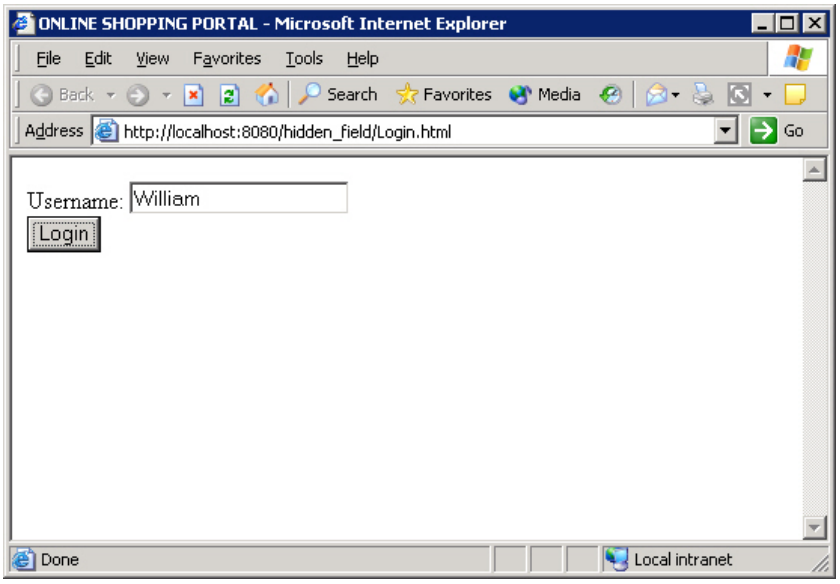


Рис. 60. Результат вывода Login.html

Когда пользователь нажимает кнопку **Login**, информация от пользователя будет отправлена сервлету `HiddenServlet`. Затем сервлет получит имя пользователя и сгенерирует форму HTML со скрытым полем формы и кнопкой **Submit**. Скрытое поле формы содержит имя пользователя. Следующий код используется для создания сервлета `HiddenServlet`:

```
      /* Импорт требуемых пакетов. */
      import java.io.*;
      import javax.servlet.*;
      import javax.servlet.http.*;

      /* Определение сервлета HiddenServlet, который расширяет HttpServ-
let. */
      public class HiddenServlet extends HttpServlet
      {
      /* Переопределение метода doGet() класса HttpServlet. */
            public void doGet(HttpServletRequest req, HttpServletRe-
sponse res) throws ServletException, IOException
            {
      /* Вызов метода doPost() класса HttpServlet. */
                  doPost(req, res);
            }
      /* Переопределение метода doPost() класса HttpServlet для реализа-
ции функциональности сервлета. */

            public void doPost(HttpServletRequest req, HttpServletRe-
sponse res) throws ServletException, IOException
            {
                  /* Получение параметров, связанных с пользователем (па-
роль и регистрационное имя) из объекта запроса. */
                  String username = req.getParameter("user");
                  PrintWriter pw = res.getWriter();


                  pw.println("Hello! click Submit to proceed");
```

```
                pw.println("<Form name=\"login\" ac-
tion=\"http://localhost:8080/hidden_field/servlet/SecondServlet\">");
      /* Добавление скрытого поля. */
                pw.println("<input type=\"hidden\" name=\"user\" val-
ue=" + username+">");
                pw.println("<input type=\"Submit\" val-
ue=\"Submit\"></form>");
           }
      }
```

Рис. 61 показывает форму HTML, сгенерированную сервлетом HiddenServlet:

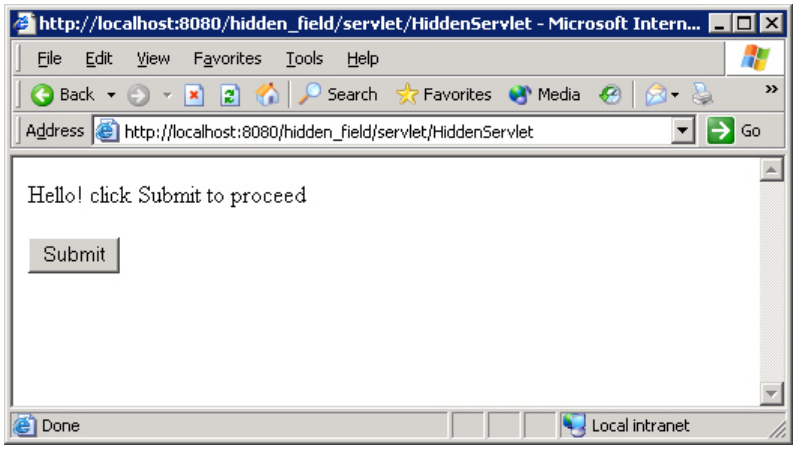


Рис. 61. Результат работы HiddenServlet

Когда пользователь нажимает кнопку **Submit**, значение, хранимое в скрытом поле формы, передается сервлету `SecondServlet`. Сервлет `SecondServlet` получает имя пользователя в скрытом поле формы, и выводит сообщение с приветствием в браузер клиента. Следующий коиспользуется для создания сервлета `SecondServlet`:

```
      /* Импорт необходимых пакетов. */
      import java.io.*;
      import javax.servlet.*;
      import javax.servlet.http.*;

      /* Определение сервлета SecondServlet, который расширяет HttpServ-
let. */
      public class SecondServlet extends HttpServlet
      {
      /* Переопределение метода doGet() в HttpServlet. */
            public void doGet(HttpServletRequest req, HttpServletRe-
sponse res) throws ServletException, IOException
            {
                  doPost(req, res);
            }

            public void doPost(HttpServletRequest req, HttpServletRe-
sponse res) throws ServletException, IOException
            {
                  /* Получение параметров, связанных с именем пользовате-
ля в скрытом поле формы из объекта запроса. */
                  String uname = req.getParameter("user");
                  PrintWriter pw = res.getWriter();
```

```
                pw.println("Hello! "+uname);
        }
}
```

Рис. 62 демонстрирует результат работы `SecondServlet`, который приветствует пользователя по имени:

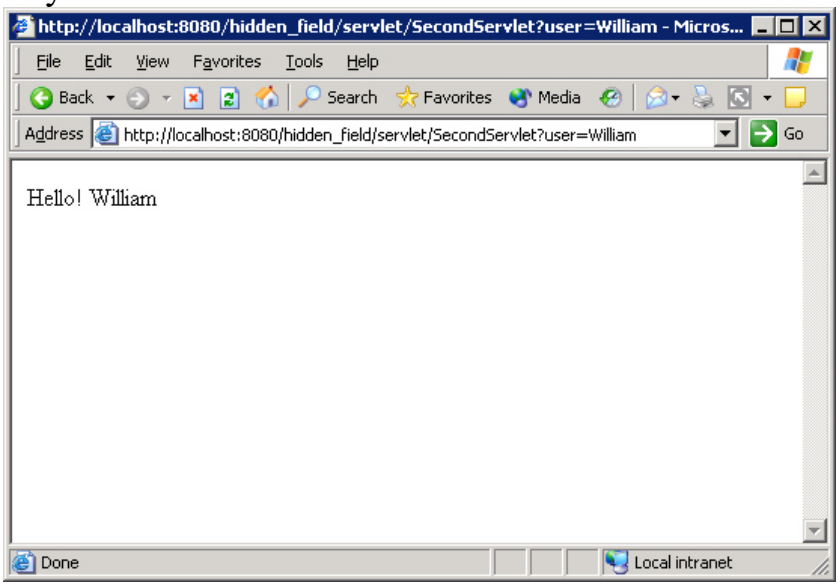


Рис. 62. Результат работы SecondServlet

Перезапись URL

Перезапись URL – это прием управления пользовательскис сеансом, модифицируя URL. Обычно этот прием используется, когда передаваемая информация не очень важна, потому что URL может быть легко перехвачен при передаче. Например, в онлайновом магазине-портале сервлет может модифицировать URL для включения в него информации о пользователе. Затем сервлет может отобразить URL и когда пользователь щелкает по гиперссылке с этим URL, информация отправляется другому сервлету, который получает информацию о пользователе и выводит сообщение с приветствием. Следующий код используется для создания сервлета `RewriteServletURL`, который модифицирует и отображает URL.

```
        /* Импорт требуемых пакетов. */
        import java.io.*;
        import javax.servlet.*;
        import javax.servlet.http.*;

        public class RewriteServletURL extends HttpServlet
        {
                public void doGet(HttpServletRequest req, HttpServletRe-
sponse res) throws ServletException, IOException
                {
                        doPost(req, res);
                }
```

```
            public void doPost(HttpServletRequest req, HttpServletRe-
sponse res) throws ServletException, IOException
            {
                  /* Получение параметров, связанных с пользователем (па-
роль и регистрационной имя) из объекта запроса. */
                  String username = req.getParameter("user");
                  PrintWriter pw = res.getWriter();

                  /* Проверка статуса login  */
                  res.setContentType("text/html");
                  pw.println("Hello! <a
href=\"http://localhost:8080/rewrite_cntxt/servlet/SecondServlet?uname="
+ username + "\"> click here </a>to proceed");
            }
      }
```

В выше приведенной программе сервлет `RewriteServletURL` читает имя пользователя из клиентского запроса, добавляет имя пользователя к URL и выводит ссылку на другой сервлет, `SecondServlet`. Рис. 63 демонстрирует результат работы сервлета `RewriteServletURL`:

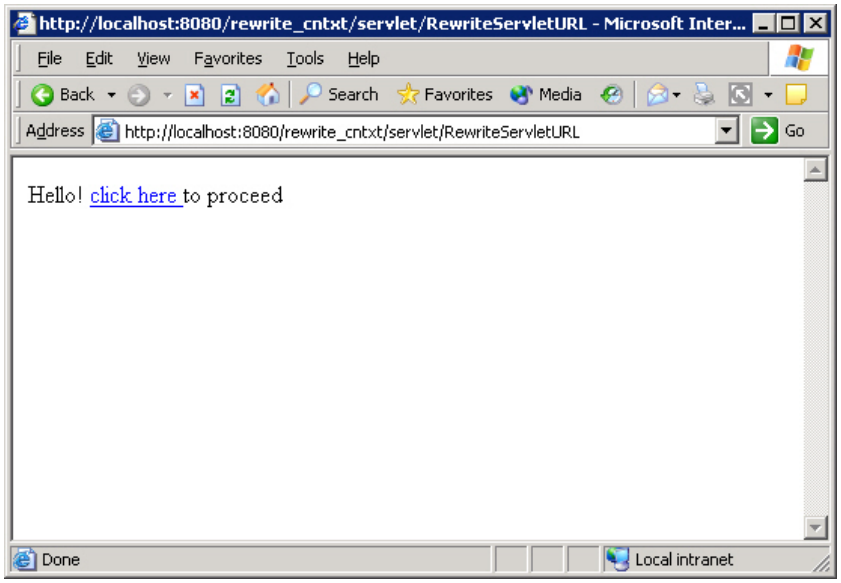


Рис. 63. Результат работы RewriteServletURL

Когда пользователь щелкает по гиперссылке, сервлет `SecondServlet` считывает имя пользователя, добавленное к URL, и выводит сообщение с приветствием. Следующий код используется для создания сервлета `SecondServlet`:

```
      /* Импорт требуемых пакетов. */
      import java.io.*;
      import javax.servlet.*;
      import javax.servlet.http.*;

      /* Define the SecondServlet that extends HttpServlet. */
      public class SecondServlet extends HttpServlet
      {
```

```
      /* Переопределение метода doGet() в HttpServlet. */
            public void doGet(HttpServletRequest req, HttpServletRe-
sponse res) throws ServletException, IOException
            {
                  doPost(req, res);
            }
            public void doPost(HttpServletRequest req, HttpServletRe-
sponse res) throws ServletException, IOException
            {
                  /* Получение параметров, связанных с именем пользовате-
ля в скрытом поле формы из объекта запроса. */
                  String uname = req.getParameter("uname");
                  PrintWriter pw = res.getWriter();

                  pw.println("Hello! "+uname);
            }
      }
```

В приведенном коде сервлет получает имя пользователя из запроса и выводит сообщение с приветствием клиенту. Рис. 64 показывает результат работы сервлета `SecondServlet`:

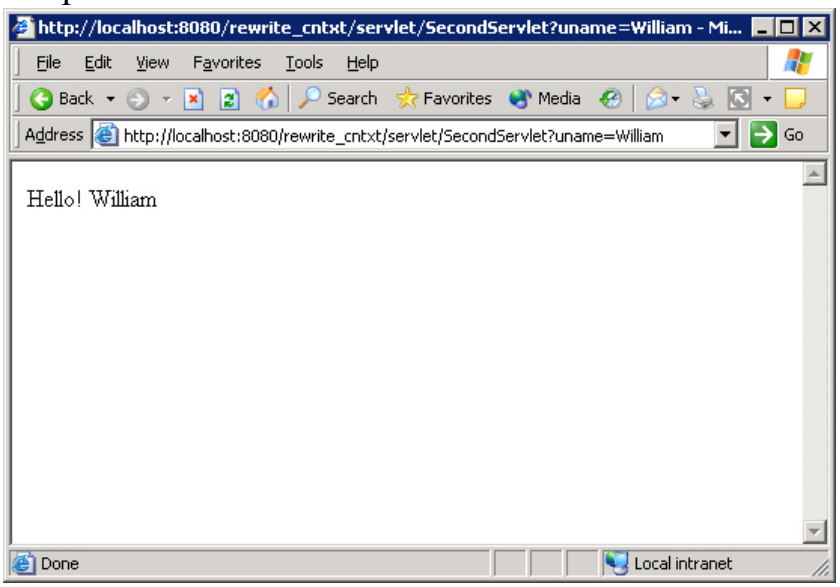


Рис. 64. Результат работы SecondServlet

Использование Cookies

Cookies – это маленькие текстовые файлы, которые сохраняются сервером приложений в браузере клиента для отслеживания всех пользователей. Cookie хранит значения в форме пар имя/значение. Например, cookie может иметь имя user со значением Michael. Они создаются сервером и отправляются клиенту в заголовках ответа HTTP. Клиент сохраняет cookies на локальном жестком диске и отправляет их вместе с заголовками запросов HTTP серверу. Веб-браузер обычно поддерживает 20 cookies для одного пользователя, а размер каждого cookie может быть максимум 4 байта. Ниже представлены различные характеристики cookies:

- Cookies могут быть прочитаны только сервером приложений, который записал их на броузер клиента. Например, если cookie создается сервером приложений `www.macromedia.com` и отправляется клиенту или броузеру, cookie может быть отправлен обратно только этому же серверу.
- Cookies могут быть использованы сервером для нахождения имени компьютера, IP-адреса или любых других характеристик компьютера клиента, получив удаленный адрес компьютера клиента, где cookies хранятся.
- Servlet API предоставляет поддержку для cookies. Класс `Cookie` пакета `javax.servlet.http` представляет cookie. Класс `Cookie` предоставляет конструктор, который принимает имя и значение для создания cookie. Следующий фрагмент кода демонстрирует конструктор для создания cookie с именем `name` и значением `value`:

```
public Cookie(String name, String value);
```

- Интерфейс `HttpServletResponse` предоставляет метод `addCookie()` для добавления cookie к объекту ответа, а затем отправки его клиенту. Следующий фрагмент кода показывает, как добавить cookie:

```
resp.addCookie(Cookie cookie);
```

- Класс `HttpServletRequest` предоставляет метод `getCookies()`, который возвращает массив cookies, который содержится в объекте запроса. Следующий фрагмент кода показывает, как получить cookies:

```
Cookie cookie[] = req.getCookies();
```

Класс `Cookie` предоставляет несколько методов для установки и получения различных свойств cookie. Табл 21 описывает различные методы класса `Cookie`:

Таблица 21

Метод	***Описание***
`public String getName()`	Возвращает имя cookie.
`public void setMaxAge(int expiry)`	Устанавливает максимальное время, которое броузер клиента сохраняет значение cookie.
`public int getMaxAge()`	Возвращает максимальный возраст cookie в секундах.
`public void setValue(String value)`	Устанавливает новое значение для cookie.
`public String getValue()`	Возвращает значение cookie.

Рассмотрим приложение онлайнового магазина, которое отслеживает пользователя с помощью cookies. Когда пользователь использует приложение первый раз, сервлет приложения получает имя пользователя, сохраняет его в cookie и записывает его в браузере клиента. При следующем запросе от клиента другие сервлеты могут получить имя пользователя, хранимое в этом cookie, чтобы идентифицировать клиента.

Следующий код используется для создания сервлета `CookieServlet`, который получает имя пользователя и сохраняет его в cookie и затем отправляет cookie клиенту, используя объект `response`:

```
/* Импорт требуемых пакетов. */
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
/* Define the CookieServlet servlet that extends HttpServlet. */

public class CookieServlet extends HttpServlet
{
      /* Переопределение метода doGet() класса HttpServlet. */
      PrintWriter pw=null;
      public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException
      {
      /* Вызов метода doPost() класса HttpServlet. */
            doPost(req,res);
      }
      /* Переопределение метода doPost() класса HttpServlet, который
реализует функциональность сервлета. */
      public void doPost(HttpServletRequest req, HttpServletResponse
res) throws ServletException, IOException
      {
            pw=res.getWriter();
      /* Получение информации от пользователя из запроса HttpServlet. */
            String username = req.getParameter("user");
            String password = req.getParameter("password");

      /* Создание cookie, которое содержит имя пользователя.  */
                  Cookie ck = new Cookie("user", username);
      /*Добавление cookie к броузеру клиента. */
                  res.addCookie(ck);

                  pw.println("Cookie containing user name is stored in
your browser.");
      }
}
```

В приведенном коде `CookieServlet` создает экземпляр класса `Cookie` и сохраняет информацию о пользователе, полученную от интерфейса `HttpServletRequest`. Рис. 65 показывает результат работы сервлета `CookieServlet`:

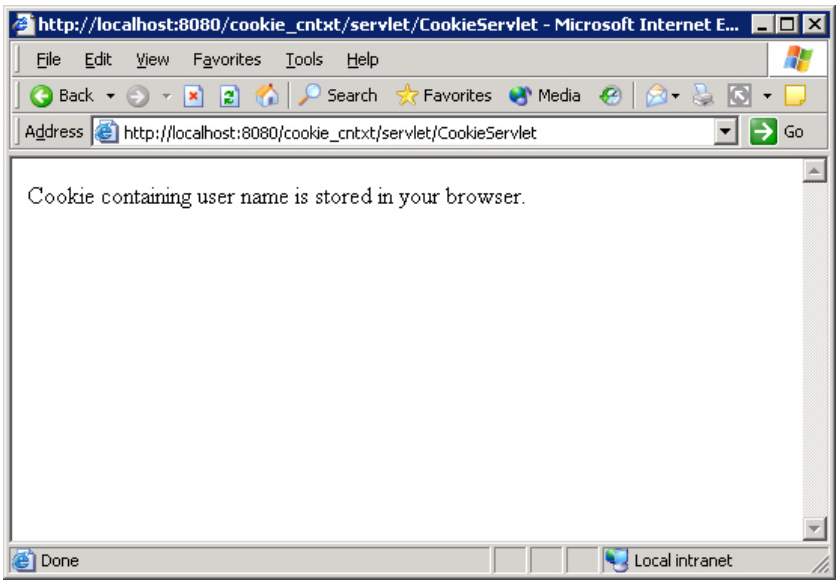


Рис. 65. Результат работы CookieServlet

Когда клиент отправляет запрос приложению, cookie отправляется вместе с запросом. Следующий код используется для создания сервлета `RetrieveCookie`, который получает cookie из объекта запроса и извлекает из него имя пользователя:

```
/* Импорт необходимых пакетов. */
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

/* Определение класса RetrieveCookie, который расширяет HttpServlet. */
public class RetrieveCookie extends HttpServlet
{
      /* Переопределение метода doGet() класса HttpServlet. */
      PrintWriter pw=null;

      public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException
      {
      /* Вызов метода doPost() класса HttpServlet. */
            doPost(req,res);
      }
      /* Переопределение метода doPost() класса HttpServlet, который
реализует функциональность сервлета. */
      public void doPost(HttpServletRequest req, HttpServletResponse
res) throws ServletException, IOException
      {
      /* Получение экземпляра класса PrintWriter. */
                  pw=res.getWriter();
                  String username=null;
      /* Получение cookies, если таковые есть, хранимых в броузере кли-
ента.*/
                  Cookie ck[] = req.getCookies();

            if (ck!=null)
            {
```

```
                    for (int i=0; i<ck.length; i++)
                    {
                          if (ck[i].getName().equals("user"))
                    /*  Получение имени пользователя из cookie. */
                          username = ck[i].getValue();

                    }
                    pw.println(" Hello!   "+username);
              }
              else
              {
                    pw.println("No cookies found");
              }
        }
}
```

В приведенном коде сервлет `RetrieveCookie` получает cookies, хранимые у клиента, используя метод `getCookies()` интерфейса `HttpServletRequest`, и выводит полученное имя пользователя в браузере. Рис. 66 представляет информацию о пользователе, полученную из cookie:

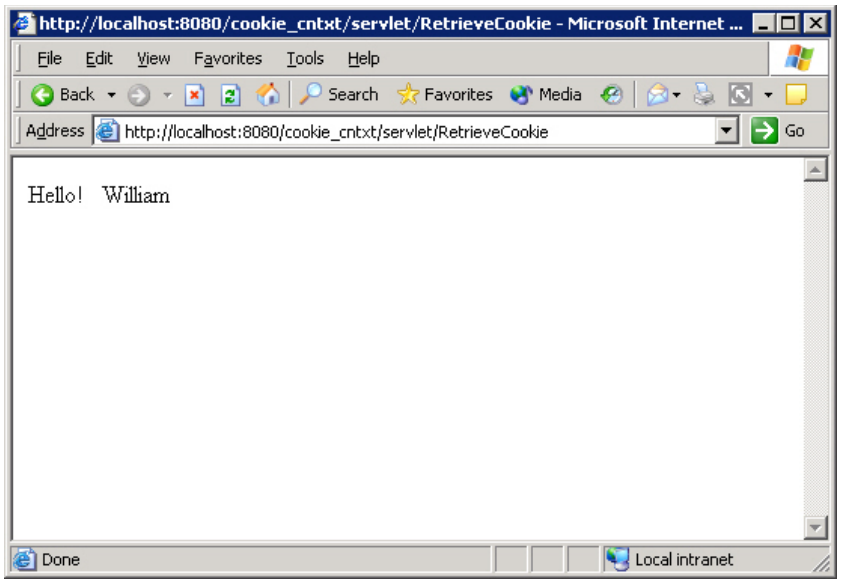


Рис. 66. Результат работы RetrieveCookie

Сервер приложений может сохранять cookies на машине клиента, только тогда, когда cookies разрешены в браузере клиента.

Servlet Session API

Можно использовать классы и интерфейсы, определенные в Servlet Session API для создания и управления пользовательскими сеансами. Ниже перечислены различные интерфейсы, предоставляемые Servlet Session API для создания и управления сеансом пользователя: `javax.servlet.http.HttpSession`,

`javax.servlet.http.HttpSessionListener` и `javax.servlet.http.HttpSessionBindingListener`.

Интерфейс `javax.servlet.http.HttpSession` предоставляет методы для управления сеансом пользователя. Можно создать объект интерфейса `HttpSession` для сохранения информации сеанса как пару имя/значение. Позднее можно извлечь эту информацию для управления сеансами пользователей.

Табл. 22 описывает различные методы, определенные в интерфейсе `HttpSession`:

Таблица 22

Метод	***Описание***
`public void setAttribute(String name, Object value)`	Добавляет атрибут с уникальным именем в объект сеанса и сохраняет пару name/value в текущем сеансе. Если уже есть объект с таким атрибутом, то новый объект замещает существующий.
`public getAttribute(String name)`	Извлекает объект, связанный с именем атрибута, указанным в методе, из объекта сеанса. Если объект с указанным атрибутом не найден, то метод `getAttribute()` возвращает null.
`public Enumeration getAttributeNames()`	Возвращает имена всех объектов, которые находятся в объекте сеанса.
`public void removeAttribute(String name)`	Удаляет атрибут с именем, указанным в методе, из объекта сеанса.
`public void setMaxInactiveInterval(int interval)`	Устанавливает максимальное время, на которое сеанс будет оставаться активным. Время задается в секундах. Если в течение этого времени нет запросов от клиента, то сервер уничтожает сеанс. Отрицательное значение в этом методе означает, что сеанс должен всегда оставаться активным.
`public int getMaxInactiveInterval()`	Возвращает максимальное время в секундах, в течение которого сервер не уничтожит сеанс, даже если нет клиентского запроса.
`public String getId()`	Возвращает строку, которая содержит уникальный идентификатор, связанный с сеансом.

Метод	Описание
`public void invalidate()`	Уничтожает сеанс. Все объекты, связанные с сеансом, автоматически освобождаются.

Следующий код используется для создания сервлета `SessionServlet`, который показывает, как использовать различные методы интерфейса `HttpSession` используются для создания сеанса пользователя, записи информации пользователя в объект сеанса и получения этой информации из объекта сеанса:

```
/* Импорт требуемых пакетов. */
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/* Определение SessionServlet, который расширяет HttpServlet. */
public class SessionServlet extends HttpServlet
{
/* Переопределение метода doGet() в HttpServlet. */
      public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException
      {
      /* вызов метода doPost() в HttpServlet. */
            doPost(req, res);
      }
/* Переопределение метода doPost() в HttpServlet. */
      public void doPost(HttpServletRequest req, HttpServletResponse
res) throws ServletException, IOException
      {
/* Получение имени пользователя из HttpServletRequest.*/
            String username = req.getParameter("user");
/* Получение экземпляра PrintWriter. */
            PrintWriter pw = res.getWriter();
/* Создание объекта HttpSession и проверка, существует ли уже сеанс для
этого пользователя, вызовом метода getSession() в HttpServletRequest.
Если сеанс не существует, метод getSesssion() создает новый сеанс.*/
            HttpSession session = req.getSession(true);
/* Установка username как атрибута в сеансе пользователя. */
            session.setAttribute("user", username);
            pw.println("Your user name has been set as an attribute to
the session<br>");
/* Получение атрибута username из сеанса пользователя. */
                        String user=(String) ses-
sion.getAttribute("user");
             pw.println("Your user name retrieved from the session is
"+user);
      }
}
```

В предложенном коде сервлет получает информацию пользователя и проверяет, существует или нет сеанс для этого пользователя. Если сеанс пользователя не существует, сервлет вызывает метод `getSession()` интерфейса `HttpServletRequest` и создает объект

интерфейса `HttpSession` и затем устанавливает username как атрибут в объекте сеанса. Также сервлет извлекает username, хранимое как атрибут в объекте сеанса.

Можно закончить сеанс пользователя, уничтожив его. Можно потребовать удаление сеанса в различных ситуациях, например, когда пользователь выходит из приложения, или, когда пользователь бездействует значительный период времени. Уничтожить сеанс можно, используя различные методы интерфейса `HttpSession`:

- Устанавив максимальное время простоя с помощью метода `setMaxInactiveInterval(int interval)`. Время простоя – это время, в течение которого клиент не производит никаких действий.
- Явно вызвав метод `invalidate()` объекта сеанса. Вы можете использовать следующий код для уничтожения сеанса:

```
HttpSession ss=req.getSession(true);
session.invalidate();
```

- Указав значение таймаута сеанса в минутах в элементе session-timeout дескриптора развертывания.

Следующий фрагмент кода демонстрирует элемент `<session-timeout>`, который можно включить в дескриптор развертывания для установки опции уничтожения сеанса:

```
<session-config>
  <session-timeout>
30
  </session-timeout>
</session-config>
```

Обработка ошибок и исключений в сервлетах

Сервер приложений при обслуживании запросов от клиентов может столкнуться с проблемами, такими как: запрашиваемый ресурс не найден, или работающий сервлет генерирует исключение. Например, в онлайновом магазине проблема возникает, если пользователь выбирает товар, который временно не доступен, и пытается добавить его в магазинную тележку. В этом случае отображается сообщение об ошибке в окне браузера. Можно выполнить обработку исключений в сервлете, чтобы обработать исключения, генерируемые сервлетом. Также можно создать настраиваемые страницы ошибок для отображения исключений и сообщений об ошибках и записи этих сообщений в файл журнала сервера приложений.

Исключения в сервлетах

Servlet API определяет два класса исключений, которые определены в пакете `javax.servlet`. Это `ServletException` и `UnavailableException`. Класс `UnavailableException` является подклассом класса `ServletException`.

Класс javax.servlet.ServletException

Класс `javax.servlet.ServletException` определен в пакете `javax.servlet`, и является подклассом класса `java.lang.Exception`. Он определяет исключение в сервлете, которое сервлет генерирует во время обработки запроса от клиента. Этот класс содержит метод `getRootCause()`, который возвращает объект типа `java.lang.Throwable`, который представляет основную причину исключения.

Класс javax.servlet.UnavailableException

Класс исключения `javax.servlet.UnavailableException` генерируется сервлетом, когда сервлет временно или постоянно недоступен. Ниже представлены методы, предоставляемые классом `UnavailableException`:

- `public boolean isPermanent()`: Возвращает булево значение, которое описывает постоянно ли сервлет недоступен или нет. Если сервлет недоступен постоянно, то этот метод возвращает `true`.
- `public int getUnavailableSeconds()`: Возвращает время в секундах, в течение которого сервлет будет оставаться недоступным.

Обработка ошибок

Сервлеты в веб-приложениях могут генерировать ошибки и исключения. При эхтом можно предоставлять информацию пользователям относительно этих ошибок и исключений, отправляя сообщения об ошибках и коды статуса, которые представляют различные типы ошибок. Также можно создавать настраиваемые страницы ошибок для форматирования и вывода информации о типе ошибки и возникающих исключениях и/или записывать сообщения в файл журнала сервера.

Сообщения об ошибках и коды статуса

Код статуса – это информация, которую сервер приложений отправляет клиенту о том, что запрос клиента был удачно или неудачно обработан. Код статуса возвращается браузеру клиента в объекте ответа. Класс HttpServletResponse пакета javax.servlet.http определяет различные поля, которые представляют коды статуса, которые можно использовать для отправки кодов статуса из сервлета. Коды статуса логически сгруппированы в следующие пять категорий:

- `Информация:` Группа информационных кодов представляет сообщения о приеме запроса, и что сервер приложений обрабатывет запрос. Ниже представлены некоторые из полей HttpServletResponse, которые представляют информационные коды:
  - `SC_SWITCHING_PROTOCOLS`: Представляет код статуса 101 и указывает на переключение протоколов сервером приложений.
  - `SC_CONTINUE`: Представляет код статуса 100 и указывает, что клиент может продолжать взаимодействие с сервером приложений.
- `Успех:` Группа кодов успеха представляет сообщение об успехе, которое указывает на то, что запрос удачно выполнен. Ниже представлены некоторые из полей HttpServletResponse, которые представляют коды успеха:
  - `SC_OK`: Представляет код статуса 200 и указывает, что запрос успешно принят.
  - `SC_ACCEPTED`: Представляет код статуса 202 и указывает, что сервер принял запрос и обрабатывает его.
- `Перенаправление:` Группа кодов перенаправления указывает, что запрос перенаправлен на другую страницу для обработки и обслуживания. Ниже представлены некоторые из полей HttpServletResponse, которые представляют коды перенаправления:
  - `SC_MOVED_PERMANENTLY`: Представляет код статуса 301 и указывает, что ресурс удален на длительное время.
  - `SC_MOVED_TEMPORARILY`: Представляет код статуса 302 и указывает, что ресурс временно удален.
- `Ошибка клиента:` Группа кодов ошибки клиента указывает, что запрос имеет некоторую ошибку и не может быть обслужен. Ниже представлены некоторые из полей

HttpServletResponse, которые представляют коды ошибки клиента:

- SC_BAD_REQUEST: Представляет код статуса 400 и указывает, что синтаксис клиентского запроса некорректен.
- SC_NOT_FOUND: Представляет код статуса 404 и указывает, что ресурс. требуемый клиентом, недоступен.
- SC_GONE: Представляет код статуса 410 и указывает, что ресурс больше недоступен.

- Ошибка сервера: Группа ошибок сервера указывает, что сервер не в состоянии выполнить запрос клиента. Ниже представлены некоторые из полей HttpServletResponse, которые представляют коды ошибки сервера:
  - SC_INTERNAL_SERVER_ERROR: Представляет код статуса 500 и указывает, что на сервере имеет место ошибка, которая препятствует выполнению запроса.
  - SC_NOT_IMPLEMENTED: Представляет код статуса 501 и указывает, что сервер не предоставляет функциональность для выполнения запроса.

Вы можете отправить ответ с ошибкой или со статусом клиенту, независимо от того, где возникла ошибка при работе веб-приложения. sendError() и setStatus() – два метода объекта HttpServletResponse, которые можно использовать для отправления сообщения с ошибкой и статусом клиенту.

Метод sendError()

Метод sendError() определен в интерфейсе HttpServletResponse. Вы можете использовать этот метод для отправления сообщения об ошибке клиенту. Табл. 23 описывает сигнатуру переопределяемого метода sendError():

Таблица 23

Метод	*Описание*
public void sendError(int status)	Принимает поле целого типа HttpServletResponse как аргумент, который описывает тип возникшей ошибки, и отправляет ответ с ошибкой клиенту.
public void sendError(int status, String message)	Принимает дополнительный аргумент String, который описывает ошибку, и отправляет ответ с ошибкой клиенту.

Метод sendError() генерирует исключение IllegalStateException, если он вызывается после получения ответа.

Следующий фрагмент кода используется для отправки сообщения об ошибке клиенту:

```
public void doPost (HttpServletRequest request, HttpServletResponse re-
sponse)
{
PrintWriter pw=null;
      try
      {
      pw= response.getWriter ();
      pw.println ("This page is going to give an Error");
response.sendError (HttpServletResponse.SC_NOT_FOUND, "Sorry the source
you have requested is not available.");
/* Не пытайтесь записывать в буфер ответа после вызова метода
sendError().*/
}
catch(Exception ex)
{
      pw.println(ex.printStackTrace());
}
}
```

В методе `doPost()` вышеуказанного фрагмента кода метод `sendError()` вызывается для отправки кода статуса `SC_NOT_FOUND` с сообщением об ошибке.

Метод setStatus()

Метод `setStatus()`, определенный в интерфейсе `HttpServletResponse`, устанавливает статусную информацию о сервлете. Табл. 24 описывает сигнатуры переопределяемого метода `setStatus()`:

Таблица 24

Метод	***Описание***
public void setStatus (int status)	Принимает целочисленное поле `HttpServletResponse` как аргумент, который представляет статусную информацию, и устанавливает информацию для ответа.
public void setStatus (int status, String message)	Принимает дополнительный строковый аргумент, который описывает статус сервлета.

Метод `setStatus()` генерирует исключение `IllegalStateException`, если он вызывается после того, как ответ принят. Следующий фрагмент кода применяется для выполнения метода `setStatus()`, чтобы отправить ответ с ошибкой клиенту:

```
public void doPost (HttpServletRequest request, HttpServletResponse re-
sponse)
{
      PrintWriter pw =null;
try
{
pw=response.getWriter();
      pw.println("This page is going to display a status code");
response.setStatus (HttpServletResponse.SC_GONE);
// Вы можете писать в ответ после вызова setStatus ().
      }
      catch(Exception ex)
      {
      pw.println(ex.printStackTrace();
      }
}
```

В методе `doPost()` предложенного фрагмента кода код статуса `SC_GONE` записывается в объект ответа.

Настройка страницы ошибок

Сервер приложений предоставляет страницы ошибок для вывода сообщений с исключениями и кодами статуса. Эти страницы ошибок ненаглядны, и с их помощью трудно понять причину ошибки. Чтобы решить эту проблему, можно создать собственную страницу ошибок в вашем веб-приложении для вывода сообщений с исключениями и ошибками. Вам нужно отобразить исключения на страницу ошибок во время развертывания приложения. Например, вы можете отобразить исключение типа `java.lang.ArithmeticException` на страницу ошибок. Где бы исключение типа `ArithmeticException` не было сгенерировано приложением, веб-контейнер выведет настраиваемую страницу ошибок.

Веб-контейнер использует три поля для отправки информации об исключении, перехватываемом страницей ошибок. Например, чтобы отправить информацию на страницу ошибок, соотнесенную с исключением `ArithmeticException`, сервер устанавливает поля `javax.servlet.error.status_code`, `javax.servlet.error.message` и `javax.servlet.error.exception_type` как атрибут в объекте запроса и передает объект запроса странице ошибок.

Можно извлечь поля из объекта запроса в странице ошибок и соответственно сгенерировать ответ, который описывает информацию об исключении.

Следующий фрагмент кода демонстрирует элемент `<error-page>` дескриптора развертывания:

```
<error-page>
<exception-type>java.lang.NumberFormatException </exception-type>
<location>/servlet/ErrorServlet</location>
</error-page>
```

Запись исключений и ошибок в файл серверного журнала

Вы можете записывать сообщения об ошибках и исключения в файл серверного журнала, чтобы отслеживать тип ошибок, генерируемых приложением. Запись ошибок и исключений помогает определить ошибки, которые возникают в вашем веб-приложении. Интерфейс `ServletContext` предоставляет метод `log()` для записи сообщений об ошибках и исключений. Можно использовать следующий код для создания сервлета, который генерирует исключение и записывает данные об исключении в файл серверного журнала:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io. *;

public class LogDemo extends HttpServlet
{
      public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException
      {
            ServletContext sc = getServletContext();
            PrintWriter pw = response.getWriter();
            pw.println ("<B>Writing an exception to the server log
file...</B>");
            try
            {
                  int x = 9 ;
                  int y = x/0 ;
            }
            catch(ArithmeticException ae)
            {
                  sc.log("Dividing an integer by zero will not give the
result", ae);
            }
      }
}
```

В предложенном коде сервлет пытается разделить число на 0. В результате генерируется исключение типа `ArithmeticException`. Сервлет получает объект интерфейса `ServletContext` и вызывает метод `log()` для записи исключения в файл журнала сервера.

Рис. 67 показывает содержимое файла `server.log` с информацией об исключении `ArithmeticException`:

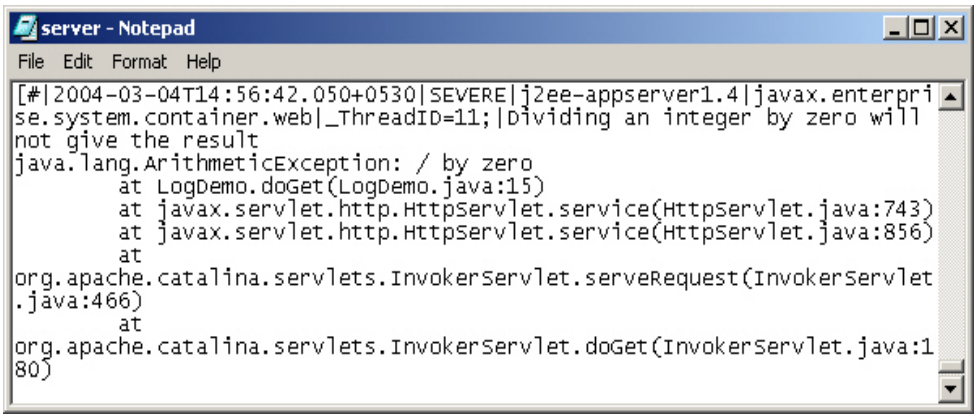


Рис. 67. Содержимое исключения в файле server.log

Взаимодействие сервлетов

В веб-приложении различные сервлеты могут потребовать взаимодействия друг с другом для обработки клиентских запросов. Также сервлет в веб-приложении может передавать запрос другому сервлету, если возникло некоторое исключение во время обработки запроса. Вы можете реализовать приемы взаимодействия сервлетов в веб-приложении следующими способами:

1. Используя диспестчер запросов
2. Используя объект запроса сервлета

Диспетчер запросов

Диспетчер запросов – это объект интерфейса `javax.servlet.RequestDispatcher`, который делает возможным взаимодействие сервлетов. Можно использовать диспетчер запросов в вашем сервлете для передачи запроса другому ресурсу, который может быть статической HTML-страницей или сервлетом. Также можно использовать объект `RequestDispatcher` для включения содержимого другого ресурса в ваш сервлет. Интерфейс `ServletContext` предоставляет метод `getRequestDispatcher(String path)`, который возвращает объект `RequestDispatcher`. Аргумент path этого метода задает путь относительно корня контекста целевого ресурса. Если вы получили объект `RequestDispatcher`, можнр выполнить следующие две функции:

1. Включение содержимого другого сервлета
2. Передача запроса другому сервлету

Включение содержимого другого сервлета

Интерфейс RequestDispatcher предоставляет метод include(), который можно использовать для включения содержимого другого сервлета. Для этого необходимо сначала получить объект интерфейса RequestDispatcher, а затем вызвать метод include(). Например, рассмотрим сервлет CopyrightServlet, который выводит информацию о правопользовании ABC Inc. на вебсайте этой организации.

Можно использовать следующий код для разработки сервлета CopyrightServlet:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

/* Этот сервлет печатает информацию о правопользовании.*/
public class CopyrightServlet extends HttpServlet
{
      public void doGet(HttpServletRequest request, HttpServletResponse
response)   throws ServletException, IOException
      {
            PrintWriter pw = response.getWriter();
            pw.println("Copyright  2000-2004 ABC, Inc. All Rights Re-
served.<BR>");
      }
}
```

Чтобы включить содержимое CopyrightServlet в сервлет IncludeServlet, сначала необходимо вызвать метод getRequestDispatcher() интерфейса ServletContext, передав путь к CopyrightServlet как параметр. Это вернет объект RequestDispatcher. Затем можно вызвать метод include() для включения содержимого CopyrightServlet. Вы можете использовать следующий код для создания сервлета IncludeServlet:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class IncludeServlet extends HttpServlet
{
      public void doGet(HttpServletRequest request,HttpServletResponse
response)throws ServletException, IOException
      {
            /*Получение объекта RequestDispatcher */
            RequestDispatcher dispatch = getServletCon-
text().getRequestDispatcher("/servlet/CopyrightServlet");
            PrintWriter pw = response.getWriter();
            pw.println("<B> The copyright information included from cop-
yright servlet: </B><BR>");
/*использование метода include() в RequestDispatcher для включения со-
держимого*/
```

```
            dispatch.include(request, response);
      }
}
```

При запуске сервлета `IncludeServlet` в вашем браузере сервлет выведет его содержимое вместе с содержимым `CopyrightServlet`.

Передача запроса другим сервлетам

Объект `RequestDispatcher` используется для передачи запросов другим сервлетам приложении. Например, рассмотрим компанию ABC, Inc., которая прелагает туристические поездки на время отпуска. Вам нужно создать веб-сайт ABC, Inc., где пользователи могут получить информацию, такую как доступные авиарейсы, транзитные аэропорты и номера в гостиницах. Можно создать главный сервлет для обработки всех этих запросов и отправлять назад ответы. Однако это увеличит время обработки запроса главного сервлета. Вместо этого можно разработать различные сервлеты для обработки определенных клиентских запросов. Например, сервлет `HotelInformation` может обрабатывать запросы на гостиничные номера, а сервлет `FlightInformation` может обрабатывать запросы, относящиеся к информации о перелетах. Главный сервлет веб-приложения может получать все запросы пользователя и использовать объект `RequestDispatcher` для передачи запроса соответствующему сервлету. Следующий фрагмент кода демонстрирует, как можно использовать `RequestDispatcher` в методе `doGet()` для передачи запросов различным сервлетам приложения:

```
      public void doGet(HttpServletRequest request,HttpServletResponse
response) throws ServletException, IOException
      {
String requestType=request.getParameter("Type");
if(requestType.equals("hotel"))
{
/*Получение объекта RequestDispatcher */
            RequestDispatcher dispatch = getServletCon-
text().getRequestDispatcher("/servlet/HotelInformation");
/*Использование метода forward() объекта RequestDispatcher для передачи
запроса*/
            dispatch.forward(request, response);
}
if(requestType.equals("cab"))
{
/*Получение объекта RequestDispatcher */
            RequestDispatcher dispatch = getServletCon-
text().getRequestDispatcher("/servlet/CabInformation");

/*Использование метода forward() объекта RequestDispatcher для передачи
запроса*/
            dispatch.forward(request, response);
```

```
}
if(requestType.equals("flight"))
{

/*Получение объекта RequestDispatcher */
            RequestDispatcher dispatch = getServletCon-
text().getRequestDispatcher("/servlet/FlightInformation");
/* Использование метода forward() объекта RequestDispatcher для передачи
запроса */
            dispatch.forward(request, response);
}
}
```

В предложенном фрагменте программы сервлет проверяет тип запроса, который он получает через объект `RequestDispatcher` и передает его другому сервлету для обработки.

Использование объекта запроса

Мы рассмотрели, как объект `RequestDispatcher` обеспечивает взаимодействие сервлетов с помощью передачи запроса другому сервлету и включения ответа от другого сервлета. Однако объект `RequestDispatcher` не позволяет сервлетам разделять данные. Чтобы сделать данные общими для сервлетов, можно использовать объект запроса сервлета, которые должны взаимодействовать.

Вы можете использовать метод `setAttribute()` интерфейса `javax.servlet.ServletRequest` для записи значений данных в объект запроса. Другие сервлеты приложения могут использовать метод `getAttribute()` интерфейса `javax.servlet.ServletRequest` для получения значений данных.

Например, рассмотрим веб-приложение, в котором `CalculatorServlet` выполняет арифметические вычисления. Другой сервлет веб-приложения будет выводить результат вычислений. Чтобы отправить вычисленный результат второму сервлету, `CalculatorServlet` может выполнять следующие действия:

1. Хранить вычисленные данные как атрибуты в объекте запроса первого сервлета.
2. Передавать запрос второму сервлету, используя `RequestDispatcher`.

Затем второй сервлет получает данные из объекта запроса и выводит результат.

Можно использовать следующий код для создания сервлета `CalculatorServlet`, который складывает два числа, заданных пользователем, и сохраняет результат как атрибут объекта запроса. Затем сервлет передает запрос сервлету `DisplayServlet`:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class CalculatorServlet extends HttpServlet
{
      public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException
      {
                  /*Получение чисел, введенных пользователем в HTML-
форме*/
            int num1=Integer.parseInt(request.getParameter("number1"));
             int num2=Integer.parseInt(request.getParameter("number2"));
                   /*Нахождение суммы чисел, введенных пользователем.
*/
                           int result=num1+num2;
                  /*Запись результата в атрибут объекта запроса*/
                           request.setAttribute("result",new Inte-
ger(result));
                       /*Получение объекта ServletContext*/
                           ServletContext
contx=getServletConfig().getServletContext();
                      /*Получение объекта RequestDispatcher*/
                           RequestDispatcher reqDis-
patcher=contx.getRequestDispatcher("/servlet/DisplayServlet");
                           /*Передача запроса*/
                           reqDispatcher.forward(request,response);
      }
}
```

В приведенном коде сервлет `CalculatorServlet` получает числа, отправленные как параметры запроса, в методе `doGet()`. Затем сервлет складывает числа и сохраняет результат в объекте запроса, используя метод `setAttribute()`. В конце сервлет использует объект `RequestDispatcher` для передачи запроса сервлету `DisplayServlet`.

Можно использовать следующий код для разработки сервлета `DisplayServlet`, который выводит результат вычислений:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class DisplayServlet extends HttpServlet
{
      public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException
      {
                     /*Получение результата, хранимого в объекте запро-
са*/
                  Integer res=(Integer)request.getAttribute("result");

      /*Получение объекта PrintWriter*/
                         PrintWriter pw = response.getWriter();
            /*Отправка ответа для вывода результата*/
                          pw.println("The result of the calculation is:
"+res.toString());
      }
```

```
}
```

В предложенном коде сервлет `DisplayServlet` использует метод `getAttribute()` для получения значения результата, хранимого в объекте запроса. `DisplayServlet` затем отправляет сообщение, выводя результат как ответ.

Можно использовать следующую HTML-страницу `CalculatorPage.html` для получения пользовательских данных и передачи этих данных сервлету `CalculatorServlet`:

```
<html>
<head>
<title>Login</title>
</head>
<body>

      </CENTER>
      <form name = frm method="GET" ac-
tion="http://localhost:8080/intercntx/servlet/CalculatorServlet">
      <p align="center"><font size="5"><center><b>Calculation
Form</b><center></font></p>
      <TABLE ALIGN="center" height="57">
      <TR>
            <TD >
            <b>Enter First Number:</b>
            </TD>
            <TD >
            <input type="text" name="number1" size="20" tabindex="1">
            </TD>
      </TR>
      <TR>
            <TD >
            <b>Enter Second Number:</b>
            </TD>
            <TD >
            <input type="text" name="number2" size="20" tabindex="2">
            </TD>
      <TR align="center">
            <TD colspan=2>
            <input type="Submit" value="  Add  " name="B1" stabin-
dex="3">
            </TD>
      </TR>
      </TABLE>
      </form>
</body>
</HTML>
```

Рис. 68 показывает результат работы указанной выше HTML-страницы, которая принимает два числа:

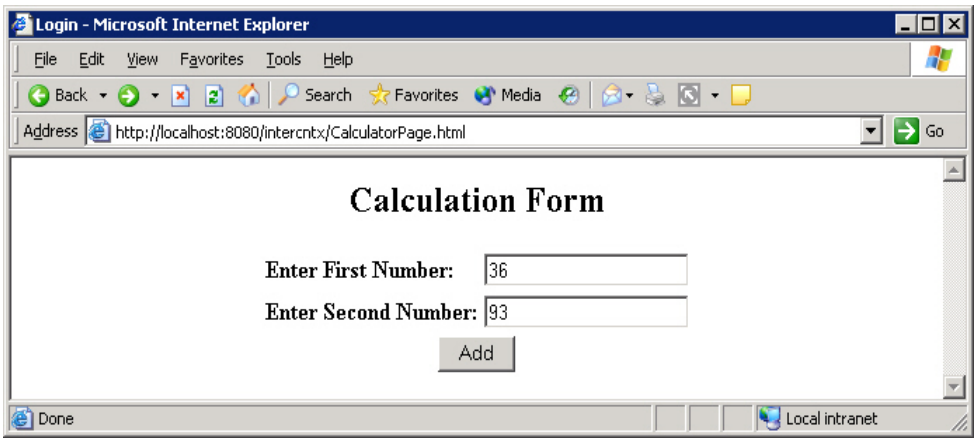


Рис. 68. HTML-страница, которая принимает два числа

После ввода чисел, которые нужно сложить, нажмите кнопку **Add** для вызова `CalculatorServlet`. Сервлет `CalculatorServlet` складывает числа и передает результат сервлету `DisplayServlet`, который выводит результат в броузере клиента. Рис. 69 показывает результат работы `CalculatorServlet`, выведенный сервлетом `DisplayServlet`:

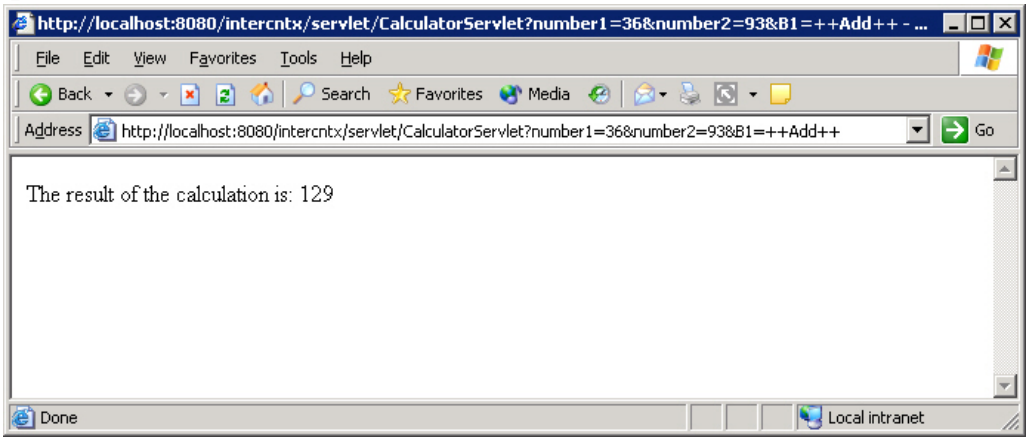


Рис. 69. Результат работы сервлета CalculatorServlet

Потоковая модель сервлета

Спецификация сервлетов определяет две модели потоков, которые устанавливают, как веб-контейнер должен обрабатывать сервлеты в многопоточной среде. Первая модель называется многопоточной моделью, и все сервлеты выполняются по этой модели по умолчанию. В этой модели веб-контейнер запускает новый поток каждый раз, когда клиент отправляет запрос сервлету. Это значит, что одновременно может быть несколько потоков, имеющих доступ к вашему сервлету. Рис. 70 показывает, как различные потоки получают доступ к сервлету, работающему в веб-контейнере многопоточной модели:

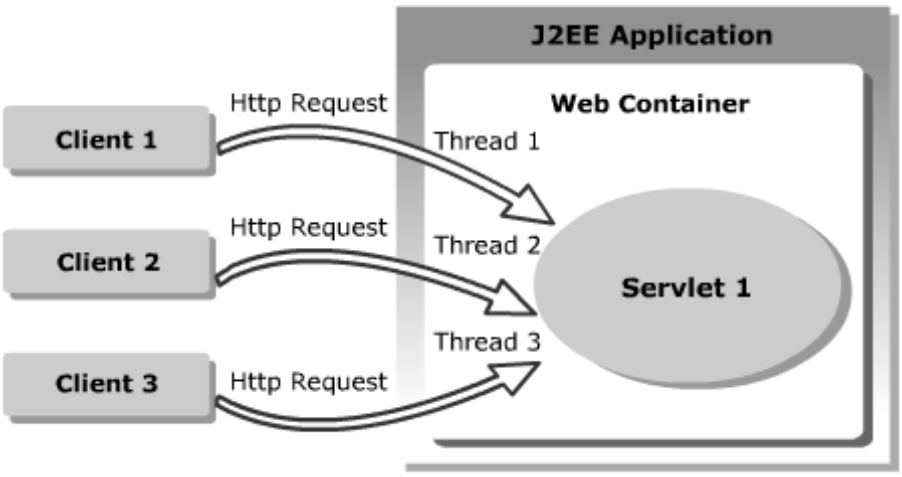


Рис. 70. Многопоточная модель

Во второй модели, называемой однопоточной моделью, веб-контейнер создает пул экземпляров сервлетов и назначает один экземпляр на запрос. Если количество запросов превышает количество экземпляров в пуле, запросы ставятся в очередь. Вы можете указать, чтобы ваш сервлет выполнялся по однопоточной модели, реализовав интерфейс `SingleThreadModel` в вашем сервлете. Реализация интерфейса `SingleThreadModel` пакета `javax.servlet` гарантирует, что только один поток будет выполнятся внутри метода `service()` вашего сервлета. Рис. 71 показывает, как контейнер управляет потоками для сервлетов, которые реализуют интерфейс `SingleThreadModel`:

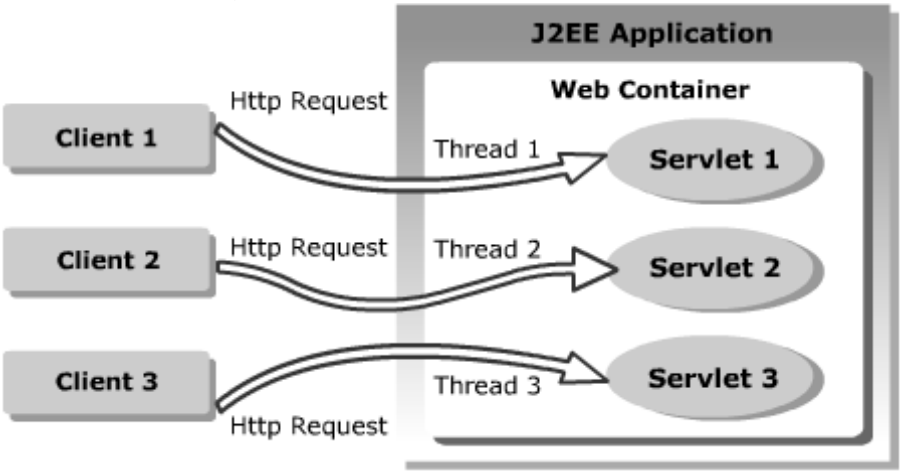


Рис. 71. Однопоточная модель

Реализация интерфейса `SingleThreadModel` в сервлете приведет к понижению производительности сервера. Это происходит из-за того, что серверу нужно создавать отдельные экземпляры для каждого клиентского запроса. К тому же, реализация `SingleThreadModel` не гарантирует синхронизацию доступа к разделяемым ресурсам, таким как переменные классов внутри сервлета.

Безопасная работа с потоками в сервлетах

Необходимо хорошо разбираться в механизме потоков, чтобы предотвратить доступ к разделяемым ресурсам во время разработки сервлетов в многопоточной модели. Чтобы разработать сервлеты, корректно работающие с потоками, сначала нужно определить типы атрибутов, которые по сути безопасны, и типы, которые должны быть защищены для безопасности потоков. Ниже представлены различные характеристики безопасности потоков атрибутов, методов и полей в сервлете:

Методы `init()` и `destroy()`: Эти методы жизненного цикла вызываются только один раз за время жизни экземпляра сервлета. Эти два метода вашего сервлета гарантированно являются безопасными в потоках.

Локальные переменные: Локальные переменные безопасны в потоках. Это из-за того, что существует только отдельный экземпляр локальной переменной для каждого вызова метода, который содержит эту переменную.

Атрибуты запроса: Атрибуты запроса по своему существу безопасны в потоках. Каждый запрос клиента обрабатывается и управляется веб-контейнером, как уникальным запросом. Данные, хранимые в атрибуте объекта запроса одного клиента, недоступны объекту запроса другого клиента.

Атрибуты контекста: Различные потоки могут одновременно получать доступ к атрибутам контекста. Поэтому вы должны обеспечивать синхронизированный доступ к этим атрибутам контекста.

Данные, хранимые в атрибутах сеанса: Различные потоки могут одновременно получать доступ к атрибутам сеанса. Вы должны также синхронизировать доступ к этим атрибутам сеанса.

Чтобы разработать безопасные в потоках сервлеты, вы можете принять во внимание следующие подходы:

Синхронизируйте блок кода, получающий доступ к разделяемому ресурсу. Следующий фрагмент кода демонстрирует, как синхронизировать блок кода, который инкрементирует счетчик в сервлете:

```
public class CounterServlet extends HttpServlet {
  int count = 0;

public void doGet(HttpServletRequest req, HttpServletResponse res)
  throws IOException, ServletException {

...synchronized(this)
 {
  count = count++;
}
```

```
}
```

Синхронизируйте методы, которые получают доступ к разделяемому ресурсу. Следующий фрагмент кода демонстрирует, как можно синхронизировать метод `setCount()`, который инкрементирует счетчик, хранимый в переменной count:

```
public class CounterServlet extends HttpServlet {
  int count = 0;

public void doGet(HttpServletRequest req, HttpServletResponse res)
  throws IOException, ServletException {
    setCount();
...
public synchronized void setCount() {
  count++;
}
```

Также можно синхронизировать методы service() и doXXX() сервлета, чтобы разрешить доступ к ним только одному потоку. Однако в спецификации сервлетов не рекомендуется это делать, так как уменьшается производительность приложения

Фильтры сервлетов

Каждый браузер отображает информацию приложения согласно настройкам в соответствующем окне веб-браузера. Также возможно, что веб-приложение имеет некоторые тэги HTML или особенности, которые веб-браузер клиента не поддерживает. Приложение в этом случае может не заработать или сгенерировать неприемлемый результат в веб-браузере клиента. Чтобы избежать этой проблемы, необходимо определить тип браузера и другую специфичную информацию о клиенте, который отправляет запрос веб-приложению, до вызова сервлета, используя фильтры сервлетов. После определения типа браузера и информации о клиенте, можно настроить вывод приложения так, что его можно будет просматривать в различных браузерах.

Фильтр сервлетов – это объект, который перехватывает запросы и ответы, которые передаются между клиентом и сервлетом. Фильтр может изменять заголовки и содержимое запросов, приходящих от веб-клиента, и передавать их целевому сервлету. Также фильтр может перехватывать и манипулировать заголовками и содержимым ответа, который сервлет отправляет назад. Объекты фильтров сервлетов отличаются от сервлетов тем, что они только преобразуют запросы и ответы для других веб-приложений, а не обрабатывают запросы и не генерируют ответы сами. Фильтры сервлетов разрабатываются как независимые программы, которые могут быть присоединены к различным веб-

приложениям. Рис. 72 показывает, как фильтр перенаправляет запросы и ответы, передаваемые между клиентом и сервлетом:

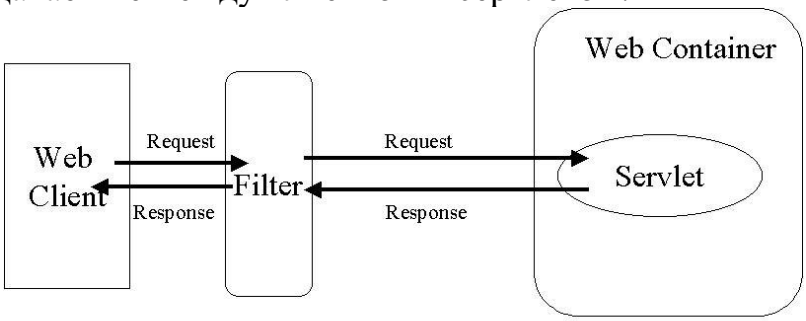


Рис. 72. Фильтр, перенаправляющий запросы и ответы

Преимущества использования фильтров

Фильтры сервлетов контролируют запросы, приходящие сервлету для обработки, и ответы, отправляемые назад сервлетом клиенту. Ниже представлены различные преимущества использования фильтров сервлетов:

Вы можете устанавливать типы запроса, приходящие от веб-клиента, такие как HTTP и FTP, и вызывать сервлет, который должен обрабатывать запрос.

Вы можете проверять клиентов, используя фильтры сервлетов до того, как клиент получит доступ к сервлету.

Вы можете получать информацию о пользователе из параметров запроса для аутентификации пользователя.

Вы можете использовать фильтры сервлетов для получения информации о MIME-типах и других данных заголовка запроса. Затем вы можете использовать фильтр для преобразование MIME-типов в совместимые типы, соответсвующие сервлету.

Вы можете использовать фильтры сервлетов для помощи сервлету в взаимодействии с внешними ресурсами. Например, можно использовать фильтр для регистрации драйвера базы данных, так что сервлет сможет испозовать драйвер для создания соединения с базой данных.

Вы можете использовать фильтры сервлетов для перехвата ответов и сжатия их до отправки клиенту. Также вы можете шифровать ответы в фильтре перед отправкой их клиенту.

Программирование фильтров

Можно разрабатывать фильтры сервлетов для выполнения различных функций, таких как вычисление времени обработки запроса сервлета, определение типа броузера, который отправляет запросы и определение MIME-типа содержимого, отправленного пользователем. Может быть несколько фильтров, которые фильтруют информацию о клиентском запросе. Эти фильтры группируются в виде цепочки фильтров. Если ваше приложение использует группу фильтров для фильтрации запросов и ответов, вам нужно явно соотнести последовательность вызовов фильтров и их целевых сервлетов во время развертывания. Рис. 73 показывает группу фильтров, фильтрующих запросы и ответы для различных сервлетов в веб-приложении:

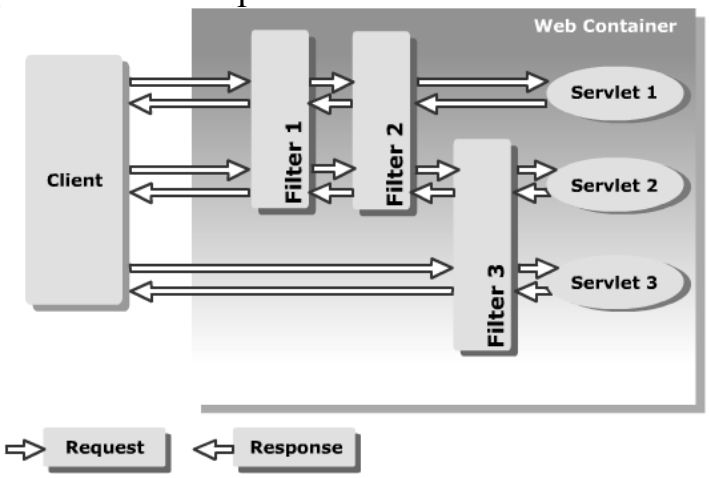


Рис. 73. Группа фильтров, перехватывающих запросы и ответы в веб-приложении

Создание фильтров

Servlet API предоставляет интерфейс `Filter`, интерфейс `FilterConfig` и интерфейс `FilterChain` пакета `javax.servlet`, которые можно использовать для разработки фильтров. Чтобы разработать фильтр, необходимо реализовать интерфейс `Filter` в вашем классе фильтра. Интерфейс `Filter` определяет методы, которые веб-контейнер вызывает для управления жизненным циклом фильтра. Табл. 25 описывает различные методы интерфейса `javax.servlet.Filter`, которые нужны фильтру для переопределения.

Таблица 25

Метод	***Описание***
`public void init(FilterConfig filt_cnfg)`	Инициализирует фильтр. Веб-

throws ServletException	контейнер вызывает его, передавая объект интерфейса `FilterConfig`, который содержит информацию инициализации фильтра.
public void doFilter(ServletRequest serv_req, ServletResponse serv_resp, FilterChain filter_chain) throws IOException, ServletException	Принимает объекты `ServletRequest`, `ServletResponse` и `FilterChain`. Получает информацию о запросе, заголовках ответа и фильтрах, которые должны вызываться.
public void destroy()	Веб-контейнер вызывает этот метод перед удалением экземпляра фильтра. Это позволяет фильтрам освобождать ресурсы, которые они занимали.

Когда клиент вызывает сервлет, который имеет присоединенный к нему фильтр, веб-контейнер инициализирует фильтр вызовом метода `init()` фильтра. Веб-контейнер передает объект `FilterConfig`, который содержит инициализационную информацию, как аргумент метода `init()`.

После инициализации фильтра веб-контейнер вызывает метод `doFilter()` фильтра и передает объекты `ServletRequest`, `ServletResponse` и `FilterChain` как аргументы. Объект `FilterChain` представляет цепочку фильтров, находящуюся в настоящий момент в приложении. Веб-контейнер вызывает метод `destroy()` перед удалением фильтра.

Вы можете использовать следующий код для разработки фильтра, который определяет полное время, занятое сервлетом для обработки запроса клиента:

```
/* Импорт необходимых пакетов. */
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
/* Определение класса фильтра, который реализует интерфейс Filter. */
public class ProcessingTimeFilter implements Filter
{
       FilterConfig flt_cnfg = null;
/* Инициализация фильтра. */
       public void init(FilterConfig f_cnfg) throws ServletException
      {
            this.flt_cnfg = f_cnfg;
      }
/* Фильтрация информации о запросах клиентов. */
      public void doFilter(ServletRequest request, ServletResponse re-
sponse, FilterChain chain) throws IOException, ServletException
      {
```

```
/* Получение начального времени. */
            long service_Start = System.currentTimeMillis();
            chain.doFilter(request, response);
/* Получение конечного времени. */
            long service_Stop = System.currentTimeMillis();
/* Получение общего времени, занятого для обработки запрса. */
            long serviceTime = (service_Stop - service_Start);
/*Получение путь к запросу*/
String path = ((HttpServletRequest)request).getRequestURI();
/*Запись времени обработки запроса в файл журнала сервера*/
flt_cnfg.getServletContext().log("Time taken to process request for:  "
+path+" is: "+serviceTime+ " milliseconds");
      }

/* Уничтожение сервлета. */
      public void destroy ()
      {
            this.flt_cnfg = null;
      }
}
```

В предложенной программе фильтр сервлетов получает время от сервера, когда клиент отправил запрос серверу, в переменной `service_Start`. Также он получает время от сервера, когда сервер отправил ответ клиенту, в переменной `service_Stop`. Затем фильтр определяет общее время, занятое сервлетом на обработку клиентского запроса, вычитая время `service_Start` из переменной `service_Stop`. После чего фильтр записывает время в файл серверного журнала.

Вопросы для самопроверки

1. Что представляет собой сервлет?
2. Какой компонент системы развертывает JavaEE веб-компонент в веб-контейнере, который предоставляет среду выполнения для сервлетов?
3. В каких пакетах находятся классы и интерфейсы, которые используются для разработки сервлета?
4. Расширением какого класса может быть разработан сервлет, если клиент использует протокол HTTP для отправки запроса сервлету?
5. В каком интерфейсе определяются методы, которые используются для управления жизненным циклом сервлета?
6. Какой интерфейс реализуется веб-контейнером, чтобы передавать конфигурационную информацию сервлету?
7. Какой интерфейс определяет методы жизненного цикла сервлетов, такие как init(), service() и destroy()?
8. Какой метод экземпляра сервлета вызывается при инициализации сервлета веб-контейнером?

9. Какой метод экземпляра сервлета вызывает Веб-контейнер, когда веб-контейнер получает клиентский запрос?
10. Какой метод Веб-контейнер вызывает перед удалением экземпляра сервлета?
11. Какие процедуры необходимо выполнить для создания и запуска сервлета ?
12. Какой объект представляет запрос, отправленный клиентом с помощью HTTP?
13. Какой объект представляет ответ, отправленный экземпляром сервлета клиенту с помощью HTTP?
14. Какой интерфейс содержит методы для извлечения заголовка запроса?
15. Какой интерфейс содержит методы для отправки ответ на запрос клиента?
16. Какой интерфейс содержит методы для установки атрибутов контекста для сервлета?
17. Какие интерфейсы содержит пакет javax.servlet.http?
18. Какой интерфейс содержит методы для получения параметров запроса, отправленного с использованием HTTP?
19. Какой интерфейс содержит методы для обработки ответа и кодов статуса для сервлетов, использующих HTTP?
20. Какой интерфейс предоставляет различные методы для сохранения состояния конечного пользователя в веб-приложении?
21. Какие события жизненного цикла сервлета генерируются внутри веб-контейнера?

Введение в технологию JSP

С приходом эры Интернет архитектура монолитных приложений изменилась в сторону многозвенной архитектуры клиент-сервер. Необходимость в написании серверных программ постепенно привела к доминированию аспектов веб-программирования. Microsoft представила Active Server Pages (ASP – Активные серверные страницы), в последствии ASP.NET, чтобы удовлетворить запросы рынка на серверное программирование. Работая по этом же направлении, Sun Microsystems выпустила *Java Server Pages* (JSP), чтобы расширить функциональность серверного программирования в Java.

Типичное веб-приложение состоит из логики представления, представляющей статическое содержимое, которое используется для дизайна структуры веб-страницы на основе разметки страницы, цвета и текста. Бизнес-логика, или динамическое содержимое, включает приложение интеллектуальных ресурсов предприятия и диагностику на осно-

ве финансовых и бизнес вычислений. При разработке веб-приложений зачастую время расходуется на то, чтобы разработчик кодировал статическое содержимое.

Технология JSP способствует разделению работы веб-дизайнера и веб-разработчика. Веб-дизайнер может разрабатывать и формулировать разметку веб-страницы, используя HTML. С другой стороны, веб-разработчик, работая независимо, может использовать код Java и теги JSP для кодирования бизнес-логики. Одновременное конструирование статического и динамического содержимого способствует разработке качественных приложений с улучшенной производительностью.

JSP-страница после компиляции генерирует сервлет и, следовательно, заключает в себе все функциональные возможности сервлетов., Таким образом, сервлеты и JSP разделяют общие возможности, такие как платформенная независимость, создание основанных на базах данных веб-приложений и возможности серверного программирования. Однако существуют некоторые фундаментальные различия между сервлетами и JSP.

В частности, сервлеты объединяют файлы (HTML-файл для статического содержимого и файл Java для динамического содержимого) для независимой обработки статической логики представления и динамической бизнес-логики. Из-за этого изменение, сделанное в любом файле, требует перекомпиляции сервлета. JSP, с другой стороны, позволяет Java быть встроенным напрямую в HTML-страницу с использованием специальных тегов. Содержимое HTML и содержимое Java могут также быть размещены в отдельных файлах. Любые изменения, сделанные в содержимом HTML, автоматически компилируются и загружаются на сервер.

Программирование сервлетов включает подробное кодирование. Поэтому любое изменение, сделанное в коде, требует распознавания содержимого статического кода (для дизайнера) и содержимого динамического кода (для разработчика), чтобы способствовать объединению изменений. С другой стороны, JSP-страница, в силу раздельного размещения статического и динамического содержимого, облегчает и для веб-разработчиков, и для веб-дизайнеров независимую работу.

Жизненный цикл JSP

Когда браузер клиента запрашивает конкретную JSP-страницу, сервер отправляет запрос контейнеру JSP. Контейнер JSP – это часть веб-контейнера, который компилирует JSP-страницу в сервлет. Рис. 74 представляет процесс потока событий, который возникает после запроса клиента к JSP-странице.

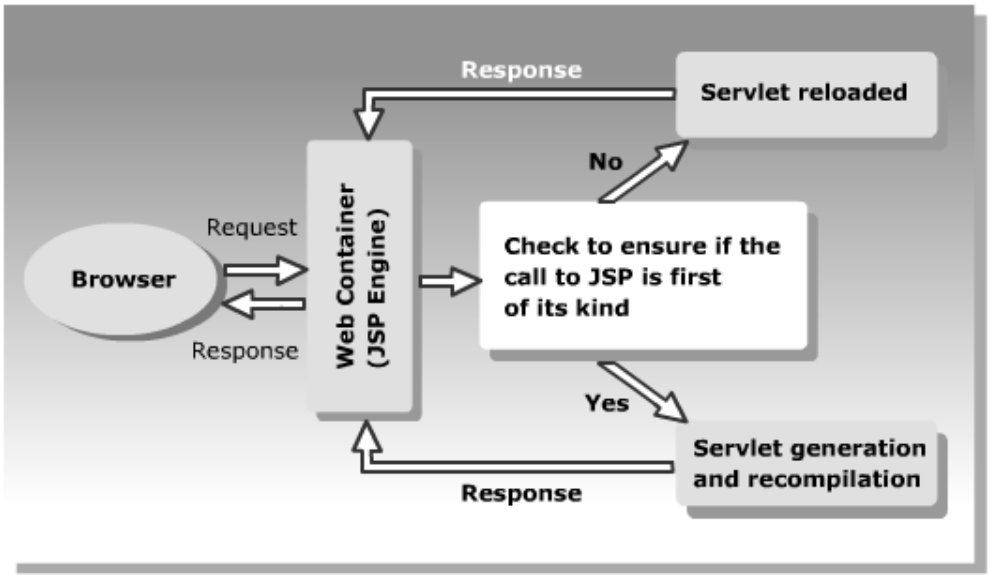


Рис. 74. Цикл запрос-ответ JSP-страницы

Цикл запрос-ответ, в сущности, заключает в себе две фазы, а именно фазу трансляции и фазу обработки запроса. Фаза трансляции реализуется контейнером JSP и заключается в генерировании сервлета. Внутренне это приводит к созданию файла класса JSP-страницы, который реализует интерфейс сервлета. Во время фазы обработки запроса генерируется ответ по спецификации запросов. Сервлет отправляет ответ полученному запросу. После того, как сервлет загружается первый раз, он остается активным, обрабатывая все последующие запросы, и сохраняет время, которое бы в противном случае тратилось бы на загрузку сервлета при каждом запросе.

После того, как JSP транслируется в сервлет, контейнер вызывает следующие методы жизненного цикла сервлета, которые определены в интерфейсе `javax.servlet.jsp.JspPage`:

`jspInit()`: Этот метод вызывается каждый раз, когда инициализируется сервлет.

`jspService()`: Этот метод вызывается, когда получен запрос на JSP-страницу.

`jspDestroy()`: Этот метод вызывается перед тем, как удаляется сервлет.

Структура JSP-страницы

JSP-страница состоит из обычных тегов HTML, представляющих статическое содержимое, и кода, заключенного в специальные теги, представляющих динамическое содержимое. Эти теги начинаются с символа “< %” и заканчиваются символом “ %>”. Элементы сценария и

директивы записываются между символами “<%” и “ %>”. Элементы сценария состоят из фрагментов кода java, в то время как директивы используются для определения спецификаций всей JSP-страницы. Комментарии, который предоставляют дополнительную информацию о различных разделах кода, заключаются в символы “<%--” и “ -- %>” . Например, следующий код JSP выводит серверное время в броузер. Он содержит HTML-содержимое и JSP-содержимое, размещенные раздельно внутри соответствующих тегов.

```
<%--Это содержимое HTML-- %>
<%@ page language=”java”  %>
<HTML>
<HEAD><TITLE>Simple JSP example></TITLE></HEAD>
<BODY>
<H1> This is a code within the JSP tags to display the server time</H1>
<%--Это содержимое JSP, которое выводит серверное время, используя класс
Date пакета java.util-- %>
<% java.util.Date now=new java.util.Date();  %>
<H2><%= now.getHours()  %>:<% =now.getMinutes()  %>:< %
=now.getSeconds()  %></H2>
</BODY>
</HTML>
```

Компоненты JSP-страницы

JSP-страница состоит из различных компонентов, которые можно использовать в вашем приложении JSP для добавления дополнительной функциональности, такой как добавление условных и циклических конструкций или использование компонентов JavaBean. Ниже представлено три компонента JSP-страницы:

- Директивы JSP
- Скриптлеты JSP
- Действия JSP

Директивы JSP

Директива в JSP-странице предоставляет глобальную информацию о конкретной JSP-странице и может быть трех типов:

- Директива page
- Директива taglib
- Директива include

Вам нужно включить символ <%@ в качестве префикса, и символ %> в качестве суффикса имени директивы, чтобы вызвать директиву. Директива может иметь более одного атрибута. Синтаксис определения директивы:

<%@ directive attribute=”value” %>;

Директива page

Директива *page* определяет атрибуты, которые уведомляют веб-контейнер о главных установках JSP-страницы. Вы можете задавать разные атрибуты в директиве page. Синтаксис директивы page:

<%@ page attribute_list %>

Табл. 26 описывает различные атрибуты, поддерживаемые директивой page, вместе с их возможными значениями и описаниями:

Таблица 26

Имя атрибута	*Описание*
language	Определяет язык сценария JSP-страницы.
extends	Определяет родительский класс, который расширяет сгенерированный JSP сервлет.
import	Импортирует список пакетов, классов или интерфейсов в генерируемый сервлет.
session	Задает, может ли генерируемый сервлет использовать сеанс или нет. Генерируется неявный объект session, если значение установлено в true. Значением по умолчанию атрибута session является true.
buffer	Задает размер буфера вывода. Если размер установлен в none, буферизация не выполняется. Значением по умолчанию buffer является 8 KB.
autoflush	Задает, что буфер вывода очищается автоматически, если значение установлено в true. Если значение установлено в false, возникает исключение при заполнении буфера. Значением по умолчанию атрибута autoFlush является true.
isThreadSafe	Задает, обеспечивает ли JSP-страница безопасное выполнение потоков или нет.
errorPage	Задает, что любое необработанное сгенерированное исключение будет передано данному URL.
isErrorPage	Задает, что текущая JSP-страница является страницей ошибок, если значение атрибута установлено в true. Значением по умолчанию атрибута isErrorPage является

Имя атрибута	*Описание*
	false.
contentType	Определяет тип Многоцелевых расширений почты (Multipurpose Internal Mail Extension – MIME) ответа. Значением по умолчанию атрибута `contentType` является text/html.

Директива `include`

Директива *`include`* используется для задания имен файлов (в виде их относительных URL), которые вставляются при компиляции JSP-страницы. Содержимое файлов, вставленных таким образом, становится частью JSP-страницы. Директива `include` также может быть использована для вставки части кода, который является общим для нескольких страниц. Это помогает избежать использования компонентов отдельно для каждой страницы. Синтаксис определения директивы `include`:

< %@ include file = ”URLname” %>

Например, строка кода для включения файла HTML (`Superstore.html`) в JSP-страницу, содержащего название и логотип магазина Superstore Online Shopping Mall, может быть написана так:

< %@ include file = ”Superstore.html” %>

Директива `taglib`

Директива *`taglib`* импортирует нестандартный тег в текущую JSP-страницу. Нестандартный тег – это определенный пользователем тег, который используется для выполнения повторяющихся задач на JSP-странице. Файл дескриптора библиотеки тегов (Tag Library Descriptor – TLD) определяет функциональность нестандартного тега.

Директива `taglib` ассоциируется с URI для уникальной идентификации нестандартного тега. Также она связывает строку префикса тега, который служит отличительным признаком нестандартного тега, с другой библиотекой тегов, используемой в JSP-странице. Синтаксис для импорта директивы `taglib` на JSP-странице:

< %@ taglib uri=“tag_lib_URI” prefix=”prefix” %>

Директива `taglib` принимает два атрибута. Табл. 27 описывет атрибуты директивы `taglib` вместе с их описанием:

Таблица 27

Атрибут	Описание
uri	Определяет местонахождение файла TLD нестандартного тега.
prefix	Определяет строку префикса, используемую для различения экземпляра нестандартного тега.

Элементы сценария JSP

Элементы сценария JSP позволяют вставлять код Java прямо в HTML-страницу. Ниже представлены различные типы элементов сценария JSP, которые можно использовать в JSP-странице:

Объявления: Объявления JSP предоставляют механизм определения переменных и методов. Предложения объявлений размещаются внутри символов <%! и %> и всегда заканчиваются точкой с запятой.

Следующий фрагмент кода использует объявления JSP для описания переменных:

```
<%!
int i=0;
int j=0;
int z=0;
int prod=0;
 %>
```

Выражения: Выражения JSP используются для прямой вставки значений для вывода на странице.Выражения JSP вычисляются, когда пользователь делает запрос HTTP. Синтаксис для включения выражения JSP в JSP-файл:

<%= expression %>

Скриптлеты: Скриптлет JSP состоит из полноценного фрагмента кода Java, который размещается внутри символов <% и %>. Скриптлеты JSP выполняются при запросе и могут вызвать использование объявлений, выражений или JavaBeans. Синтаксис объявления скриптлетов JSP для включения полноценного кода Java:

<% Java code %>

Неявные объекты JSP

Объекты в JSP могут быть созданы неявно с помощью директив, неявно с помощью стандартных действий или явно, объявляя их внутри скриптлетов. Неявные объекты JSP – это некоторые предопределенные переменные, которые могут быть включены в выражения JSP и скриптлеты. Неявные

объекты JSP реализуются классами сервлетов и интерфейсов. Табл. 28 описывает различные неявные объекты с их классами и описанием.

Таблица 28

Неявный объект	***Класс***	***Описание***
application	**javax.servlet.ServletContext**	Объект application описывает веб-приложение. Обычно это приложение в текущем веб-контексте.
config	**javax.Servlet.ServletConfig**	Представляет объект класса `ServletConfig`.
exception	**java.lang.Throwable**	Представляет исключение Throwable на JSP-странице.
out	**javax.servlet.jsp.JspWriter**	Представляет объект JspWriter для отправки ответа клиенту. `JspWriter` расширяет класс `PrintWriter` и используется JSP-страницами для отправки ответов клиентам.
page	**java.lang.Object**	Представляет текущий экземпляр JSP-страницы, который используется для ссылки на текущий экземпляр сгенерированного сервлета.
session	**javax.servlet.http.HttpSession**	Представляет объект сеанса интерфейса `HttpSession`.
response	**javax.servlet.http.HttpServletResponse**	Представляет объект ответа `HttpServletResponse`, который используется для отправки результата HTML клиенту.
request	**javax.servlet.http.HttpServletRequest**	Представляет объект запроса `HttpServletRequest`. Он используется для получения данных, отправленных вместе с запросом.
pageContext	**javax.servlet.jsp.PageContext**	Представляет содержимое JSP-страницы.

Неявные объекты JSP, такие как `request` и `response`, могут быть использованы для получения запроса от клиента и отправки отве-

та. Например, чтобы принять и обработать два числовых значения от пользователя, можно использовать объект request, чтобы получить введенные значения на HTML-странице. Следующие два файла используются для приема данных, их обработки и вывода результата.

`AcceptInput.htm`: Принимает два числовых значения от пользователя. Различные математические функции, такие как сложение и умножение, могут быть выполнены над введеными числовыми значениями. Эти числовые значения передаются как параметры в `Calculate.jsp`.

`Calculate.jsp`: Обрабатывает входные данные от пользователя и выводит результат.

Следующий код демонстрирует содержимое HTML файла AcceptInput, который позволяет веб-клиенту выбрать математические функции и ввести два числовых значения:

```
<html>
<body>
      <h2>Mathematical Functions</h2>
      <font size=4 face="Verdana" color=#112244>
      <form method="post" action="Calculate.jsp">
      <input type="radio" name="r1" value="add" checked="false">Addition
      <br>
      <input type="radio" name="r1" value="sub"
checked="false">Subtraction
      <br>
      <input type="radio" name="r1" value="div" checked="false">Division
            <br>
      <input type="radio" name="r1" value="mul"
checked="false">Multiplication
      <br>
      <br>
      Enter two numbers:
      <input type="text" name="t1" value="">
      <input type="text" name ="t2" value="">
      <br>
      <br>
      <input type="Submit" name="b1">
      </form>
      </body>
</head>
</html>
```

Рис. 75 показывает результат работы данного кода с выбранной функцией Addition.

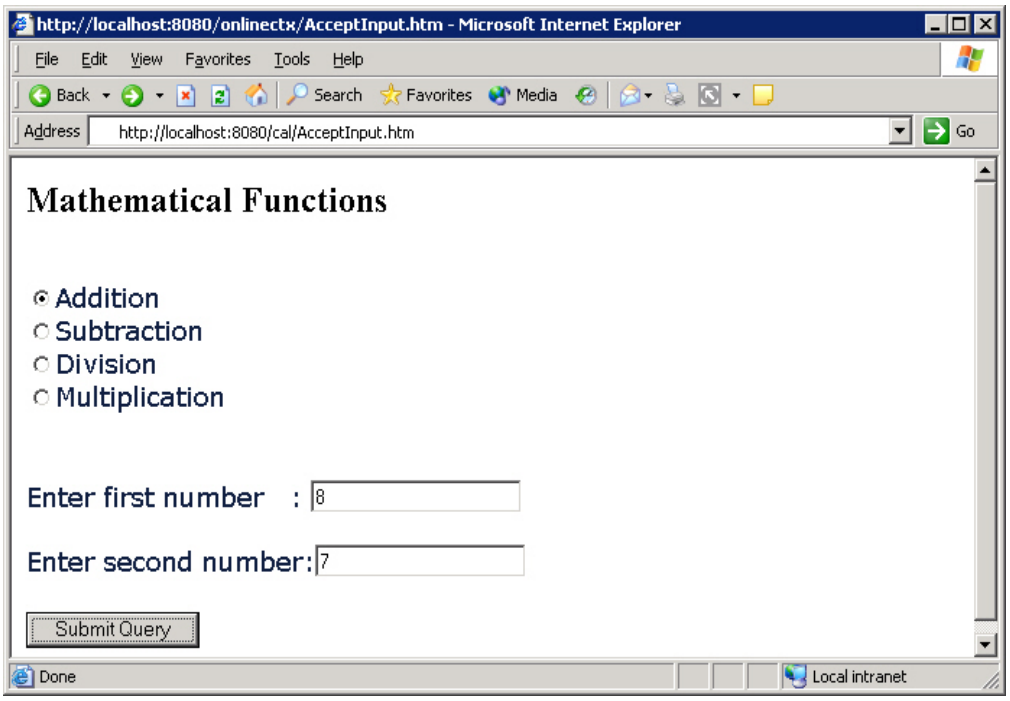


Рис. 75. Результат работы HTML-страницы AcceptInput

Когда пользователь нажимает кнопку **Submit Query**, вызывается JSP-страница `Calculate`. Следующий код демонстрирует содержимое JSP-страницы `Calculate`, которая используется для обработки выбранной математической функцией введенных числовых значений:

```
<%-- Импорт пакетов Java --%>
<%@ page language="java"%>
<%@ page import="java.lang.*"%>
<html>
<font size=4 face="Verdana" color=#112244>
<body>
<%
      /* Получение значения от радиокнопки */
      String str =request.getParameter("r1");
      String str1=request.getParameter("t1");
      String str2 = request.getParameter("t2");
      String final_output="";
      /* Объявление целочисленной переменной для выполнения вычисления*/
int num1=0;
      int num2=0;
      int num3=0;
      /* Преобразование строковых значений в целочисленные */
      num1 = Integer.parseInt(str1);
      num2 = Integer.parseInt(str2);
      /* Проверка, какая математическая функция выбрана пользователем */
      if(str.equals("add"))
      {
            num3=num1+num2;
            final_output="Addition";
      }
      if(str.equals("sub"))
      {
            num3=num1-num2;
```

```
            final_output="Subtraction";
      }
      if(str.equals("div"))
      {
            num3=num1/num2;
            final_output="Division";
      }
      if(str.equals("mul"))
      {
            num3=num2*num1;
            final_output="Multiplication";
      }
 %>
<%-- Вывод выбранного варианта --%>
The selected mathematical function by you is: <%=final_output%>
<br>
<%
      /* Вывод результата */
      Integer in=new Integer(num3);
      out.println("The Result is " + in.toString());

 %>
</body>
</html>
```

Рис. 76 показывает результат работы предсталенного кода для вывода сообщения при выборе математической функции Addition.

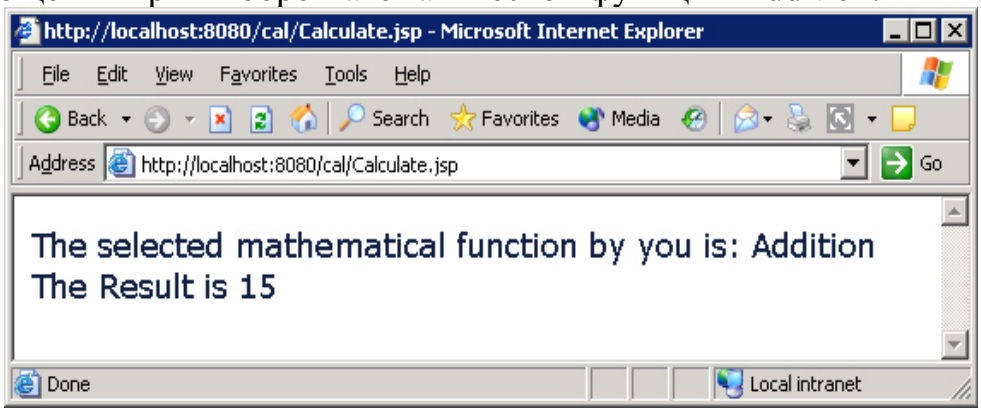


Рис. 76. Результат работы JSP-страницы Calculate

Действия JSP

Действия JSP используются для выполнения задач, таких как вставка файлов, использование bean-компонентов, передача запроса другой странице и создания объектов. Синтаксис использования действий JSP в JSP-странице:

<jsp:attribute>

Необходимо задавать значение атрибута при использовании действий JSP. Табл. 29 опиывает различные теги действий JSP вместе с их описанием, поддерживаемыми атрибутами и описанием атрибутов:

Таблица 29

Действие JSP	***Описание***	***Атрибут***	***Описание атрибута***
<jsp:useBean>	Находит и загружает существующий bean-компонент.	**id** **class** **scope** **beanName**	Однозначно идентифицирует экземпляр компонента bean. Задает класс, который реализует компонент bean. Определяет область действия компонента bean. Определяет ссылочное имя компонента bean.
<jsp:getProperty>	Получает свойство компонента bean.	**name** **property**	Определяет имя компонента bean. Определяет свойство, значение которого нужно получить.
<jsp:setProperty>	Используется для установки свойства компонента bean.	**name** **property** **value** **param**	Задает имя компонента bean. Определяет свойство, для которого устанавливается значение. Определяет точное значение свойства компонента bean. Определяет имя используемого параметра запроса.
<jsp:forward>	Используется для передачи запроса целевой странице.	**page**	Задает URL целевой страницы.
<jsp:include>	Включает файл в текущую JSP-страницу.	**page** **flush**	Задает URL включаемого ресурса. Задает, должен ли буфер освобождаться или нет. Значение flush может быть true или false.
<jsp:param>	Определяет параметр, передаваемый включенной	**name** **value**	Определяет имя ссылочного параметра. Определяет значение указанного параметра.

Действие JSP	*Описание*	*Атрибут*	*Описание атрибута*
	или перенаправленной странице.		

`<jsp:plugin>`	Выполняет апплеты Java или JavaBean.	`type`
		`code`
		
		`codebase`
			Определяет тип включаемого плагина.
			Определяет имя выполняемого плагином класса.
			Определяет путь к классу.

Действие <jsp:include>

Можно использовать действие JSP `<jsp:include>` для включения результата работы одной JSP-страницы в другую JSP-страницу. Также можно добавить файл в текущую JSP-страницу. Следует указать путь к файлу и значение параметра flush в теге <jsp:include>. Например, чтобы вывести системную дату веб-браузера, можно включить вывод другой JSP-страницы, которая имеет доступ к системной дате. Следующие два файла используются для вывода системной даты.

`IncludeDatePage.jsp`: Использует действие `<jsp:include>`, чтобы включить результат работы файла JSP `DatePage.jsp`.

`DatePage.jsp`: Использует выражение JSP для отображения текущей системной даты.

Введите `http://localhost:8080/cal/IncludeDatePage.jsp` в веб-браузере, чтобы запустить это приложение. Следующий код демонстрирует содержимое JSP-страницы IncludeDatePage, которая использует действие JSP `<jsp:include>` для включения вывода JSP-страницы DatePage:

```
<%-- Задание Java используемым языком --%>
<%@ page language="java" %>
<html>
<body>
<%-- Использование действия <jsp: include> --%>
<h4> Today's Date is: <jsp:include page="DatePage.jsp"
flush="true"/></h4>
<%
      /* Вывод сообщения пользователю */
      out.println("<h3> The output of the file DatePage.jsp is shown
above</h3>");
%>
</body>
</html>
```

Действие JSP `<jsp:include page="DatePage.jsp" flush="true"/>`, данное на JSP-странице IncludeDatePage, вызывает JSP-страницу DatePage, которая выводит системную дату. Следующий код демонстрирует содержимое файла JSP DatePage:

```
<html>
<body>
<%-- Получение системной даты --%>
<%= new java.util.Date() %>
</body>
</html>
```

Рис. 77 показывает результат работы приложения с JSP.

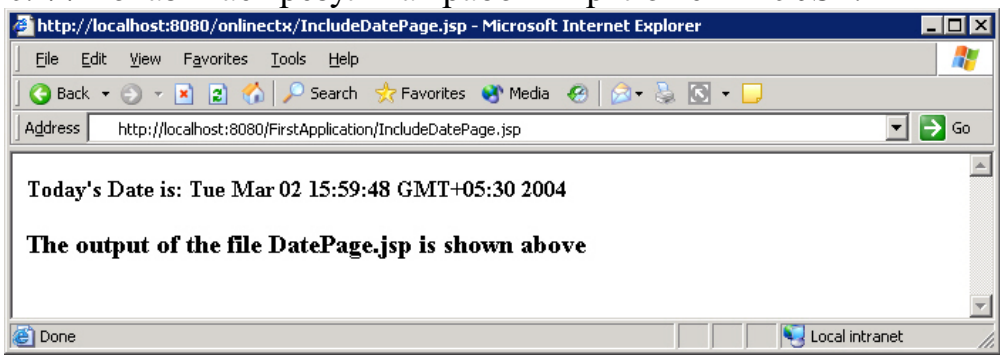


Рис. 77. Результат работы JSP-страницы IncludeDatePage

Программирование JSP

Следует понять использование различных методов, определенных в классах и интерфейсах JSP API, для создания приложения JSP. Кроме того необходимо знать различные компоненты JSP, такие как действия JSP, директивы JSP и сценарии JSP, которые обсуждались ранее. Классы, определенные в JSP API, предоставляют методы, которые доступны из JSP-страниц с помощью неявных объектов.

Классы JSP API

JSP API – это набор классов и интерфейсов, которые вы можете использовать для создания JSP-страниц. Эти классы и интерфейсы хранятся в пакете `javax.servlet.jsp`, который определяет связь между JSP-страницей и ее средой времени выполнения. Ниже представлены некоторые из классов, определенных в пакете `javax.servlet.jsp`:

- ErrorData
- JspWriter
- PageContext

Класс ErrorData

Класс ErrorData определяет информацию об ошибках для страниц ошибок. Необходимо установить значение директивы page с атрибутом isErrorPage в true для указания, что страница является страницей ошибок. Класс ErrorData расширяет класс java.lang.Object. Ниже представлены некоторые из методов, определенных в классе ErrorData, используемые в JSP-странице:

getRequestURL(): Возвращает запрашиваемый URL в виде строки.

setServletName(): Возвращает имя вызываемого сервлета в виде строки.

getStatusCode(): Возвращает код статуса ошибки в виде целого значения.

getThrowable(): Возвращает исключение Throwable, которое вызвало ошибку.

Класс JspWriter

Класс JspWriter используется для записи действий и шаблонных данных в JSP-странице. На объект класса JspWriter ссылается неявная переменная out. Класс JspWriter расширяет класс java.io.Writer. Ниже представлены некоторые из методов, определенные в классе JSPWriter, которые вы можете использовать в JSP-странице:

clear(): Удаляет содержимое буфера. Метод clear() генерирует исключение IOException, если буфер уже пустой.

close(): Закрывает и очищает поток.

flush(): Очищает поток буфера. Метод flush() очищает все буферы в цепи Writers и OutputStreams. Он генерирует исключение java.io.IOException, если сделан вызов метода write() или flush() после закрытия потока.

getBufferSize(): Возвращает размер буфера, испрользуемый JspWriter.

print(): Печатает значение логического, целого, символьного, длинного целого, с плавающей точкой, с плавающей точкой двойной точности числовых типов, а также массивы символов, строки и объекты. Метод print() генерирует исключение java.io.IOException, если во время печати возникает ошибка.

println(): Печатает значение логического, целого, символьного, длинного целого, с плавающей точкой, с плавающей точкой двойной

точности числовых типов, а также массивы символов, строки и объекты. Этот метод записывает разделитель строк, чтобы завершить текущую строку. Метод println() генерирует исключение java.io.IOException, если во время печати возникает ошибка.

Класс PageContext

Класс PageContext предоставляет информацию о контексте, когда технология JSP используется в среде сервлетов. Класс PageContext расширяет класс JspContext. Экземпляр PageContext предоставляет доступ к пространствам имен, связанным с JSP-страницей. Ниже представлены некоторые из методов, определенные в классе PageContext:

forward(): Перенаправляет текущие запрос и ответ сервлета другой странице. Это метод принимает URL целевой страницы как аргумент.

getPage(): Возвращает текущее значение объекта страницы.

getRequest(): Возвращает текущее значение объекта запроса. Возвращаемым типом метода getRequest() является запрос сервлета.

getResponse(): Возвращает текущее значение объекта ответа. Возвращаемым типом метода getResponse() является ответ сервлета.

getServletConfig(): Возвращает ServletConfig текущей страницы.

getServletContext(): Возвращает ServletContext текущей страницы.

getSession(): Возвращает HttpSession для текщего PageContext. Возвращаемым типом метода getSession() является HttpSession.

include(): Обрабатывает текущий запрос сервлета и ресурс, указанный в URL. Метод include() принимает два аргумента, путь URL и значение flush логического типа.

Этапы создания приложения JSP

Различные методы, определенные в классах JSP API, могут быть использованы для создания приложения JSP, предоставляющего различные возможности. Приложение JSP может состоять из файлов HTML, JavaBean или JSP. Следуйте представленным ниже этапам для создания приложения JSP:

Создайте пользовательский интерфейс: Пользовательский интерфейс используется для принятия входных данных от пользователя. Эти данные передаются как запрос JSP-странице.

Создайте JSP-страницу: JSP-страница предоставляет бизнес-логику для обработки запросов, переданных из интерфейса HTML. Также JSP-страница отправляет ответ веб-браузеру.

Упакуйте пользовательский интерфейс и JSP-страницу.

Разверните пакет, который вы создали. :

Запустите приложение JSP, используя веб-браузер.

Лабораторный практикум: лабораторная работа № 4, часть 2

Разработка клиентских тегов JSP

Другой подход, который можно принять для работы с повторяющейся бизнес-логикой, требуемой в приложениях JSP, состоит в использовании нестандартных тегов.

Тег – это многократно используемый блок, который используется в программах для выполнения повторяющихся задач. Например, `<CENTER>` – тег HTML, который размещает текст в центре веб-страницы. JSP позволяет создавать нестандартные теги, которые подобны тегам XML, чтобы избежать повторяющихся действий в приложении JSP. Например, вы можете создать тег приветствия, который выводит сообщение с приветствием. Все JSP-страницы вашего приложения могут использовать этот тег для вывода сообщения. Нестандартные теги упаковываются в библиотеку тегов. Вы можете включить библиотеку тегов в вашу JSP-страницу, чтобы использовать нестандартные теги из этой библиотеки.

Клиентские теги JSP

Нестандартный тег предоставляет веб-программисту механизм многократного использования и инкапсуляции сложного повторяющегося кода в приложении JSP. Нестандартные теги обеспечивают простоту и возможность повторного использования кода Java. Нестандартные теги позволяют выполнять различные функции, такие как:

- Доступ ко всем неявным объектам JSP-страницы, таким как request, response, in и out.
- Изменение ответа, сгенерированного вызывающей JSP-страницей.

- Инициализация и обработка компонентов JavaBean.

Ниже представлены различные типы нестандартных тегов, которые вы можете разрабатывать в JSP:

Пустой тег: Нестандартный тег, который не имеет атрибутов или внутреннего содержимого. Следующий фрагмент кода демонстрирует пустой нестандартный тег:

```
<td:welcome />
```

Теги с атрибутами: Нестандартные теги, для которых вы можете определять атрибуты для настройки поведения нестандартного тега. Вы можете установить значение атрибута из строковой переменной или из выражения времени выполнения с объектом запроса. Следующий фрагмент кода демонстрирует нестандартный тег с атрибутом color:

```
<td: welcome color=″blue″></td:welcome>
```

Теги с содержимым: Нестандартные теги, внутри которых можно определять вложенные нестандартные теги, элементы сценария, действия, текст HTML и директивы JSP. Следующий фрагмент кода демонстрирует нестандартный тег, который содержит элемент сценария JSP в виде его содержимого:

```
<td: welcome>
      < %=today_date %>
</td:welcome>
```

Вложенные теги: Множество нестандартных тегов, в котором один нестандартный тег содержит один или более нестандартных тегов. Тег, который содержит другие теги, называется родительским тегом. Тег, который содержится в родительском теге, называется дочерним тегом. Следующий фрагмент кода демонстрирует вложенный нестандартный тег:

```
<td1:ifTag condition "< %=eval>" >
      <td2:valueTrue>
            The expression evaluates to true
      </td2:valueTrue>
</td1:ifTag>
```

Создание клиентского тега

Следующие этапы используются для разработки нестандартных тегов:

1. Разработка обработчика тега.
2. Разработка файла дескриптора библиотеки тегов (Tag Library Descriptor – TLD).
3. Включение библиотеки тегов в JSP-страницу.
4. Развертывание приложения.

1. Разработка обработчика тега

Все нестандартные теги имеют соответствующий обработчик тега, который является классом Java, реализующим функциональность нестандартного тега. Пакет `javax.servlet.jsp.tagext` предоставляет классы и интерфейсы, которые можно использовать для разработки обработчиков тегов. Базовые классы, такие как `TagSupport` и `BodyTagSupport` пакета `javax.servlet.jsp.tagext`, реализуют интерфейс `Tag`, чтобы обеспечить реализацию методов интерфейса. Вы можете расширить эти вспомогательные классы в ваших классах обработчиков тегов и переопределить их методы, которые требуются для реализации функциональности вашего тега.

Чтобы разработать обработчик тега для пустого тега, можно расширить класс `TagSupport` пакета `javax.servlet.jsp.tagext` в вашем обработчике тега. Следующий фрагмент кода демонстрирует обработчик тега WelcomeTag, который расширяет класс `TagSupport` для реализации нестандартного тега:

import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
/* Расширение интерфейса TagSupport */
public class WelcomeTag extends TagSupport

Интерфейс `Tag` определяет методы жизненного цикла нестандартного тега. Базовые классы, такие как `TagSupport` и `BodyTagSupport`, предоставляют реализацию по умолчанию этих методов. Вы можете переопределить эти методы в вашем обработчике тега для указания, как ваш нестандартный тег должен себя вести. Ниже представлены некоторые важные методы, которые должны быть переопределены в обработчике тега пустого нестандартного тега:

`doStartTag()`: Этот метод вызывается, когда контейнер встречает начальный тег нестандартного тега. Вы можете переопределить этот метод в вашем обработчике тега, чтобы реализовать функциональность вашего тега. Для пустого нестандартного тега метод `doStart()` должен возвращать поле `SKIP_BODY` типа int, определенное в интерфейсе `Tag`. Это поле указывает, что контейнер не должен обрабатывать содержимое тела тега. Следующий фрагмент кода демонстрирует метод `doStartTag()` обработчика нестандартного тега, который выводит сообщение с приветствием:

```
public int doStartTag() throws JspException
      {
            try
                  {
```

```
                  /* Создание экземпляра JSPWriter для вывода результата
*/
                  JspWriter out=pageContext.getOut();
                  out.println("<BR><B>Welcome to New Tech Books
Inc.<B>");
            }
            catch (Exception ioException)
            {
                  System.err.println("IO Exception");
                  System.err.println("ioException.toString()");
            }
            /* Получение SKIP_BODY, так как содержимое тега не обрабаты-
вается */
            return SKIP_BODY;
      }
```

doEndTag(): Этот метод вызывается, когда контейнер встречает конечный тег нестандартного тега. Для пустого нестандартного тега метод doEndTag() должен возвращать SKIP_PAGE. SKIP_PAGE указывает, что контейнер не должен обрабатывать оставшееся содержимое JSP-страницы. Следующий фрагмент кода демонстрирует метод doEndTag() обработчика нестандартного тега, который выводит сообщение с приветствием:

```
public int doEndTag() throws JspException
      {
      /* Пропуск обработки оставшейся страницы */
      return SKIP_PAGE;
   }
```

2. Разработка файла TLD

Файл TLD определяет нестандартный тег в формате XML. Файл TLD, имеющий расширение .tld, предоставляет информацию, такую как версия библиотеки тегов, имя тега, описание тега и имя обработчика тега, который реализует тег.

Файл TLD содержит корневой элемент <taglib>, внутри которого находятся различные элементы. Файл TLD может определять более одного нестандартного тега. Каждое описание нестандартного тега расположено внутри элемента <tag>, который, в свою очередь, расположен внутри элемента <taglib>. Например, если файл TLD определяет два нестандартных тега, в нем будет два элемента <tag> внутри элемента <taglib>. Табл. 30 описывает различные элементы, которые присутствуют внутри элемента <taglib>:

Таблица 30

Имя элемента	***Описание***
<tlib-version>	Определяет версию библиотеки тегов. Элемент <tlib-version> </tlib-

	version> должен быть объявлен, когда вы определяете новую библиотеку тегов.
<jsp-version>	Определяет версию JSP-страницы, которая использует библиотеку тегов.
<short-name>	Определяет краткое наименование библиотеки тегов, с помощью которого на нее ссылаются.
<description>	Описывает библиотеку тегов, например, тип тегов, хранящихся в библиотеке.
<uri>	Определяет уникальный id библиотеки тегов.
<tag>	Содержит элементы для определения нестандартного тега.

Элемент tag, который размещается внутри элемента `taglib`, определяет нестандартный тег. Табл. 31 представляет различные подэлементы элемента tag:

Таблица 31

Имя элемента	***Описание***
<name>	Определяет имя нестандартного тега. Это обязательный тег, потому что он должен быть определен при создании нестандартного тега JSP.
<tag-class>	Определяет класс обработчика тега, который обеспечивает функциональность нестандартного тега. Это обязательный элемент, и вы должны указать полное имя класса.
<description>	Определяет функциональность тега. Это не обязательный элемент.
<body-content>	Определяет содержимое, заключенное внутри открывающего и закрывающего тегов нестандартного тега. Для пустого нестандартного тега содержимое этого элемента пусто.

Следующий фрагмент кода демонстрирует файл TLD, который определяет пустой тег Welcome, реализованный обработчиком тега WelcomeTag:

```
<taglib>
      <tlib-version>1.0</tlib-version>
      <jsp-version>1.2</jsp-version>
      <short-name>Welcome Tag</short-name>
```

```
      <description>
            A custom tag to display welcome message
      </description>
      <tag>
            <name>Welcome</name>
            <tag-class>welcome.WelcomeTag</tag-class>
            <body-content>empty</body-content>
      </tag>
</taglib>
```

3. Включение библиотеки тегов в JSP-страницу

После того, как вы реализовали обработчик тега и определили тег в TLD, вы можете использовать этот тег в вашей JSP-странице. Директива JSP `<taglib>` позволяет включать библиотеку тегов в JSP-страницу. Директива `<taglib>` содержит атрибут `uri`, который задает расположение файла TLD, и атрибут `prefix`, который задает имя, с которым в JSP-странице будет использоваться нестандартный тег.

Следующий фрагмент кода используется для включения библиотеки тегов, определенную в `Welcome.tld`, в вашу JSP-страницу:

```
< %@ taglib uri="/Welcome.tld" prefix="mytag" %>
```

После включения библиотеки тегов можно использовать следующий фрагмент кода для применения нестандартного тега:

```
<mytag:Welcome />
```

4. Развертывание приложения

Чтобы запустить JSP-страницу, которая использует нестандартные теги, необходимо развернуть JSP-страницу вместе с файлом класса обработчика тега и файлом TLD. При развертывании необходимо отобразить uri библиотеки тегов, задаваемого в атрибуте uri директивы `taglib`, на реальное расположение файла TLD в дескрипторе развертывания `web.xml` в элементе `<jsp-config>`, показанный в следующем фрагменте кода:

```
<jsp-config>
      <taglib>
         <taglib-uri>/Welcome.tld</taglib-uri>
         <taglib-location> /web-inf/Welcome.tld</taglib-location>
      </taglib>
   </jsp-config>
```

В предложенном фрагменте кода элемент taglib-uri задает значение атрибута uri директивы taglib. Элемент <taglib-location> задает действительное положение файла TLD в структуре веб-приложения.

Лабораторный практикум: лабораторная работа № 5

Библиотека стандартных тегов JSP (JavaServer Pages Standard Tag Library – JSTL)

JSTL – это коллекция стандартных тегов, которые позволяют использовать теги в JSP-страницах для выполнения повторяющихся задач, вместо использования кода скриптлетов Java. JSTL – это коллекция методов и тегов, которая общая для многих приложений JSP. Вы можете использовать JSTL для минимизации кода Java в вашем приложении JSP. JSTL позволяет изучить один набор тегов, который можно использовать в различных контейнерах JSP. Ниже представлены четыре группы, на которые делится JSTL:

- Базовая библиотека тегов
- Библиотека тегов форматирования и интернационализации
- Библиотека тегов для баз данных
- Библиотека тегов для XML

Базовая библиотека тегов

Базовая библиотека тегов – это коллекция тегов, которые используются для выполнения часто повторяющихся задач в JSP-страницах. Например, базовая библиотека тегов состоит из тегов, связанных с выражениями JSP, управлением потоками, циклами, вводом и выводом. Кроме того, базовая библиотека тегов предоставляет теги, основанные на URL ресурсах, чье содержимое может быть включено или обработано внутри JSP-страницы. Вам нужно поставить в начале букву c, когда вы используете базовую библиотеку тегов. Базовые теги делятся на четыре группы, в зависимости от функции, которую тег выполняет:

Теги поддержки переменных: Состоит из двух тегов, `remove` и `set`. Тег `set` устанавливает значение переменной языка выражений (EL) в области JSP. Вы можете использовать EL для доступа к данным приложения, хранимым в компоненте JavaBean. Тег `remove` используется для удаления переменной EL.

Теги управления потоком: Состоит из четырех тегов, `choose`, `forEach`, `forTokens` и `if`. Тег управления потоком используется для исключения скриптлетов Java из JSP-страницы. Тег `choose` используется для выполнения блоков с условием. Тег `forEach` используется для перебора всех объектов в коллекции. Тег `forTokens` позволяет перебирать коллекцию символов, разделенных разграничителем. Тег `if` разрешает условное выполнение его содержимого, заданного внутри открывающего и закрывающего тегов.

Теги управления URL: Состоит из трех тегов, import, redirect и URL. Тег import используется для доступа к основанным на URL ресурсам в JSP-страницах. Тег redirect используется для перенаправления запроса HTTP. Тег URL используется для перезаписи URL, возвращаемого JSP-страницей.

Разные теги: Содержит два тега, catch и out. Тег out вычисляет выражения, генерирует результат выражения и передает его текущему объекту JspWriter.

Библиотека тегов форматирования и интернационализации

Библиотека тегов форматирования и интернационализации состоит из тегов, которые поддерживают интернационализацию JSP-страниц. Вам нужно использовать префикс fmt при использовании тега internationalization. Тег internationalization также используется с именем I18N. I18N обозначает 18 букв между I и N в слове internationalization. Теги интернационализации делятся на три группы, в зависимости от функции, которую выполняет тег:

Теги установки локали: Состоит из двух тегов setLocale и requestEncoding. Тег setLocale переопределяет указанную клиентом локаль для страницы. Тег requestEncoding устанавливает кодировку символов запроса.

Теги передачи сообщений: Состоит из двух тегов bundle и message. Тег bundle задает группу ресурсов для страницы. Вы задаете параметр контекста javax.servlet.jsp.jstl.fmt.localizationContext в дескрипторе развертывания веб-приложения для определения группы ресурсов. Тег message используется для вывода локализованных строк.

Теги форматирования чисел и дат: Состоит из шести тегов, formatNumber, formatDate, parseDate, parseNumber, setTimeZone и timeZone. Тег formatNumber используется для вывода локализованных чисел. Теги formatDate и parseDate используются для форматирования и анализа дат. Тег timeZone устанавливает часовой пояс для использования любыми вложенными тегами formatDate.

Библиотека тегов SQL

Библиотека тегов SQL состоит из тегов, которые обеспечивают механизм взаимодействия с базой данных. Используйте префикс sql

перед тегом SQL. Ниже представлены различные теги из библиотеки тегов SQL:

Тег setDataSource: Устанавливает информацию об источнике данных для базы данных. Вы можете предоставить имя JNDI или параметры DriverManager для установки источника данных в теге setDataSource.

Тег query: Выполняет запрос SQL, который возвращает результирующее множество.

Тег update: Выполняет запрос SQL для обновления строк базы данных.

Тег transaction: Устанавливает содержимое транзакции для выполнения запросов и обновлений.

Библиотека тегов XML

Библиотека тегов XML состоит из тегов, которые могут получать доступ к элементам XML для обработки или анализа XML-документа. Используйте префикс x с библиотекой тегов XML. Теги XML делятся на три группы, в зависимости от функции, которую выполняет тег:

Базовые теги: Обрабатывают и анализируют XML-документ. Эта группа состоит из трех типов тегов, таких как, parse, set и out. Тег parse используется для анализа XML-документа. Тег set вычисляет выражения Xpath и устанавливает результат в переменную EL JSP, задаваемую атрибутом var. Тег out вычисляет выражение Xpath в текущем контекстном узле. Результат вычисления тега out записывается в текущий объект JspWriter.

Теги управления потоком: Исключают скриптлеты Java из JSP-страницы.

Теги преобразования: Преобразуют формат XML-документа, заданный таблицей стилей XSLT. Дочерний тег param может использоваться с тегом transform для установки параметра преобразования, который определяет формат.

Вопросы для самопроверки

1. Разделению каких ролей (видов деятельности) разработки веб-приложения способствует технология JSP?
2. Какой компонент и функциональность веб-приложения формирует JSP-страница, после компиляции?
3. Какие две фазы включает в себя цикл запрос-ответ JSP?
4. Из каких элементов состоит JSP-страница для представления

статического и динамического?
5. Какие различные элементы используются в JSP-страницах
6. Какими тремя способами могут быть созданы объекты в JSP?
7. Какие классы JSP API использует JSP-страница, которые определены в пакете javax.servlet?
8. Каких этапов необходимо придерживаться при разработке приложения JSP?
9. Как можно указать контейнеру, что он должен ставить в очередь многочисленные запросы к JSP-странице и отправлять только один запрос за раз?
10. Как можно указать размер буфера объекта out?
11. Экземпляром какого класса является неявный объект JSP exception?
12. Укажите неявный объект и соответствующий метод, используемые для записи в журнал сообщений с ошибками в JSP-странице.
13. Какая константа может использоваться в обработчике тега для указания, что контейнер должен пропустить обработку оставшегося содержимого JSP-страницы?

Введение в XML и WEB-сервисы

Роль XML в платформе Java

Для эффективного обмена данными между подразделениями и организациями в веб-приложениях корпоративного уровня формат, в котором данные передаются, должен быть независимым от системы, поскольку взаимодействующие приложения могут использовать гетерогенные информационные системы и базы данных. XML позволяет представлять и передавать данные независимым от системы способом и обеспечивает переносимость данных, допуская использование определяемых пользователем тегов и схем файлов. Определяемые пользователем теги представляют информацию, которая описывает ее содержание. Например, в XML-документе можно определить тег `<empid>` для хранения идентификатора служащего, и тег `<salary>` для хранения его заработка. Файл схемы XML описывает структуру XML-документа; любая система может проверить, что информация, хранимая в XML-документе, имеет в правильную структуру.

Вследствие переносимости данных и независимости от платформы XML является естественным выбором для разработки веб-приложений корпоративного уровня.

Основными чертами XML и Java являются:

- Независимость от платформы: Поддерживает межплатформенную спецификации данных XML наряду с Java для разработки интеграции приложений предприятий.
- Безопасность: Поддерживает аутентификацию и авторизацию пользователя для доступа к важным данным предприятия.
- Масштабируемость: Поддерживает внедрение новых технологий на предприятии без перепроектирования, разработки и развертывания используемых технологий.
- Надежность: Поддерживает регулярное обслуживание и качественное сопровождение данных предприятия.

  Преимуществами разработки веб-приложения с помощью XML являются:
- Поддержка обмена данными между гетерогенными базами данных и системами.
- Перераспределение нагрузки по обработке данных на веб-браузер.
- Интеграция Java-серверов с веб-браузерами.

Введение в концепцию WEB-сервисов

Веб-сервисы позволяют интегрировать приложения, разработанные на различных языках и работающие на различных платформах. Приложение, которое реализует веб-сервис, использует интерфейс для определения его сервисов в формате XML. Интерфейс веб-сервиса публикуется в централизованном реестре. Другие приложения, которым нужен доступ к веб-сервису, получают информацию о веб-сервисе из его интерфейса. Веб-сервисы позволяют интегрировать различные приложения, используя промышленные стандарты, такие как Simple Object Access Protocol – SOAP, Web Services Definition Language – WSDL и Universal Description Discovery and Integration – UDDI. Прежде всего Веб-сервисы предназначены для взаимодействия гетерогенных приложений. Для этого веб-сервисы предоставляют программируемый интерфейс для обмена данными между приложениями, независимо от их языка и операционной системы. Ниже представлены различные свойства веб-сервисов:

- Интероперабельность: Дает возможность связывать приложения, разработанные с помощью различных языков и работающих на гетерогенных платформах.
- Динамическая интеграция: Позволяет приложениям динамически определять местонахождение и интегрироваться с другими приложениями для обеспечения корпоративных решений.

- Промышленные стандарты: Позволяет приложениям взаимодействовать друг с другом, используя широко принятые промышленные стандарты, такие как XML, SOAP, WSDL и UDDI.
- Безопасность: Позволяет приложениям взаимодействовать друг с другом в безопасной среде, используя подпись и шифрование XML. Подпись и шифрование XML – механизмы защиты для поддержки целостности данных, которые передаются по Интернет.

Роли веб-сервисов

Существуют различные роли, обеспечивающие веб-сервисы. Эти роли отвечают за выполнение определенных функций в рамках жизненного цикла веб-сервиса для обеспечения взаимодействия между клиентами и веб-сервисом. Ниже представлены различные роли веб-сервисов:

- Провайдер сервиса
- Брокер сервиса
- Клиент сервиса

Провайдер сервиса

Провайдер сервиса отвечает за обеспечение программных компонентов, которые публикуются в качестве веб-сервисов. Программными компонентами могут быть простые классы Java или сложные приложения, написанные на любом языке программирования. Эти программные компоненты реализуют функциональность веб-сервиса. Провайдер сервиса описывает веб-сервис, используя интерфейс. Интерфейс определяет следующую информацию:

- Методы сервиса, вызываемые клиентом.
- URL, который клиент должен использовать для доступа к сервису.
- Сетовой протокол, требуемый для доступа к веб-сервису.

Брокер сервиса

Интерфейс, определяемый провайдером услуг, публикуется в централизованном реестре сервисов, называемом брокером сервисов. Брокер сервиса позволяет веб-клиентам искать в реестре информацию об опубликованных веб-сервисах. Клиент и веб-сервис находят друг друга, используя брокер сервиса.

Клиент сервиса

Клиент сервиса – потенциальный клиент провайдера веб-сервиса, информация о котором делается доступной брокером сервиса. Клиент сервиса после нахождения веб-сервиса в реестре вызывает сервисы, реализуемые веб-сервисом. Обнаружение веб-сервиса в реестре и вызов его методов представляют собой операцию связывания с веб-сервисом. Клиентом сервиса может быть простое веб-приложение или другой веб-сервис, получающий доступ к опубликованным веб-сервисам.

Жизненный цикл веб-сервисов

Жизненный цикл веб-сервиса состоит из различных стадий процессов разработки и развертывания веб-сервиса и включает:

- Разработка веб-сервиса
- Публикация веб-сервиса
- Размещение веб-сервиса
- Доступ к веб-сервису

Рис. 78 представляет жизненный цикл веб-сервиса:

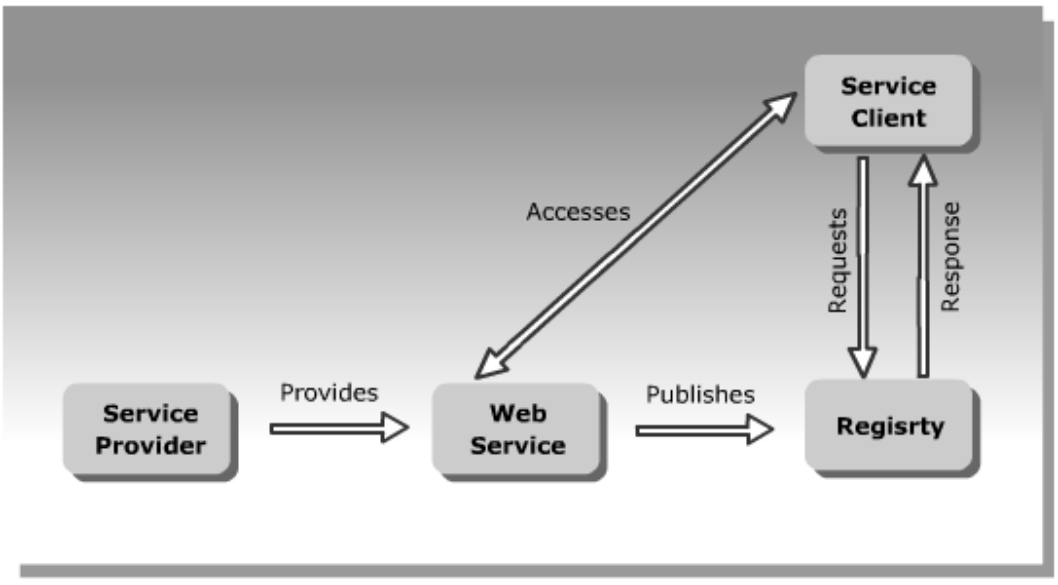


Рис. 78. Жизненный цикл веб-сервиса

Разработка веб-сервиса

Стадия разработки веб-сервиса включает описание интерфейса и реализацию бизнес-функциональности веб-сервиса. Интерфейс веб-сервиса содержит методы службы, которые могут вызываться клиентами. Провайдер сервиса отвечает за разработку веб-сервиса.

Публикация веб-сервиса

Для обеспечения доступности веб-сервис для клиентов в Интернет, веб-сервис публикуется в реестре сервисов. Опубликованный веб-сервис содержит полную информацию относительно того, как клиент может получить доступ к веб-сервису.

Поиск веб-сервиса

Клиент ищет определенный веб-сервис в реестре сервисов. Клиент может найти сервис, если сервис опубликован в реестре. Когда клиент находит веб-сервис, он получает информацию из реестра, которая требуется для доступа к веб-сервису.

Доступ к веб-сервису

После нахождения веб-сервиса клиент соединяется с ним напрямую, чтобы получить доступ к его сервисам. Клиент получает доступ к веб-сервису с помощью URL, предоставляемому провайдером сервиса в реестре по протоколу SOAP на основе XML .

Стандарты веб-сервисов

Стандарты веб-сервисов – это набор спецификаций, определенных и принятых ведущими организациями, которые обеспечивают и содействуют веб-сервисам. Эти стандарты могут реализовываться независимо различными организациями, но они предоставляют общий набор целей, к которым должны обращаться все архитектуры веб-сервисов. Стандарты веб-сервисов позволяют клиентам обращаться к веб-сервисам независимо от их типа и платформы. Ниже представлены различные стандарты веб-сервисов:

- SOAP
- UDDI
- WSDL

SOAP

SOAP – основанный на XML стандартный протокол веб-сервисов. Он является стандартом XML, используемым для обеспечения взаимодействия между веб-сервисами и клиентами. Он содержит множество правил сериализации, которые позволяют отправлять и получать информацию и позволяет различным организациям взаимодействовать и обмениваться информацией в виде сообщений SOAP. SOAP использует

данные на основе XML для шифрования сообщений и передает их через Интернет с помощью протоколов HTTP или SMTP. Он используется как гибкий коммуникационный уровень между веб-сервисами. Ниже представлены различные свойства протокола SOAP:

- Независимость от языка программирования
- Платформенная независимость
- Не требует никакой дополнительной технологии
- Объектно-ориентированный протокол

Сообщение SOAP состоит из следующих элементов:

- Элемент SOAP Envelope (конверт)
- Элемент SOAP Header (заголовок)
- Элемент SOAP Body (тело)

Все сообщения SOAP необходимо заключать внутри элемента SOAP Envelope. Элемент SOAP Envelope является самым верхним элементом сообщения SOAP, которое может быть отправлено или получено. Элемент SOAP Envelope содержит атрибут xmlns:soap, который определяет пространство имен Envelope сообщения SOAP. Атрибут soap:encodingStyle элемента SOAP Envelope определяет тип данных, используемый в сообщении SOAP.

Элемент SOAP Header используется для отправки метаданных о сообщениях SOAP. Он содержит специфичную для приложения информацию, такую как аутентификация, групповая операция и информация об оплате, связанную с сообщением SOAP. Элемент SOAP Header должен быть первым подчиненным элементом SOAP Envelope.

Элемент SOAP Body содержит сообщение, передаваемое между двумя приложениями. Это обязательный элемент сообщения SOAP и содержит спецификации о типе запроса, выполняемом клиентом, так же как в удаленном вызове процедур (Remote Procedure Calls – RPC). Элемент Body также содержит дополнительный элемент Fault, который хранит сообщения об ошибках, связанных с сообщением SOAP.

Следующий фрагмент кода демонстрирует каркас сообщения SOAP:

```
<?xml version="1.0"?>
<soap:Envelope
      xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
      soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
      <soap:Header>
            ...
            ...
      </soap:Header>
      <soap:Body>
            ...
            ...
            <soap:Fault>
            ...
```

```
            ...
            </soap:Fault>
        </soap:Body>
</soap:Envelope>
```

UDDI

UDDI – это стандарт публикации веб-сервисов. Реестры, реализующие стандарты UDDI для предоставления сервисов реестра, называются реестрами UDDI. Провайдеры сервисов используют реестры UDDI для публикации их веб-сервисов. Клиенты сервисов используют реестры UDDI для взаимодействия с опубликованными веб-сервисами. UDDI предоставляет набор API, который провайдер сервиса может использовать для публикации и размещения сервисов в реестрах. Спецификация UDDI предоставляет открытую структуру для объединения предприятий в ориентированную на сервисы архитектуру, используя промышленные стандарты, такие как XML, HTTP и SOAP. Рис. 79 демонстрирует роль реестра UDDI в веб-сервисе:

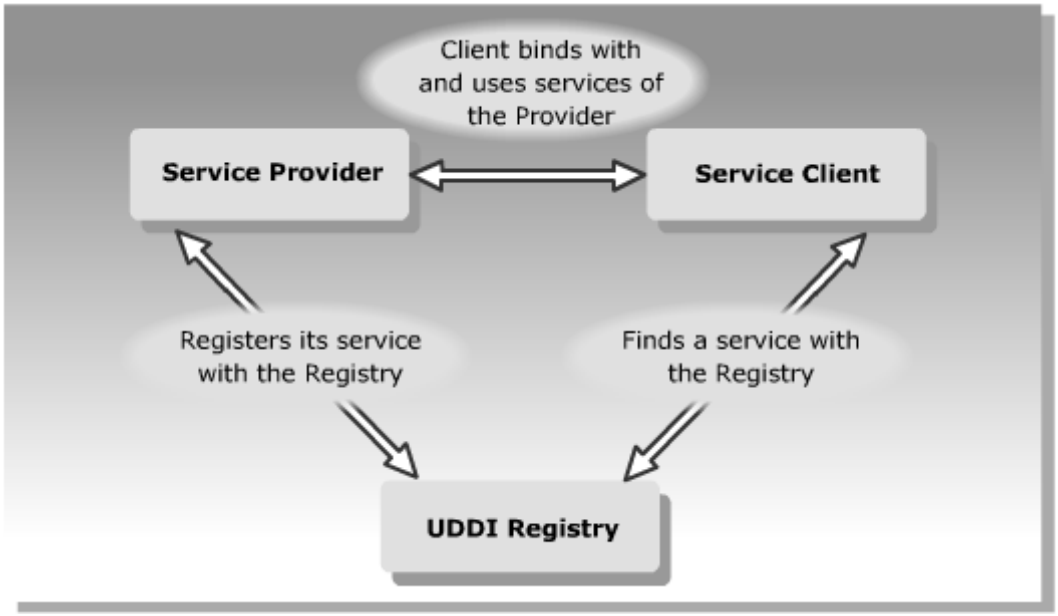


Рис. 79 Реестр UDDI

WSDL

WSDL – стандартный, основанный на XML язык, который определяет, как описывать веб-сервисы при публикации в реестре. Информация веб-сервисов публикуется в реестрах в виде документов WSDL. Документ WSDL предоставляет информацию клиентам, как получить доступ к публикуемым веб-сервисам. WSDL предоставляет эту информацию, используя различные элементы, в том числе:

`types`: Определяет различные пользовательские типы данных, которые поддерживает веб-сервис.

`message`: Определяет структуру сообщения, которая должна быть реализована для взаимодействия с веб-сервисом.

`portType`: Определяет одну или более операций, предоставляемых веб-сервисом.

`binding`: Определяет спецификации формата сообщения и протокола для определенного типа порта, такого как SOAP.

`service`: Определяет набор портов, которые представляют конечную точку веб-сервиса. Конечные точки подобны адресам веб-сервиса, которые клиент использует для взаимодействия с веб-сервисом.

Следующий фрагмент кода демонстрирует базовую структуру файла WSDL:

```
<definitions>
      <types>
            ...
            ...
      </types>
      <message>
            ...
            ..
      </message>
      <portType>
            ...
            ...
      </portType>
      <binding>
            ...
            ...
      </binding>
      <service>
            ...
            ...
      </service>
</definitions>
```

Более подробно спецификации HTTP, XML и SOAP представлены в http://www.w3c.org .

API и инструменты разработки веб-сервисов на Java

Поддержка веб-сервисов, обеспечиваемая J2EE, позволяет взаимодействовать с различными программными приложениями, которые работают на разных платформах. На платформе J2EE разрабатываются веб-сервисы, используя Java API для XML, и разворачивать веб-сервисы на сервере приложений J2EE. Большинство спецификаций в следующем списке используется когда обеспечиваются и используются веб-сервисы на языке Java. Однако, нет необходимости их использовать или даже

знать детали этих спецификаций для использования основ технологии Java для web services.

Java API веб-сервисов

Платформа J2EE предоставляет различные API и инструменты для проектирования, разработки, тестирования и развертывания веб-сервисов и их клиентов. Различными API веб-сервисов, включенными в J2EE, являются JAXP, JAXR, JAX-RPC, SAAJ и JAXB. Пакет разработки веб-сервисов на Java (Java Web Services Developer Pack – JWSDP) содержит все эти API и технологии для упрощения процесса построения веб-сервисов, используя платформу J2EE. Однако использование технологий.

Java API для обработки XML (Java API for XML Processing – JAXP)

JAXP – это API, которое используется для обработки данных XML в приложениях Java независимо от конкретной реализации обработки XML. С помощью JAXP вы можете анализировать данные в виде потока событий, используя SAX, или в виде дерева объектов, используя DOM. JAXP поддерживает стандарты таблицы стилей XML для трансформаций (XML Style sheet Transformation – XSLT), которые позволяют преобразовывать документы XML в другие форматы данных, такие как HTML и WML. С помощью JAXP вы можете использовать любой приемлемый XML-анализатор в вашем приложении.

API для JAXP определены в пакете javax.xml.parsers. JAXP поддерживает стандарты синтаксического разбора, такие как простой API для анализа XML (Simple API for XML Parsing – SAX) и объектная модель документа (Document Object Model – DOM). Ниже представлены пакеты, которые определены в JAXP API:

javax.xml.parsers
org.w3c.dom
org.xml.sax
javax.xml.transform

Java API для реестров XML (Java API for XML Registries – JAXR)

JAXR предоставляет стандартный Java API для доступа к разным реестрам веб-сервисов из приложения Java, таким как реестр UDDI. JAXR позволяет клиентам получать доступ к этим стандартным бизнес-реестрам через Интернет для информации об опубликованных веб-

сервисах. Он также позволяет провайдерам служб публиковать их сервисы в реестрах.

Java API для основанного на XML удаленного вызова процедур (Java API for XML-based Remote Procedure Calls – JAX-RPC)

JAX-RPC является Java API, который используется для создания веб-сервисов и веб-клиентов, которые выполняют удаленный вызов процедур для доступа к веб-сервису. Модель RPC позволяет клиентам использовать сервисы удаленных систем. RPC в JAX-RPC передаются с помощью SOAP. С JAX-RPC веб-сервисы и клиенты имеют преимущество использования платформенно-независимого языка. При использовании JAX-RPC даже не-Java клиенты могут получить доступ к Java веб-сервису.

API SOAP с прикрепленными данными для Java (SOAP with Attachment API for Java – SAAJ)

SAAJ –это Java API, которое используется для передачи сообщений SOAP по Интернет. Тело сообщения SOAP состоит из данных в формате XML. Данные, которые не соответствуют формату XML, могут быть отправлены в виде прикрепления к сообщению SOAP.

Сообщения SOAP бывают двух типов: сообщения с прикреплениями и сообщения без прикреплений

Java API для связывания с XML (Java API for XML Binding – JAXB)

JAXB – это API, который позволяет отображать XML-документ на объекты Java. JAXB позволяет разрабатывать приложения, которые создают объекты Java, отображаемые на XML-документ. Это позволяет работать с данными XML объектно-ориентированным способом без знания лежащей в основе структуры XML-документа.

Java API для XML Web Services (Java API for XML Web Services - JAX-WS)

JAX-WS являетtся Java API самого высокого уровня для построения веб-сервисов. Реализация JAX-WS предполагает использование реализации SAAJ, которая, в свою очередь, использует JAXP и другие Java XML спецификации. Данная спецификация появилась в реализации JavaEE на основе Java 5 и в значительной степени упрощает создание и

развертывание вею-сервисов. В настоящем пособии будет рассматриваться именно применение JAX-WS.

Пакет для разработки Java веб-сервисов (Java Web Services Developer Pack – JWSDP)

JWSDP – это интегрированный пакет, который позволяет создавать, тестировать и разворачивать XML-приложения, веб-приложения и веб-сервисы. Он обеспечивает доступ к веб-сервисам. JWSDP позволяет использовать новейшие технологии XML и веб-сервисов в приложениях Java, используя его различные компоненты. Ниже представлены различные компоненты JWSDP:

XML and Web Services Security: Гарантирует безопасность веб-сервисов, осуществляя шифрование XML и цифровые подписи XML, используя расширяемый язык разметки управления доступом (eXtensible Access Control Markup Language – XCAML), язык разметки установок безопасности (Security Assertion Markup Language – SAML), и реализуя модель безопасности веб-сервиса.

Java Server Faces (JSF) 1.0: Включает набор API, который предоставляет стандартные компоненты пользовательского интерфейса, поддерживает проверки на стороне сервера и выполняет задания обработки событий. JSF также включает библиотеку нестандартных тегов JSP, которую вы можете использовать в вашем приложении JSP для реализации бизнес-логики.

Java API for XML Binding (JAXB) 1.0.2: Предоставляет пакеты, содержащие классы и интерфейсы для автоматизации отображения XML-документов в объекты Java.

Java API for XML Processing (JAXP) 1.2.4: Предоставляет пакеты, содержащие классы и интерфейсы для анализа и преобразования XML-документов.

Java API for XML Registries (JAXR) 1.0.5: Предоставляет пакеты, содержащие классы и интерфейсы для досупа к разным типам реестров XML, таким как реестр UDDI.

Java API for XML-based Remote Procedure Calls (JAX-RPC) 1.1: Предоставляет пакеты, содержащие классы и интерфейсы для создания веб-сервисов и клиентов веб-сервисов, использующих XML и реализующих механизм RPC.

SOAP with Attachment API for Java (SAAJ) 1.2: Предоставляет пакеты, содержащие классы и интерфейсы для создания и отправки сообщений SOAP и взаимодействия между веб-сервисами и клиентами веб-сервисов.

JavaServer Pages Standard Tag Library (JSTL) 1.1: Инкапсулирует функциональность, которая является общей для различных приложений JSP, определяя стандартный набор библиотек тегов. Этот набор библиотек тегов выполняет задачи, такие как управление потоком, управление URL, управление XML-документами, управление локализованными версиями и обеспечение связи с базами данных.

Java WSDP Registry Server 1.0_06: Предоставляет реестр UDDI для объявления и поиска веб-сервисов и хранения информации реестра в репозитории XML. Вы можете использовать Java WSDP Registry Server как тестовый реестр для разработки веб-сервисов.

Tomcat (Контейнер Java Servlet и Java Server Pages): Реализует спецификации Servlet 2.4 и JavaServer Pages 2.0 и предоставялет платформу для разработки и развертывания различных веб-приложений и веб-сервисов.

Ant Build tool: Автоматизирует различные задачи разработки приложений Java, такие как создание файлов реализации веб-сервисов, используя основанные на XML конфигурационные файлы.

WS-I Sample Application: Демонстрирует практическое применение технологий веб-сервисов в приложении Supply Chain Management, удовлетворяющем требованиям рынка.

Разработка приложений с помощью JAXP

Как уже отмечалось ренее JAXP – это API, которое используется для обработки данных XML в приложениях Java независимо от конкретной реализации обработки XML.

API JAXP

Веб-сервисы платформы Java используют XML как формат данных. XML является предпочтительным форматом данных, потому что он не зависит от платформы, легко переносим и расширяем. Более того, XML таким образом кодирует данные, что их можно легко обрабатывать, используя различные языки. Необходимо также преобразовывать XML-документы в другие форматы для целей представления. Все эти возможности доступа, обработки и преобразования XML-документов с помощью приложений Java обеспечивается Java API для обработки XML (Java API for XML Processing – JAXP).

JAXP – это набор API, который используется а приложениях Java для обработки и преобразования XML-документов. JAXP состоит из трех API, Simple API for XML – SAX, Document Object Model – DOM и XML Stylesheet Language for Transformation – XSLT. *SAX API* позволяет

использовать анализатор SAX для обработки XML-документов последовательно, используя поток данных. *DOM API* позволяет использовать анализатор DOM для обработки XML-документов объектно-ориентированным способом. DOM строит объектное дерево во время обработки XML-документов. В двух словах, SAX предоставляет последовательный доступ, в то время как DOM предоставляет основанный на дереве иерархический доступ к данным в XML-документах. Можно использовать *XSLT API* для представления XML-документов в других форматах, таких как HTML, eXtensible HyperText Markup Language – XHTML и Wireless Markup Language – WML, чтобы улучшить представление и форматирование XML-данных.

Использование SAX API

SAX API используется для обработки XML-документов. Анализатор SAX является управляется событиями, при этом читает синтаксические конструкции XML-документа от начала до конца последовательно. SAX API является удобным для обработки большого XML-документа, так как он использует меньше памяти для анализа XML-документа, прскольку не загружает XML-документ в память при обработке.

Работа SAX

SAX использует классы и интерфейсы, определенные в SAX API для обработки XML-документа в форме потока данных. SAX API также определяет различныеслушателей событий, которые уведомляют анализатор SAX о событиях, таких как начало документа, конец документа, начало тега и конец тега. Анализатор SAX уведомляет приложение SAX всякий раз, когда анализатор SAX распознает любую синтаксическую конструкцию в XML-документе, которая в данный момент анализируется. Уведомление выполняется с помощью методов обратного вызова, таких как `startDocument()`, `characters()` или `endDocument()`. Эти методы определяются интерфейсами обработчика, такими как `ContentHandler`, `DTDHandler`, `ErrorHandler` и `EntityResolver`. Необходимо реализовать интерфейсы обработчика в приложении SAX и переопределить методы обратного вызова для получения уведомления о событиях анализа. Например, анализатор SAX вызывает метод `startElement()`, когда он встречает символ <, и вызывает метод `endElement()`, когда анализатор встречает символ <\. Вы можете переопределить эти методы обратного вызова в вашем приложении, чтобы реализовать логику, когда встречаются эти символы. Рис. 80 показывает работу SAX:

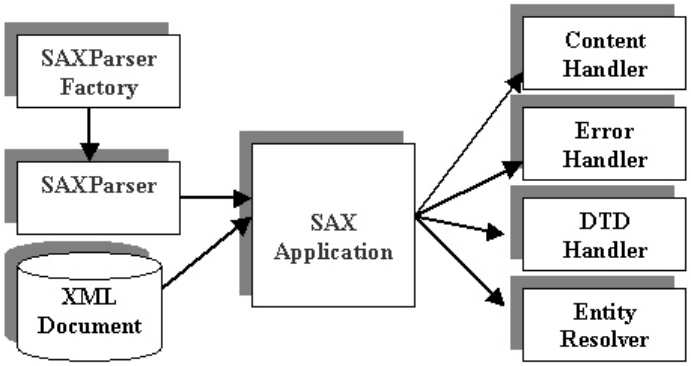


Рис. 80. Работа SAX API

Следующий фрагмент кода комментирует, как работают различные методы обратного вызова во время обработки XML-документа:

```
<!-- анализатор вызывает метод startElement() -->
<Address>
<!-- анализатор вызывает метод startElement(), метод characters() и за-
тем метод endElement() -->
<name> Tony </name>
<!-- анализатор вызывает метод startElement(), метод characters() method
и затем метод endElement() -->
<street> 172,Churchill</street>
<!-- анализатор вызывает метод startElement(), метод characters() и за-
тем метод endElement() -->
<phone number>01-8282882</phone number>
<!-- анализатор вызывает метод endElement()-->
</Address>
```

Пакеты SAX API

SAX API состоит из различных классов и интерфейсов, помогающие создавать приложения, которые могут анализировать XML-документы. SAX API состоит из следующих пакетов:

org.xml.sax
org.xml.sax.ext
org.xml.sax.helpers
javax.xml.parsers

Пакет org.xml.sax

Пакет `org.xml.sax` содержит базовые интерфейсы SAX API. Ниже представлены некоторые из интерфейсов, которые позволяют приложению Java получать уведомления, когда анализатор SAX проводит синтаксический разбор XML-документа:

ContentHandler

ErrorHandler
DTDHandler
EntityHandler

Интерфейс ContentHandler

Интерфейс ContentHandler предоставляет различные методы обратного вызова, которые вызываются, когда анализатор SAX проводит синтаксический разбор XML-документа. Можно реализовать интерфейс ContentHandler в приложении для получения уведомлений о различных событиях анализа. Ниже представлены различные методы, объявленные в интерфейсе ContentHandler:

- `startDocument()`: Вызывается, когда анализатор начинает разбор XML-документа.
- `endDocument()`: Вызывается, когда анализатор заканчивает разбор XML-документа.
- `startElement()`: Вызывается, когда анализатор встречает открывающий элемент.
- `endElement()`: Вызывается, когда анализатор встречает закрывающий элемент.
- `characters()`: Вызывается, когда анализатор встречает символьные данные.
- `ignorableWhitespace()`: Вызывается, когда анализатор встречает пробелы.

Интерфейс ErrorHandler

Интерфейс `ErrorHandler` определяет различные методы обратного вызова для обработки любой ошибки, которая может возникнуть при анализе. Можно реализовать интерфейс `ErrorHandler` в приложении для получения уведомлений о различных ошибках анализа. Ниже представлены различные методы, определенные в интерфейсе `ErrorHandler`:

- `warning()`: Получает уведомления о предупреждениях, которые генерируются во время обработки XML-документа.
- `error()`: Вызывается, когда возникает исправимая ошибка при анализе XML-документа.
- `fatalError()`: Вызывается, когда возникает фатальная ошибка при анализе XML-документа.

Интерфейс DTDHandler

Интерфейс `DTDHandler` определяет методы для получения уведомлений, когда анализатор обрабатывает Определение типа документа (Document Type Definition – DTD) XML-документа. Эти методы распознают объявления таких элементов, которые помечаются в виде ссылок, таких как графические данные и графические файлы. Ссылки представляют элементы двоичных данных в XML-документе, которые не могут анализироваться. Ниже представлены различные методы, определенные в интерфейсе `DTDHandler`:

- `notationDecl()`: Получает уведомление об элементе, объявляемом как ссылка.
- `unparsedEntityDecl()`: Получает уведомление об элементах XML-документа, который не может анализироваться.

Пакет org.xml.sax.ext

Пакет `org.xml.sax.ext` определяет различные расширения SAX, которые вы можете использовать в программе Java для выполнения сложной обработки SAX. Интерфейсы, определенные в пакете `org.xml.sax.ext` позволяют выполнять задачи, такие как доступ к лексической информации и объявлениям DTD XML-документа. Ниже представлены два интерфейса, определенные в пакете `org.xml.sax.ext`:

- `DeclHandler`: Объявляет методы, которые обеспечивают уведомления об объявлениях DTD в XML-документе.
- `LexicalHandler`: Объявляет методы, которые обеспнчивают уведомления о лексических событиях в XML-документе.

Пакет org.xml.sax.helpers

Пакет `org.xml.sax.helpers` определяет различные вспомогательные классы SAX API. Например, класс `DefaultHandler` пакета `org.xml.sax.helper` реализует интрефейсы `ContentHandler`, `DTDHandler`, `EntityResolver` и `ErrorHandler`. Вы можете расширить класс `DefaultHandler`, когда вы хотите переопределить только требуемые методы интерфейсов в вашем приложении Java.

Пакет javax.xml.parsers

Пакет `javax.xml.parsers` является коллекцией классов, которая позволяет приложениям Java обрабатывать XML-документы, используя либо SAX, либо DOM анализаторы. Ниже представлены некоторые из классов пакета `javax.xml.parsers`, специфичных для SAX:

- `SAXParser`: Позволяет анализировать XML-документы.
- `SAXParserFactory`: Позволяет получать объект SAXParsers.

Анализ XML-документа

Вы можете использовать JAXP в вашем классе Java для анализа XML-документа и вывода его содержимого. Ниже представлены этапы создания приложения JAXP для анализа XML-документа с помощью SAX:

- Импорт требуемых пакетов.
- Расширение пакета `DefaultHandler`.
- Анализ XML-документа.
- Переопределение методов обратного вызова.

Импорт требуемых пакетов

Необходимо импортировать различные пакеты для анализа XML-документа. Следующий фрагмент кода демонстрирует различные классы и пакеты, которые вам нужны для импорта анализатора XML-документа:

```
/* Нужен для операций вывода */
import java.io.*;
/* Определяет все интерфейсы анализатора SAX */
import org.xml.sax.*;
/* Определяет класс, который будет обрабатывать события SAX */
import org.xml.sax.helpers.DefaultHandler;
/* Создает экземпляр, который вы будете использовать */
import javax.xml.parsers.SAXParserFactory;
/*Анализ XML-документа*/
import javax.xml.parsers.SAXParser;
/* Представляет исключение, которое генерируется, если приложение не мо-
жет провести анализ, соответствующий выбранной конфигурации */
import javax.xml.parsers.ParserConfigurationException;
```

Расширение интерфейса `DefaultHandler`

Вам нужно реализовать интерфейс `DefaultHandler` в вашем приложении Java, потому что интерфейс `DefaultHandler` по умолчанию реализует интерфейсы `ContentHandler`, `ErrorHandler`,

DTDHandler и EntityHandler, которые используются анализатором SAX. Вам нужно переопределить методы интерфейса ContentHandler, чтобы получать уведомления, которые предоставляет анализатор SAX в ответе для различных событий анализа. Различными методами уведомлений являются: startDocument(), startElement(),endElement() и characters().

Анализ XML-документа

Чтобы проанализировать XML-документ, сначала нужно создать объект класса SAXParserFactory. Класс SAXParserFactory позволяет менять различные реализации анализаторов. Кроме того, класс SAXParserFactory также позволяет устанавливать различные свойства конфигурации анализатора SAX. Затем нужно получить объект класса SAXParser, который представляет анализатор по умолчанию. Затем вы можете проанализировать документ XML с помощью метода parse(). Метод parse() принимает два аргумента. Первый аргумент задает документ XML, который необходимо проанализировать. Второй аргумент задает объект DefaultHandler, который получает уведомления о событиях при анализе. Можно использовать следующий фрагмент кода для настройки анализатора и анализа документа XML:

```
public static void main(String fileName[])
    {
            ..
        /* Получаем экземпляр DefaultHandler, который принимает уведом-
ления о событиях при анализе */
        DefaultHandler defaultHandler = new JAX_SAX();
      /* Создаем интерфейс класса SAXParserFactory */
SAXParserFactory saxParserFactory = SAXParserFactory.newInstance();
        try
        {
            /* Устанавливаем выходной поток */
            writerOut = new OutputStreamWriter(System.out,"UTF8");
            /* Создаем экземпляр класса SAXParser для анализа документа
XML*/
            SAXParser Sax_Parser = saxParserFactory.newSAXParser();
      /* Анализируем входной файл, используя метод parse() */
Sax_Parser.parse( new File(fileName[0]), defaultHandler);
        }
        catch (Throwable thr)
        {
           /* Выводим последовательность методов, в которых возникла
ошибка */
thr.printStackTrace();
        }
        System.exit(0);
    }
```

Переопределение методов обратного вызова

Вы можете переопределить различные методы обратного вызова для обработки событий в вашем приложении, такие как события документа, события элемента и события символа. Следующий фрагмент кода показывает метод, который можно переопределить в приложении для получения уведомлений о событиях анализа:

```
public void characters(char[] ch, int start, int length) throws SAXEx-
ception
{
/*Логика для обработки уведомления*/
}

/* Получаем уведомление о событии анализа для конца документа XML */
public void endDocument() throws SAXException
{
/*Логика для обработки уведомления*/
}

/* Получаем уведомление о событии анализа для конца элемента */
public void endElement(java.lang.String namespaceURI, java.lang.String
sName, java.lang.String qName) throws SAXException
{
/* Логика для обработки уведомления */
}

/* Получаем уведомление о событии анализа для начала документа XML */
public void startDocument() throws SAXException
{
/* Логика для обработки уведомления */
}

/* Получаем уведомление о событии анализа для начала  элемента*/
public void startElement(java.lang.String namespaceURI, java.lang.String
sName, java.lang.String qName, Attributes atts) throws SAXException
{
/* Логика для обработки уведомления */}
```

Лабораторный практикум: лабораторная работа № 6, часть 1.

Использование DOM API

DOM API аналогичен SAX API, который обрабатывает документы XML. Программное обеспечение, которое реализует DOM для анализа документов XML, называется анализатором DOM. Анализатор DOM во время анализа документа XML создает структуру дерева объектов в памяти, соответствующую структуре документа XML.

Работа DOM

JAXP предоставляет DOM API, в котором определены различные классы и интерфейсы, которые вы можете использовать в приложении Java для анализа документов XML. При анализе DOM-анализатор создает структуру дерева XML-документа, известную как дерево DOM. Дерево DOM состоит из узлов, где каждый узел представляет компонент XML-документа. Различные типы узлов, поддерживаемые DOM API, следующие:

Узел элемента: Представляет отдельный тег или пару тегов в коде HTML. Узел элемента может иметь дочерний узел типов, элемент или текст.

Текстовый узел: Представляет данные типов, содержимое или символы XML-документа. Текстовые узлы не имеют дочерних узлов, но могут иметь узлы того же уровня. Узлы, которые расположены на том же уровне в дереве DOM как другой дочерний узел одного родительского узла, называются узлы одного уровня.

Узел атрибута: Представляет атрибут элемента. Узел атрибута не имеет родительского, дочернего узлов, а также узлов одного уровня.

Анализ XML-документов с помощью DOM требует много памяти, так как DOM-анализатор хранит дерево DOM в виде объектов в памяти. При этом DOM API обеспечивает объектно-ориентированный подход для анализа XML-документов. Рис. 81 показывает, как DOM-анализатор создает дерево DOM из XML-документа, который он анализирует:

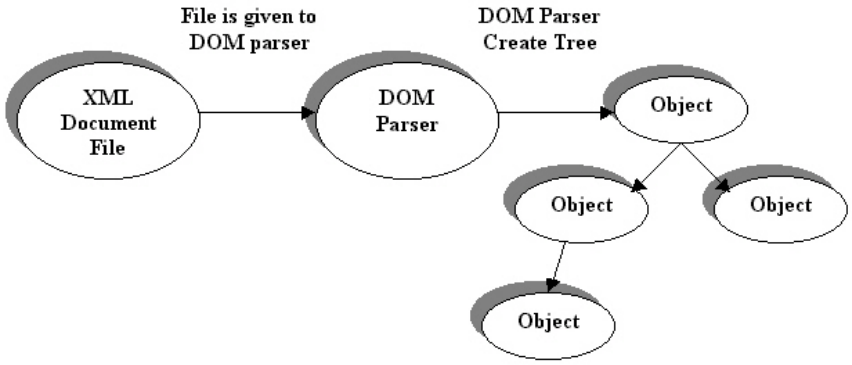


Рис. 81. Работа DOM API

Пакеты DOM API

DOM API состоит из двух пакетов, которые вы можете использовать для разработки приложения, которое может анализировать XML-документы с помощью DOM. В DOM API определены следующие два пакета:

- `org.w3c.dom`: Сожержит набор интерфейсов, позволяющие анализировать XML-документы с помощью DOM. Набор интерфейсов определяется Рабочей Группой W3C DOM.
- `javax.xml.parsers`: Кроме специфичных для SAX классов, содержит специфичные для DOM классы `DocumentBuilderFactory` и `DocumentBuilder`. `DocumentBuilderFacory` обеспечивает доступ к объектам DocumentBuilder. Объекты `DocumentBuilder` могут использоваться для анализа XML-документа и создания дерева DOM документа.

Анализ и вывод XML-документа

Ниже представлены этапы создания приложения JAXP с помощью DOM API для анализа XML-документа:

- Импорт классов, относящихся к DOM API.
- Анализ XML-документа.
- Определение различных типов узлов DOM.

Импорт классов, относящихся к DOM API

Сначала вы импортируете различные классы в ваше приложение DOM, DOM.java, чтобы вывести содержимое XML-файла. Ниже представлены различные классы, импортируемые в файл Java для вывода XML-файла:

- `java.io.*`: Содержит все интерфейсы для выполнения операций ввода/вывода.
- `org.xml.sax.*`: Содержит все интерфейсы анализатора SAX.
- `org.xml.sax.helpers.*`: Содержит класс, который обрабатывает события SAX.
- `org.w3c.dom.*`: Содержит класс `Document` и различные классы для компонентов DOM.
- `javax.xml.parsers.*`: Определяет XSLT APIs для преобразования XML-документов.

Анализ XML-документа

Имя XML-файла задается в командной строке, для выполнения анализа. Ниже представлены этапы настройки DOM для анализа:

1. Создается объект класса `DocuementBuilderFactory`.
2. Создается объект класса `DocumentBuilder`.

3. Вызывется метод `parse()` класса `DocumentBuilder` для анализа XML-документа. Имя XML-документа для анализа указывается как аргумент метода `parse()`.

Следующий фрагмент кода используется для анализа XML-документа:

```
* Шаг 1: Создание экземпляра класса DocumentBuilderFactory */
DocumentBuilderFactory docbuilderfactory = DocumentBuilderFac-
tory.newInstance();
/* Шаг 2: Создание экземпляра класса DocumentBuilder */
DocumentBuilder docBuilder = docbuilderfactory.newDocumentBuilder();
/* Шаг 3: Анализ XML-файла */
Document doc = docBuilder.parse(new File(filename));
```

Определение различных типов узлов DOM

После выполнения анализа XML-документа, анализатор создает дерево DOM для XML-документа. Можно определить различные типы узлов дерева DOM, которые соответствуют различным компонентам XML-документа. Интерфейс `Node` представляет узлы дерева DOM. Например, узел элемента в дереве DOM представляет элемент XML-документа. Поле `ELEMENT_NODE` интерфейса Node представляет узел элемента дерева DOM. Также поля `TEXT_NODE` и `ATTRIBUTE_NODE` представляют текстовые узлы и узлы атрибутов дерева DOM. Следующий код используется для определения различных типов узлов дерева DOM:

```
private void printNode(Node nd)
    {
            /* Получаем тип узла */
            int type = nd.getNodeType();
            switch (type)
            {

                  /* Определяем ELEMENT NODE */
                  case Node.ELEMENT_NODE:
                        out.print("ELEMENT NODE: ");
                        println_Com(nd);
                        break;
                  /* Определяем Text NODE */

                  case Node.TEXT_NODE:
                        out.print("TEXT NODE: ");
                        println_Com(nd);
                        break;
- - - - -
```

В представленном фрагменте кода метод `println_Com()` вызывается для вывода имени и содержимого узла. Метод `println_Com()` вызывает методы `getNodeName()` и `getNodeValue()` интерфейса `Node` для получения имени и содержимого узла. Этот метод затем выводит имя и содержимое узла. Следующий фрагмент кода демонстрирует метод `println_Com()`, который выводит имя и содержимое узла:

```
private void println_Com(Node nd)
      {
```

```
String nodeName = nd.getNodeName();
            if(nodeName != " ")
                  out.print("Node Name = \"" + nodeName + "\"");
            /* Получаем содержимое узла */
            String nameSpace = nd.getNodeValue();
            if(nameSpace != null )
            {
                  out.print("nodeValue=");
                  /* Выводим содержимое узла */
                  out.print("\"" + nameSpace + "\"");
            }
```

Лабораторный практикум: лабораторная работа № 6, часть 2

Использование XSLT API

XSLT это язык преобразований, который применяется для преобразования XML-документа в иной формат. Например, вы можете сгенерировать HTML-файл из XML-файла с помощью XSLT. XSLT это расширение языка стилей XML (Stylesheet Language – XSL), который является языком описания стилей для XML. XSL также включает словарь XML, который задает, как форматировать XML-документы. JAXP предоставляет две версии реализации XSLT:

- Интерпретирующая версия, известная как Xalan.
- Компилирующая версия, известная как компилятор XSLT (XSLTC).

Работа XSLT

XSLT состоит из трех компонентов, которые трансформируют XML-документ в требуемый формат. Эти три компонента следующие: экземпляр TransformerFactory, экземпляр Transformer и предопределенные инструкции преобразований. TransformerFactory это абстрактный класс, используемый для создания экземпляра класса Transformer, который ответственен за преобразование исходного объекта в результирующий объект.

Процесс преобразования XML начинается, когда вы создаете экземпляр класса `TransformerFactory`. Экземпляр класса `Transformer` затем создается с помощью экземпляра класса `TransformerFactory`. Этот экземпляр класса `Transformer` использует XML-документ как исходный объект и опционально использует предопределенные инструкции, требуемые для преобразований, чтобы сгенерировать форматированный результирующий объект. Вы можете создать исходный XML-документ с помощью SAX, DOM или входного потока. Результирующий объект процесса преобразования

имеет форму обработчика событий SAX, DOM или выходного потока. Рис. 82. демонстрирует работу XSLT:

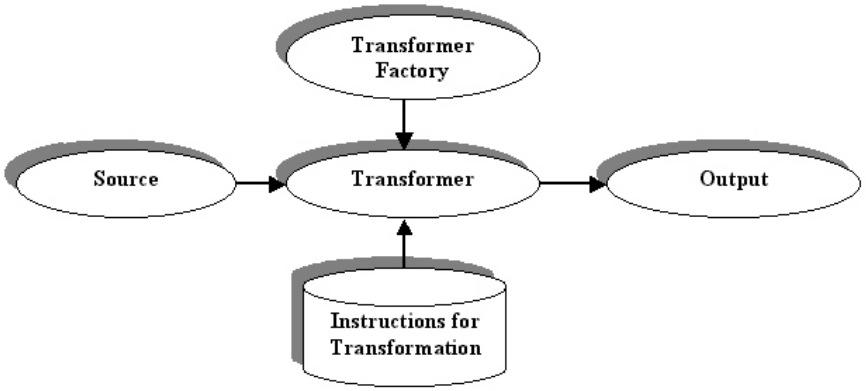


Рис. 82. Работа XSLT API

XSLT API

Пакеты XSLT API предоставляют различные объекты-фабрики для создания трансформеров. Трансформеры выполняют преобразование XML-документов другие требуемые форматы, такие как WML и XHTML. XSLT API состоит из следующих пакетов:

`javax.xml.transform`: Определяет классы `TransformerFactory` и `Transformer` для выполнения преобразования XML-документов. Экземпляр абстрактного класса `TransformerFactory` используется для создания объекта `Transformer`. Класс `Transformer` содержит метод `transform()` для преобразования входного объекта-источника в результирующий объект.

`javax.xml.transform.dom`: Содержит классы `DOMSource` и `DOMResult` для создания специфичных для DOM входных и выходных объектов.

`javax.xml.transform.sax`: Содержит классы `SAXSource` и `SAXResult` для создания входных и выходных объектов из анализатора SAX и обработчика событий SAX, соответственно.

`javax.xml.transform.stream`: Содержит классы `StreamSource` и `StreamResult` для создания входных и выходных объектов из потока ввода/вывода.

Преобразование XML-документа

Для преобразования XML-документа необходимо выполнить следующим этапы:

1. Создать таблицу стилей.
2. Создать класс Java для преобразования XML-документа.

1. Создание таблицы стилей

Необходимо создать таблицу стилей XSL для применения стилей и форматов к генерируемому результату. Таблица стилей соответствует XML-документу, который нужно преобразовать в другой формат и задает инструкции преобразования XML-файла в HTML-файл. Можно использовать элемент `xsl:output` в таблице стилей для задания результата преобразования. Атрибут метода элемента `xsl:output` задает формат вывода. Атрибут отступа задает необходимость отступа. Следующий фрагмент кода демонстрирует элемент `xsl:output` таблицы стилей:

```
<xsl:stylesheet
...
...
>
<xsl:output method="html" indent="yes"/>
...
<xsl:stylesheet>
```

Таблица стилей может содержать некоторое количество шаблонов, задаваемых тегом `<xsl:template>`. Тег `<xsl:template match="/">` обрабатывает корневой элемент таблицы стилей. Выражение `<xsl:apply-templates>` обрабатывает дочерний узел текущего узла. Каждый шаблон содержит атрибут сравнения, который выбирает элементы, которые шаблон будет применять.

Следующий фрагмент кода используется для обработки корневого элемента:

```
<xsl:template match="/">
  <html>
  <body>
   <xsl:apply-templates/>
  </body>
  </html>
</xsl:template>
```

Необходимо добавить шаблон, чтобы преобразовать элемент заголовка в тег HTML. Следующий фрагмент кода используется для обработки элемента TITLE:

```
<xsl:template match="/ARTICLE/TITLE">
<h1 align="center"> <xsl:apply-templates/> </h1>
</xsl:template>
</xsl:stylesheet>
```

В данном фрагменте кода `<xsl:apply-templates/>` гарантирует, что внутренние теги также преобразуются в элемент TITLE.

Вы также можете получить содержимое определенного узла с помощью тега `<xsl:value-of select=”name”/>`. Вы можете использовать следующий фрагмент кода для получения содержимого элементов `BookName`, `BookPrice` и `BookAuthor`:

```
      <xsl:value-of select="BookName"/>
<xsl:value-of select="BookPrice"/>
      <xsl:value-of select="BookAuthor"/>
```

2. Создание класса Java для преобразования XML-документа

Сначала импортируются пакеты, такие как `javax.xml.transform.*`, `javax.xml.transform.stream.*` и `java.io.*` в Java-класс для преобразования XML-документа в HTML-файл. Необходимо получить объект класса `TransformerFactory`, вызвав метод `newInstance()` класса `TransformerFactory`. Затем вызывается метод `newTransformer()` класса `TransformerFactory` для создания объекта класса `Transformer`. Метод `newTransformer()` принимает объект, который представляет таблицу стилей для преобразования в виде аргумента. Чтобы преобразовать XML-документ, вызывается метод `transform()` класса `Transformer`. Метод `transform()` принимает два аргумента, которые задают входной и выходной объекты преобразования. Имена XML-документа, принятого в качестве входящего, и HTML-файла в качестве выходящего, могут быть получены как аргументы командной строки. Следующий фрагмент кода используется для преобразования XML-документа в HTML-файл:

```
        /* Создаем экземпляр TransformerFactory */
            TransformerFactory transFactory = TransformerFac-
tory.newInstance();
            /* Создаем экземпляр Transformer */
            Transformer transformer = transFactory.newTransformer(new
StreamSource(argv[1]));
            /* Вызов метода transform класса Transformer */
            transformer.transform(new StreamSource(argv[0]), new Stream-
Result(new FileOutputStream (argv[2])));
```

Лабораторный практикум: лабораторная работа № 6, часть 3

Концепция создания веб-сервисов на основе JAX-WS

XML, HTTP, и SOAP являются не только спецификациями, используемыми для достижения интероперабельности между веб-сервисами и их клиентами. Поскольку большое количество спецификаций используются в веб-сервисах, для спецификаций устанавливаются версии для использования, аналогично тому как Java EE 5 устанавливает спецификации EJB 3 и Servlet 2.5.

Как уже отмечалось ранее, JAX-WS является Java API самого высокого уровня и выполняет следующие задачи:

- Замещает JAR-RPC.
- Практически не требует знаний XML или WSDL для основных WEB-services.
- Использует JARB для определения, каким образом отображаются типы данных в технологии Java и XML.
- Использует SAAJ для посылки, приема и анализа сообщений SOAP.

Использование JAX-WS на стороне клиента или сервера не требует его использования на другой стороне соединения. Если Вы разрабатываете клиентов и серверов веб-сервисов, используя Java технологию, то вам следует использовать JAX-WS API. Даже там где незначительное применение клиентов и сервисов существует исключительно на основе Java платформы, существует ряд причин для рассмотрения использования веб-сервисов.

- Гибкость поддержки будущих клиентских платформ.
- Не связана с определенной серверной платформой. Например, если серверная платформа плохо масштабируется, то она не может быть заменена без воздействия на клиентские приложения.
- Использование HTTP как технологии связи позволяет применять существующие правила firewall во многих случаях.

Рис. 83 показывает как JAW-WS технология управляет связью между Web сервером и клиентом.

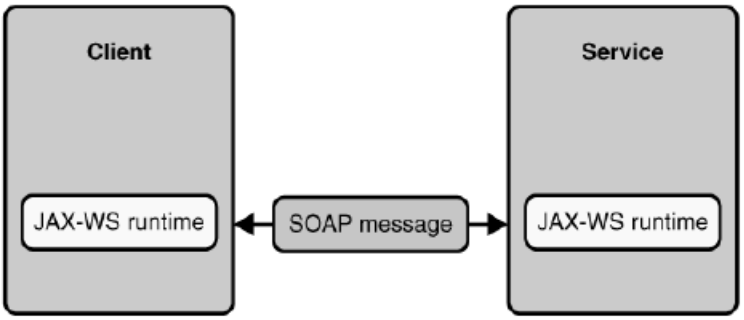


Рис. 83. JAW-WS технология управляет связью между Web сервером и клиентом

Реализация веб-сервисов JEE с помощью JAX-WS

Основными задачами, решаемыми при создании веб-сервисов являются:

- Описание конечных точек, поддерживаемых платформой Java EE 5

- Описание требований конечных точек сервлетов JAX-WS
- Описание требований конечных точек EJB JAX-WS
- Разработка клиентов веб-сервисов

Конечные точки Web Service

JAX-WS представляет собой спецификацию, которая вместе с реализацией поддерживают выполнение удаленных методов, используя сообщения на основе XML-схемы. Протоколом, используемым для транспорта сообщений, является HTTP, а для форматирования XML-сообщений используется SOAP. Sun обеспечивает реализацию ссылок (Reference Implementation (RI)) для серверных приложений для обеспечения поддержки JAX-WS. Одной из центральных частей реализации JAX-WS RI является сервлет, который обеспечивает обработку запросов и ответов HTTP.

Когда клиент веб-сервиса обращается за выполнением удаленного метода, он доставляет SOAP сообщение серверу, который в нашем случае, использует JAX-WS. Работой реализации JAX-WS является преобразование сообщения SOAP в вызовы методов и аргументов. Вам необходимо представить методы, вызываемые JAX WS и эти методы существуют только в компонентах конечной точки.

Конечная точка веб-сервиса является удаленно исполняемым компонентом, который существует на сервере и исполняется в результате получения сообщения SOAP от клиента веб-сервиса. JAX-WS 2.0, который является частью Java EE 5, определяет два типа компонентов, которые могут функционировать как конечные точки:

- Компоненты сервлета JAX-WS – JAX-WS обеспечивает сервлет, который выполняет методы в простом классе Java, содержащим аннотацию веб-сервиса. Этот сервлет иногда представляется как конечная точка веб-сервиса.
- Компоненты bean сессии без поддержки текущего состояния (Stateless) – Bean сессии без поддержки ткущего состояния являются единственным типом компонента EJB, который может являться веб-сервисом.

Вы по-прежнему можете использовать JAX-RPC, прежнюю спецификацию Java EE 5, если ваши веб-сервисы уже разработаны. Веб-сервисы, находящиеся в состояни разработки, следует разрабатывть с использованием JAX-WS. JAX-WS замещает JAX-RPC и упрощает создание и использование веб-сервисов.

Конечные точки Servlet JAX-WS

Конечная точка JAX-WS не является сервлетом в традиционном смысле. Структура JAX-WS обеспечивает реализацию сервлета, используемого для управления запросов HTTP и сообщений процесса SOAP. Конечная точка web JAX-WS:

- Является стандартным классом, созданным для обеспечения функциональности веб-сервиса.
- Является многопоточным приложением
- Не требует контейнера EJB. Это является основным достоинством этого типа конечной точки. Контейнеры сервлета и JSP, такие как Apache Tomcat могут быть установлены для поддержки этого типа веб-сервисов.

Реализация конечной точки сервлета JAX-WS

Конечная точка web JAX-WS удовлетворяет следующим требованиям:

- Имеет класс аннотаций `@javax.jws.WebService`
- Должна иметь бизнес-методы, которые являются public, но не final или static
- Содержит незащищенные методы веб-сервисы, которые аннотированы с `@javax.jws.WebMethod`
- Не может быть классом типа abstract или final
- Требует по умолчанию конструктор без аргументов

Ниже приведен фрагмент кода примера конечной точки веб компонента

```
package example;

import javax.jws.*;
@WebService public class SayHello {
@WebMethod public String getGreeting(String name)  {
return "Hello " + name;
      }
}
```

Конфигурация конечной точки сервлета JAX-WS

Необходимо изменить web.xml в вашем web archive (war) для обеспечения применимости URL для веб-сервера. Элементе <load-on-startup> следует специфицировать таким образом, что сервер загружает этот класс при запуске. Когда загружаются классы, содержащие аннотацию @WebService, то сервлет, обеспечивающий JAX-WS, выполниет

обработку всех HTTP-запросов, полученных для конфигурированной URL. Ниже приводится фрагмент кода, который представляет содержимое файла web.xml для конечной точки веб компонента:

```
<servlet>
      <servlet-name>hello</servlet-name>
      <servlet-class>example.SayHello</servlet-class>
      <load-on-startup>1</load-on-startup>
</servlet>
<scrvlet-mapping>
      <servlet-name>hello</servlet-name>
      <url-pattern>/sayhello</url-pattern>
</servlet-mappings
```

Конечные точки EJB JAX-WS

Конечными точками EJB JAX-WS могут быть только beans-сессии без поддержки текущего состояния. Аналогично конечной точки web JAX-WS, сервер приложений поддерживает обработку HTTP и SOAP, как с сервлетом. Конечная точка beans – сессии без поддержки текущего состояния:

- Как правило, уже существующая bean-сессия в разработанном приложении.
- Преобразуется в веб-сервис. Сессия bean, основанная на веб-сервисе, допустима для использования не Java клиентским приложениям. Иными словами, это может быть простым способом, позволяющим другим платформам вызывать компоненты сессии EJB без поддержки текущего состояния.
- Требует контейнер EJB.

Требования конечных точек EJB JAX-WS

Конечная точка EJB JAX-WS удовлетворяет следующим требованиям:

- Имеет аннотацию класса `@javax. jws.WebService`.
- Должна иметь бизнес-методы, которые являются public, но не final или static
- Содержит незащищенные методы веб-сервиса, которые аннотированы с `@javax.jws.WebMethod`
- Не может быть классом типа abstract или final
- Требует по умолчанию конструктор без аргументов
- Должна быть bean-сессия без поддержки текущего состояния

Ниже приведен фрагмент кода примера конечной точки EJB компонента

```
import  javax.ejb.*;
import.javax.jws.*;

@WebService @Stateless @Remote
public class SayHelloBean implements SayHelloInterface
{
      @WebMethod
      public String getGreeting(String name) {
            return "Hello " + name;
      }
}
```

Жизненный цикл конечной точки JAX-WS

Конечные точки JAX-WS для веб и EJB могут иметь дополнительные методы жизненного цикла, которые вызываются автоматически, если представлены. Любой метод может быть использован в качестве метода жизненного цикла с правильной аннотацией:

- @PostConstruct – вызывается контейнером перед тем как реализуемый класс начинает обрабатывать запрос от клиентов веб-сервиса..
- @PreDostroy – вызывается контейнером перед тем как конечная точка удаляется из обработки.

Конечная точка веб компонента следует типичной модели выполнения сервлета, которая оозначает, что обычно существует один экземпляр, который выполняется параллельно для каждого клиента. Реализации JAX-WS могут расширять пул экземпляров bean для того, чтобы обрабатывать запросы аналогично сессии EJB компонента без поддержки текущего состояния.

Разрешенные типы данных JAX-WS

В отличии от JAX-PRC, JAX-WS не определяет для типов данных связь между Java и XML. JAXB определяется посредством JAX-WS как способ преобразования данных между Java и XML в обоих направлениях.

Основные типы Java, такие как String поддерживаются автоматически, но для возвращения или передачи сложных экземпляров объекта необходимо применять программирование JAXB.

Рассмотрение JAXB выходит за рамки настоящего пособия и следует знать, JAXB используется не только JAX-WS, а может использоваться вами всякий раз, когда необходимо преобразовать Java объект в документ XML. JAXB также поддерживает генерацию классов соответ-

ствии со схемой XML и наоборот, схемы XML в соответствии с классом Java.

Ниже представлены основы аннотированного JAXB который может быть возвращен методом JAX-WS.

```
import javax.bind.annotation.XmlType;

@XmlType
public class Person {
      private String name;
      private int age;
      public Person() { }
      public person(String name, int age) {
            this.setMame(name);        "
            this.setAge(age);
      }
      public String getNamef) {
            return name;
      }
      public void setName(String name) {
            this.name = name;
      }
      public int getAge() {
            return age;
      }
      public void setAge(int age) {
            this.age = age;
      }
}
```

Ниже представлен SOAP ответ аннотированного объекта JAXB

```
<?xml version="L.0" encoding-"UTF-8"?>
<soapenv:Envelope
xmlns:soapcnv="http://schemas.xmlsoap.org/soap/envelope
/" xmlns:xsd="http://www.w3.org/2001/XKLSchema"
xmlns:nsl-"http://server/">
<soapenv:Body>
<nsl:getPersonResponse>
<return>
<age>40</age>
<name>Bob</name>
</return>
</nsl:getGersonResponse>
</soapenv:Body>
</soapenv:Envelope>
```

Клиенты веб-сервиса

Ключом к созданию клиента веб-сервиса на любом языке является наличие копии файла Web Services Description Language (WSDL). WSDL описывает операции, аргументы и возвращаемые значения используемого веб-сервиса.

Вы можете создавать JAX-WS веб-сервис либо с файлом WSDL либо с классом Java, аннотированным с @WebService. Реализацию

ссылки обеспечивают два инструментальных средства, предназначенных для генерации либо Java класса, либо файла WSDL: wsgen и wsimport. Вы можете выполнить эти программы вручную, однако, файл WSDL для JAX-WS веб-сервиса генерируется автоматически, когда веб-сервис размещается, таким образом, в этом нет необходимости.

Хотя в спецификации не определяется, но для получения сгенерированного файла WSDL применяя Sun Java Application Server можно использовать URL аналогично следующему фрагменту:

http://localhost: 8080/WSApp-war/SayHelloService?WSDL

Разработка клиентов JAX-WS

Для доступа к веб-сервису от клиента JAX-WS, необходимо выполнить следующие шаги:

- Получить файл WSDL – Любой клиент веб-сервиса не представляет каким образом сервис закодирован. Файл WSDL содержит информацию для создания интерфейсов со стороны клиента для веб-сервисов.
- Создать объект Proxy для управления созданием сообщений SOAP и связи HTTP. Этот тип класса `proxy` известен как класс `Port` в JAX-WS.
- Сгенерировать класс Port или источник кода, а также любые другие требуемые артифакты для веб-сервиса.

Ссылка на реализацию JAX-WS и the Sun Java Application Server использует wsimport для создания всего необходимого клиентского кода. Вы можете выполнить программу wsimport, используя IDE при создании процесса, обычно как задачу Ant.

A JAX-WS Client Example

В следующем примере клиента JAX-WS класс Service используется для создания экземпляра `Port` или прокси. Класс `Port` обрабатывает создание и передачу всех сообщений SOAP.

```
import javax.xml.ws.WebServiceRef;
public class WSTest {
public KSTest 0 { }

      public static void main(String[] args) {
      SayHelloService service = new SayKelloScrvice();
      SayHello port = service.getSayHelloPort();
      System.out.println(port.sayHello("Duke"));
}
}
```

Другие типы клиентов Web Service

WEB-сервисы JAX-WS являются независимыми от платформы и могут быть вызваны любым клиентом. Фактически клиент JAX-WS никогда даже не знает использует ли он сервис JAX-WS. Если необходимо поддержать клиента, разработанного на другой платформе или языке программирования, то клиент должен:

- Иметь доступ к файлу WSDL
- Поддерживать правильную версию спецификации SOAP.

Для максимальной совместимости клиенту следует поддерживать WS-I Basic Profile 1.1.

Вы можете найти дополнительную информацию о WS-I и достижении интероперабельности веб-сервисов на WEB сайте о Web Services Interoperability Organization's http://www.ws-i.org/

Вопросы для самопроверки

1. Какой язык играет важную роль в интеграции данных организаций на основе применения Java и других платформ?
2. Какой язык поддерживает платформенно-независимые данных?
3. Какие возможности предоставляет использование языка XML? (
4. Охарактеризуйте назначение Веб-сервисов
5. С помощью каких средств идентифицируется Веб-сервис для доступа клиентов, использующих основанные на XML протоколы поверх протоколов Интернет.
6. Из каких объектов состоит архитектура веб-сервисов?
7. Из каких стадий состоит жизненный цикл веб-сервиса?
8. Перечислите стандарты веб-сервисов.
9. Дайте характеристику SOAP.
10. Дайте характеристику UDDI.
11. Дайте характеристику WSDL.
12. Какая серверная платформа Java поддерживает два различных типа конечных точек веб-сервисов: сервлет и сессии EJB без сохранения состояния?
13. Дайте определение конечной точке веб-сервиса.
14. Требует ли сервлет контейнер EJB при реализации конечной точки веб-сервиса?
15. В какой последовательности должны выполняться операции по созданию и использованию веб-сервисов при использовании JAV-WS?
16. Какие различные схемы, используются для безопасности XML и веб-сервисов?

Для освоения материала предлагается выполнить лабораторную работу № 7 в разделе Лабораторный практикум.

Заключение

Авторы стремились в рамках одного пособия (без необходимости обращаться к дополнительных источниках информации) дать достаточно обширную информацию по современным методологиям и технологиям создания информационных и телекоммуникацмонных систем. В этой связи вспоминается одно из высказываний Козьмы Пруткова: «Нельзя объять необъятное». Мир открытых технологий, к числу которых относится Java, очень широк и разнообразен и вряд ли возможно решить поставленную задачу. Мы коснулись только достаточно базовых концепций применения Java для создания распределенных информационных систем, но рассмотренных технологий уже может оказаться достаточно для разработки сложных информационных систем.

В настоящее время наряду с реализациями спецификации JavaEE разработано большое количество API, которые в значительной степени упрощают и ускоряют разработку различных элементов распределенных информацинных систем. Вот только простой перечень наименований систем, которые можно применять совместно с описанной технологией JavaEE или даже вместо нее: Spring, Struts, Tapestry, Axis2, Hibernate, Seam, OSGi и мн. мн. другие.

Авторы надеются, что знания и навыки, полученные на основе данного пособия, послужат стимулом к более серьезному и глубокому изучению Java технологий.

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Установка и настройка программного обеспечения

В курсе лабораторных работ используется следующее программное обеспечение:

1. Java Development Kit (JDK) – базовый набор инструментов для разработки Java-приложений (`http://java.sun.com/javase`). Рекомендуется использовать JDK версии 1.5.0_12.

Файл дистрибутива: jdk-1_5_0_12-windows-i586-p.exe

Примечание: использование версий JDK 6 и выше может привести к проблемам несовместимости при создании и развертывании веб-сервисов на сервере JBoss версии 4.2.1.

2. Среда разработки приложений на базе платформы Eclipse (`www.eclipse.org`).

В настоящем курсе лабораторных работ предполагается использование Eclipse Europa (версия 3.3.0 или выше). Существует несколько сборок Eclipse Europa, такие как Java, Java EE, C/C++, RCP Plugin, Classic. Эти версии различаются наборами плагинов (подробнее см. http://www.eclipse.org/downloads/moreinfo/compare.php). В данном курсе используется сборка Java EE, который содержит набор плагинов WTP (Web-Tools Platform), предназначенных для разработки JavaEE компонентов: EJB, JSP, Servlets и т. д. Тем не менее, необходимо понимать, что набор плагинов WTP может быть подключен к платформе Eclipse вручную (подробнее см. http://www.eclipse.org/webtools/).

Файл дистрибутива: eclipse-jee-europa-fall2-win32.zip

3. Сервер приложений JBoss (`www.jboss.org`). В качестве сервера приложений используется JBoss Application Server (AS) версии 4.2.1 GA

Файл дистрибутива: jboss-4.2.1.GA.zip

4. JBoss Tools

(`http://www.jboss.org/tools/download/index.html`)

JBoss Tools представляет собой набор плагинов Eclipse для разработки, тестирования и развертки приложений, ориентированных на

JBoss AS. В проектах лабораторных работ используется JBoss Tools версии 2.0.1.GA.

Файл: JBossTools-2.0.1.GA-ALL-win32.zip

5. СУБД Derby (`http://db.apache.org/derby/`)

Apache Derby – простая, но в то же время достаточно мощная система управления реляционными базами данных, реализованная полностью на JAVA. Поддерживает стандарты SQL и JDBC. Используемая версия Apache Derby 10.3.2.1.

Файлы: db-derby-10.3.2.1-bin.zip

6. Плагины Derby для Eclipse

(`http://db.apache.org/derby/integrate/derby_plugin.html`)

Предназначены для удобного подключения к БД Derby из среды разработки Eclipse, просмотра структуры БД и выполнения запросов.

Файлы:

derby_core_plugin_10.3.2.599110.zip

derby_ui_plugin_1.1.1.zip

7. Утилита wsconsume, входящая в состав JBoss AS и плагин soapui для Eclipse (`http://www.soapui.org/eclipse/index.html`)

Утилита wsconsume предназначена для генерации клиентских Java-классов на основе WSDL-файла, предоставляемого поставщиком веб-сервиса. Генерируемые классы-заглушки (stubs) необходимы для вызова клиентом удаленного веб-сервиса и скрывают детали этого вызова. Утилита wsconsume поддерживает спецификацию JAX-WS.

Плагин soapui обеспечивает удобный интерфейс для разработки и использования веб-сервисов как на серверной, так и на клиентской стороне. В курсе лабораторных работ этот плагин будет использоваться в качестве графического интерфейса к утилите wsconsume.

Файлы:

wsconsume.bat (входит в состав JBoss AS)

soapui-eclipse-plugin-2.0.2.zip

Замечания:

1. Все перечисленное программное обеспечение является свободно распространяемым.

2. Материалы и примеры в предлагаемом курсе выполнены на базе ОС Windows, однако перечисленное программное обеспечение и лабораторные работы могут быть установлены и выполнены под управлением других операционных систем (Linux, MacOS и др.). Для уста-

новки JDK и Eclipse под операционную систему, альтернативную Windows, необходимо использовать соответствующий этой ОС дистрибутив.

Установка, запуск и тестирование программного обеспечения

1. Установка JDK

Запустите jdk-1_5_0_12-windows-i586-p.exe и следуйте инструкциям установки.

Предположим, что указан путь установки C:\jdk1.5.0_12.

2. Установка и запуск Eclipse

Распакуйте содержимое файла eclipse-jee-europa-fall2-win32.zip. Предположим, что распаковка была выполнена в C:\

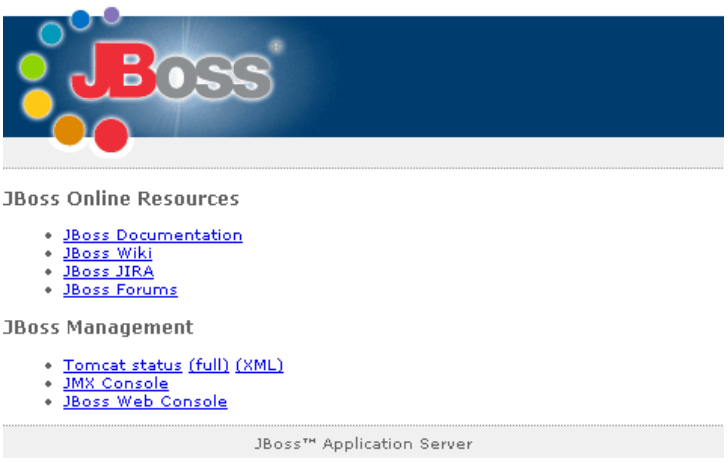
Для запуска Eclipse зайдите в каталог C:\eclipse и запустите файл eclipse.exe.

Основы работы в Eclipse и создание тестового приложения изложены в п. «Основы работы в Eclipse IDE».

3. Установка и запуск JBoss AS

1) Распакуйте соеджимое файла jboss-4.2.1.GA.zip. Предположим, что распаковка была выполнена в C:\.
2) Запустите файл C:\jboss-4.2.1.GA\bin\run.bat (или run.sh под Linux).
3) Запустите броузер и перейдите по адресу `http://localhost:8080`.

Если отобразилось следующее ниже окно, значит сервер успешно запущен и готов к работе:



4) Остановите сервер с помощью команды `C:\jboss-4.2.1.GA\bin\shutdown.bat -shutdown`
5) Попытайтесь загрузить страницу `http://localhost:8080` и убедитесь, что сервер остановлен.

Просмотр лог-файлов сервера приложений – является основным средством просмотра состояния сервера приложений и результатов выполнения тех или иных операций. Основной лог-файл сервера JBoss называется server.log и располагается в каталоге C:\jboss-4.2.1.GA\server\default\log\.

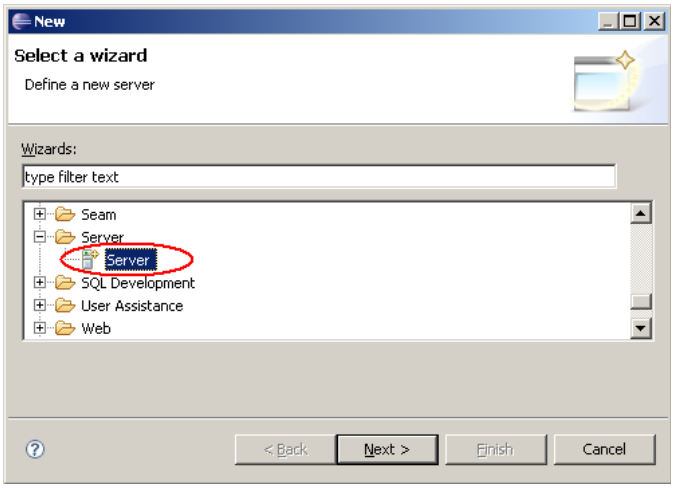
4. Подключение JBoss Tools к Eclipse

Распакуйте содержимое файла JBossTools-2.0.1.GA-ALL-win32.zip в директорию распаковки Eclipse, например в C:\.

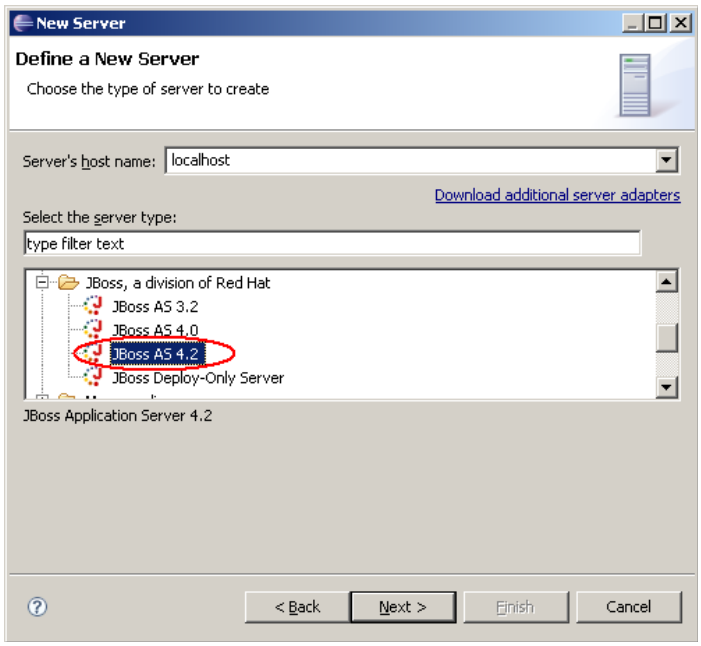
Запустите Eclipse с помощью команды `C:\eclipse\eclipse.exe -clean`.

5. Настройка подключения JBoss AS в Eclipse:

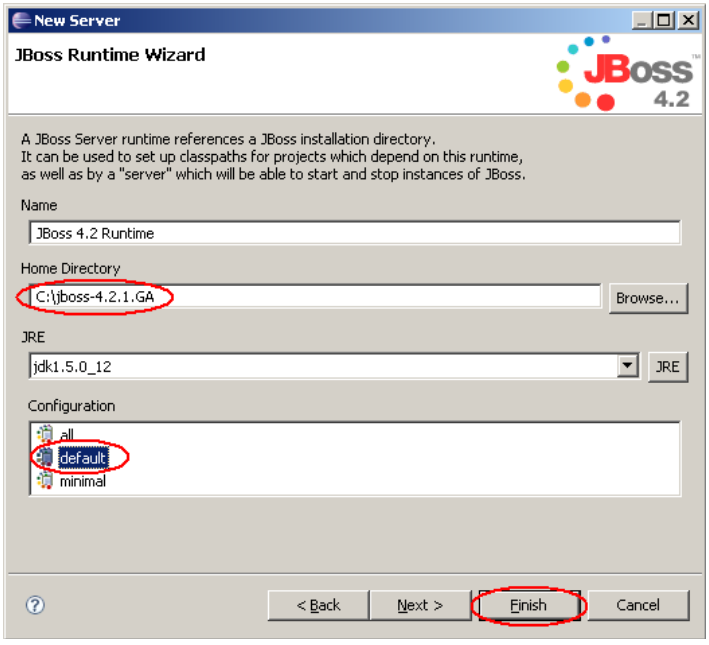
1) Выберите меню File/New/Other.
2) В появившемся окне выберите Server/Server и нажмите Next.



3) Выберите название и версию сервера приложений и нажмите Next.



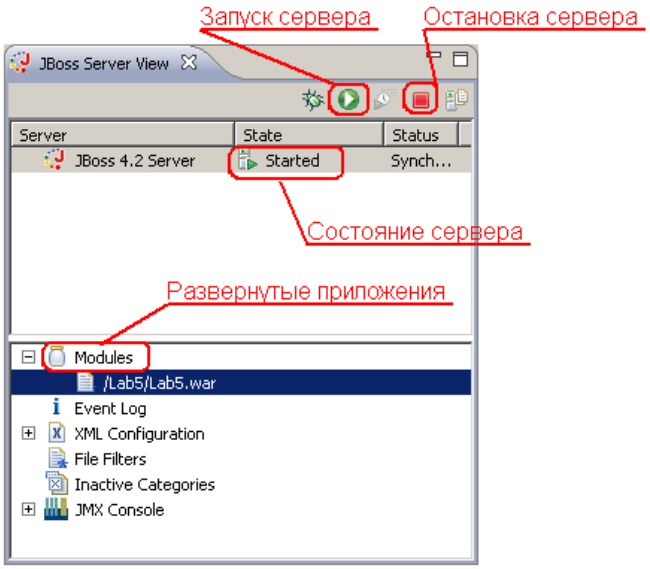
4) С помощью кнопки Browse укажите домашний каталог сервера на вашем компьютере. Если путь указан правильно, Eclipse определит имеющиеся конфигурации запуска сервера и отобразит из в списке Configuration. Оставьте по умолчанию конфигурацию Default и нажмите Finish.



Обратите внимание, что в результате этих действий автоматически открывается перспектива JBoss AS:

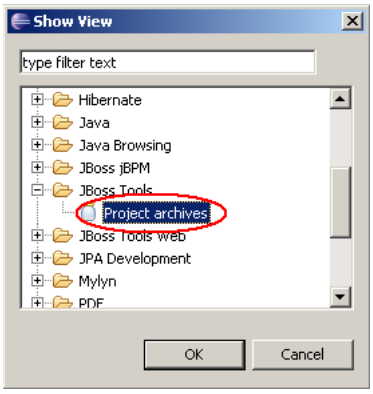
В этой перспективе, в частности, подключается панель запуска и остановки сервера приложений непосредственно из Eclipse.

Кроме того, в этой перспективе содержится представление JBoss Server View, отражающее базовые параметры состояния сервера, позволяющее управлять конфигурацией сервера, просматривать, а также удалять (undeploy) установленные приложения. Представление JBoss Server View может быть подключено с помощью команды основного меню Window/Show View/Other/Server/JBoss Server View. Здесь также можно осуществлять запуск и остановку сервера.



5) Нажмите на кнопку Start JBoss 4.2 Server в панели инструментов Eclipse. Проверьте работоспособность сервера, запустив в броузере ссылку `http://localhost:8080`.
6) Остановите сервер с помощью кнопки Stop JBoss 4.2 Server.
7) Проверьте наличие закладки Project Archives, которая по умолчанию расположена в нижней части экрана. Эта закладка понадобится при выполнении лабораторных проектов для упаковки готовых приложений. Если закладка отсутствует, подключите

ее, выбрав пункт меню Window/Show View/Other. В окне выбора вида (view), выберите JBoss Tools/Project Archives и нажмите OK.



6. Установка СУБД Apache Derby

Распакуйте содержимое файла db-derby-10.3.2.1-bin.zip. Предположим, что распаковка была выполнена в C:\.

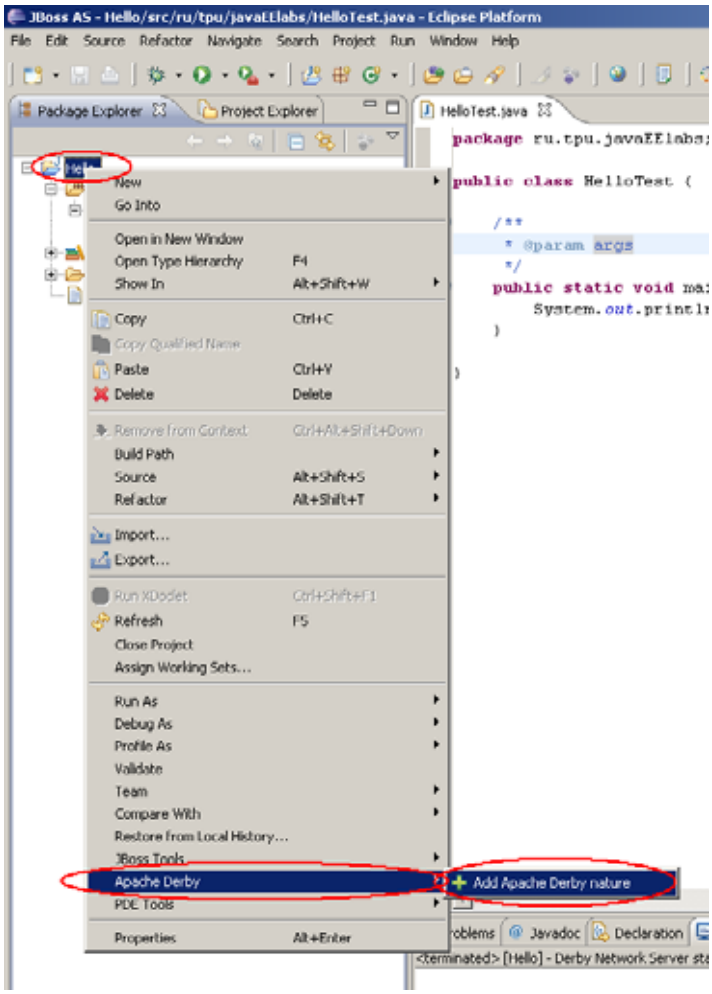
Скопируйте файл C:\db-derby-10.3.2.1-bin\lib\derbyclient.jar в каталог C:\jboss-4.2.1.GA\server\default\lib. Это необходимо для того, чтобы программы развернутые на сервере приложений могли соединиться с БД Derby. Библиотека derbyclient.jar содержит необходимые для этого классы драйвера.

7. Установка плагинов Derby для Eclipse

1) Распакуйте содержимое файлов derby_core_plugin_10.3.2.599110.zip и derby_ui_plugin_1.1.1.zip в директорию установки Eclipse, например в C:\eclipse.
2) Запустите Eclipse с помощью команды `C:\eclipse\eclipse.exe -clean`.

8. Запуск и остановка Apache Derby:

1) Выберите проект (например, Hello), откройте существующий или создайте новый проект. Щелкните правой кнопкой мыши на проект в окне Package Explorer и выберите Apache Derby/Add Apache Derby Nature. Это действие подключает необходимые библиотеки для работы с БД Derby.



2) Повторно щелкните правой кнопкой мыши на проект и выберите Apache Derby/Start Derby Network Server. В случае успешного запуска в консоли появляется сообщение:

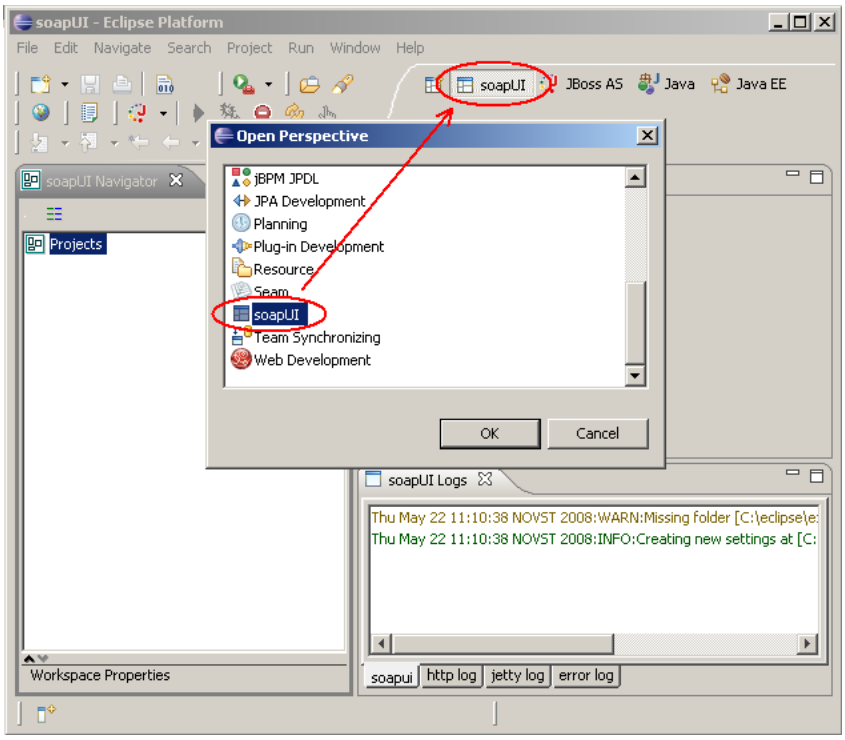
```
DRDA_SecurityInstalled.I
      Сетевой сервер Apache Derby Network Server - 10.3.2.1 -
(599110) запущен и готов принимать соединения на порту 1527
```

3) Остановите сервер, выбрав команду Apache Derby/Stop Derby Network Server. В случае успешной остановки отображается сообщение:

```
Сетевой сервер Apache Derby-10.3.2.1 - (599110) завершение работы
```

9. Настройка утилиты wsconsume и плагина soapui для Eclipse (`http://www.soapui.org/eclipse/index.html`)

1) Утилита wsconsume представлена командным файлом wsconsume.bat в каталоге <установка_jboss>/bin. Утилита не требует специальной настройки.
2) Для установки плагина soapui распакуйте содержимое файла soapui-eclipse-plugin-2.0.2.zip в директорию установки Eclipse, например в C:\eclipse.
3) Запустите Eclipse с помощью команды `C:\eclipse\eclipse.exe -clean`.
4) Выполните команду меню Window/Open Perspective/Other и проверьте наличие перспективы soapUI в списке.



Основы работы в Eclipse IDE

В настоящем разделе описываются основы использования платформы Eclipse в качестве среды разработки Java-приложений. Рассматриваются основные структурные элементы и функциональные возможности Eclipse Workbench на примере создания простого Java-приложения. В разделе рассмотрены не все возможности, используемые при выполнении лабораторных работ, однако изучение базовых принципов Eclipse позволит легко освоить новые элементы среды, необходимость в которых будет появляться в ходе выполнения заданий.

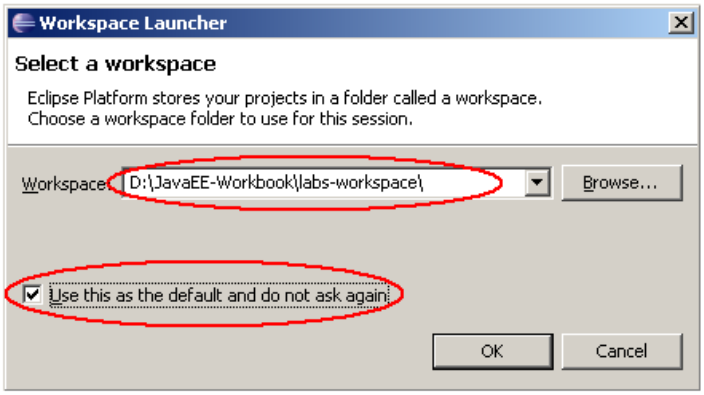
1. Установка и запуск Eclipse

Распакуйте содержимое файла eclipse-jee-europa-fall2-win32.zip. Предположим, что распаковка была выполнена в C:\.

Для запуска Eclipse зайдите в каталог C:\eclipse и запустите файл eclipse.exe.

2. Создание рабочей области (workspace)

При первом запуске Eclipse предлагает выбрать каталог рабочей области (Workspace). Это каталог, в котором будут храниться все файлы, относящиеся к проектам – настройки проектов, исходные файлы программ, результаты компиляции и сборки и т. д.



Укажите каталог рабочей области, выберите флажок Use this as the default… и нажмите OK.

При дальнейшей работе, переключение между рабочими областями, а также создание новой рабочей области выполняется с помощью команды меню File/Switch Workspace.

3. Основные представления, перспективы и редакторы

Прежде чем приступить к созданию проекта Eclipse, познакомимся поближе с основными элементами Eclipse Workbench.

Eclipse Workbench состоит из:

- перспектив (perspectives);
- представлений (views);
- редакторов (editors).

Перспектива – группа представлений и редакторов в окне Workbench. Количество перспектив в одном окне Workbench не ограничено. Так же и перспектива может содержать неограниченное количество представлений и редакторов. В одном окне каждая из перспектив может иметь различные наборы представлений, но все перспективы делят между собой один и тот же набор редакторов.

Для подключения перспективы используется команда основного меню Window/Open Perspective. Перечень открытых перспектив по умолчанию отображаются в правой верхней части экрана. С помощью этого же списка выполняется переключение между перспективами:

Представление – это визуальный компонент Workbench. Обычно он используется для перемещения по иерархии некоторой информации (например, структуры проекта), открытия редакторов, просмотра результатов выполнения и т. д. Все изменения, сделанные в представлениях, немедленно сохраняются. В большинстве случаев, в окне Workbench может существовать только один экземпляр представления некоторого типа.

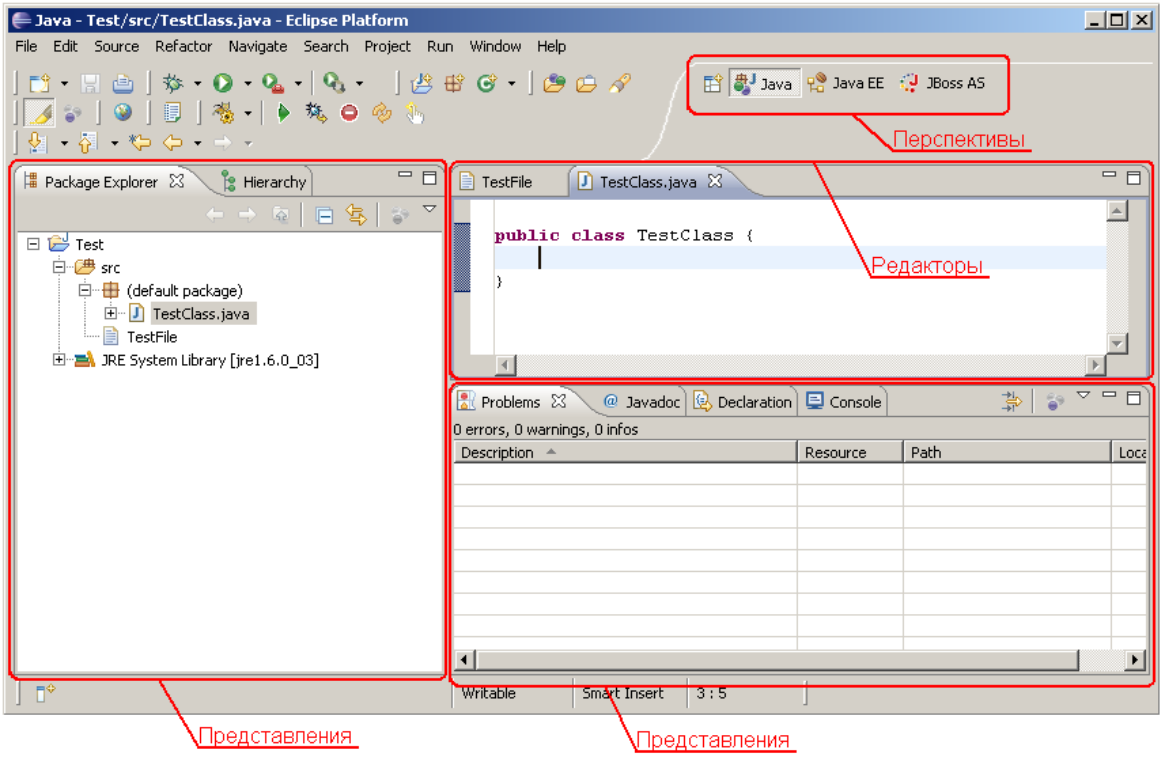
Многие представления входят в состав тех или иных перспектив и отображаются при их открытии. Тем не менее, с помощью команды меню Window/Show View можно открыть любое представление, независимо от текущей перспективы.

Восстановить набор представлений текущей перспективы можно с помощью команды Window/Reset Perspective.

Редактор также является визуальным компонентом Workbench. Используется для просмотра и редактирования некоторого ресурса, например Java-класса или файла, содержащего SQL-запросы. Изменения, делающиеся в редакторе, подчиняются модели жизненного цикла «от-

крыть-сохранить-закрыть» (an open-save-close lifecycle model). В Workbench может существовать несколько экземпляров редакторов.

Редакторы того или иного типа обычно отображаются автоматически при двойном щелчке на соответствующий ресурс (например, Java-класс). Сохранение данных выполняется с помощью команды меню File/Save, либо нажатием на клавиши Ctrl-S.



В данный момент времени может быть активным только одно преставление или редактор. Активным является представление или редактор у которого подсвечен заголовок. На активный элемент будут воздействовать общие операции вырезки, копирования и вставки (cut, copy, paste). Также активный элемент обуславливает содержимое строки состояния (status line). Если закладка редактора белая, то это означает, что данный редактор неактивен, однако представления могут отображать информацию, полученную из редактора, бывшего активным последним.

4. Создание проекта и ресурсов проекта

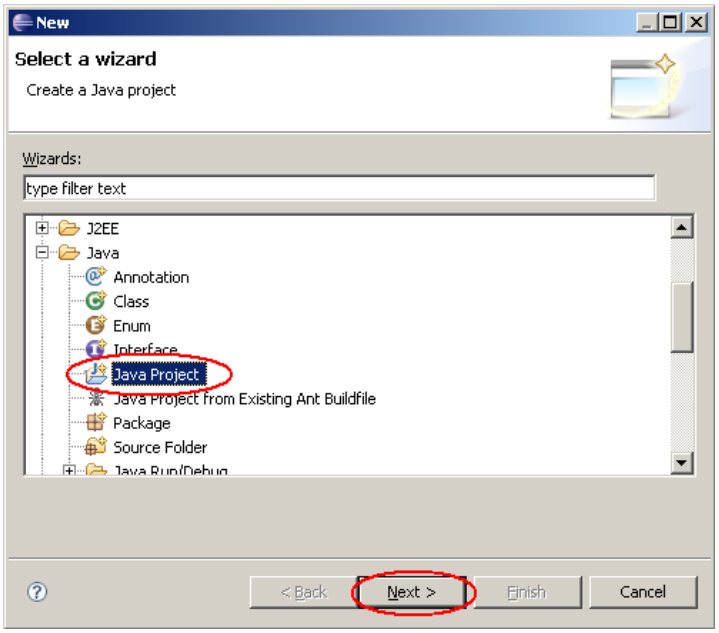
Проект Eclipse – это совокупность служебных и конфигурационных файлов Eclipse и ресурсов проекта, относящихся к конкретному разрабатываемому приложению или подсистеме приложения. Физически на диске проект представляет собой каталог по имени проекта внут-

ри каталога рабочей области, внутри которого содержатся файлы – составляющие проекта.

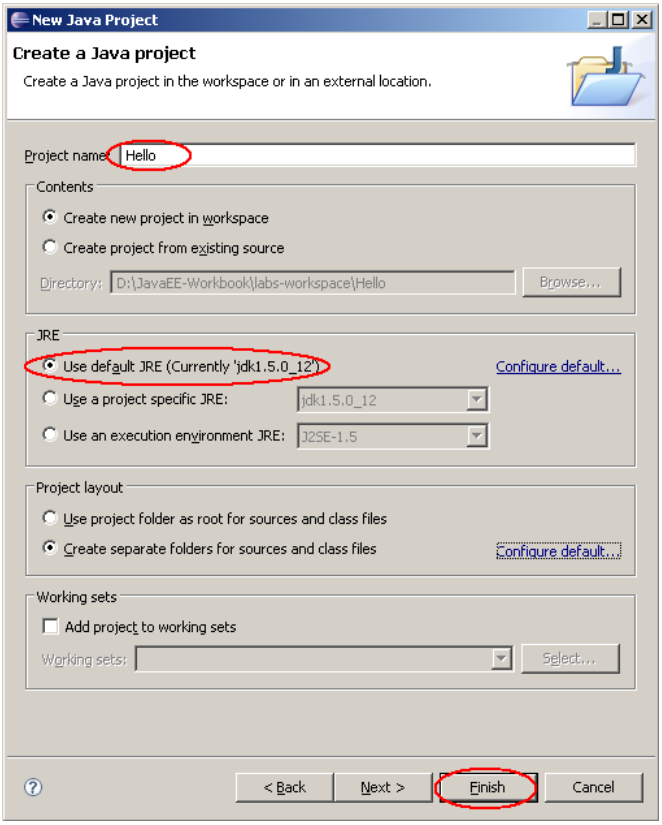
Под **ресурсом проекта** понимаются любые элементы, из которых строится приложение – Java-классы, Java-библиотеки, HTML-страницы, изображения, текстовые файлы, наборы SQL-скриптов и т. д. Строго говоря, сам проект также является ресурсом. Физически каждый ресурс проекта представлен каталогами и файлами соответствующего типа.

Существует несколько способов создания проектов и ресурсов проекта. Основным из них является команда основного меню File/New. С помощью подпункта Other можно отобразить все типы ресурсов, доступных для создания. Для удобства выбора типы создаваемых проектов и ресурсов делятся на условные категории, обычно в зависимости от типа разрабатываемого приложения. Например, в категории Java размещаются обычные Java-проекты, Java-классы, интерфейсы, пакеты и т. д., а в категории Web – Web-проекты, HTML-страницы, JSP-страницы и т. д.

1) Закройте перспективу Welcome, появившуюся на экране после первого запуска Eclipse.
2) Выберите в меню File/New/Other.
3) Выберите категорию Java, а в ней Java Project и нажмите Next.

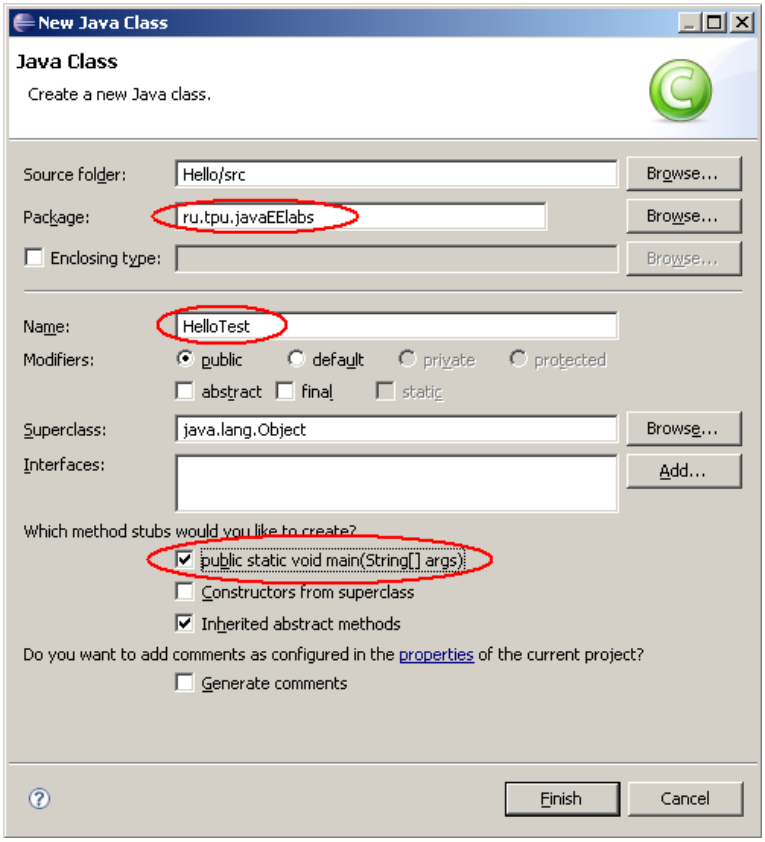


4) Введите имя проекта – Hello. Затем, если в разделе JRE не указана или указана неверная настройка JRE, выберите Configure default… и добавьте путь к JDK, установленной на предыдущем этапе (например, C:\jdk1.5.0_12).



5) Нажмите Finish.
6) Eclipse предложит открыть перспективы, связанные с разработкой Java-приложения. Выберите флажок Remember my decision и нажмите Yes.
7) В результате создания проекта автоматически открывается перспектива Java, в левой части окна открывается представление Package Explorer, в котором отражается новый проект Hello, содержащий подкаталог src, предназначенный для хранения исходных файлов программы.
8) Далее, нам необходимо добавить новый ресурс проекта – обычный Java-класс:
9) В представлении Package Explorer нажмите правой кнопкой мыши на значок проекта Hello и выберите New/Class. То же самое действие можно выполнить с помощью команды File/New/Other, затем выбрав категорию Java и тип ресурса Class.
10) В появившемся окне укажите имя пакета и имя нового класса. Имя пакета может состоять из нескольких уровней, разделенных точкой. Также выберите флажок public static void main(…) для автогенерации `main`-метода.

11) Нажмите Finish



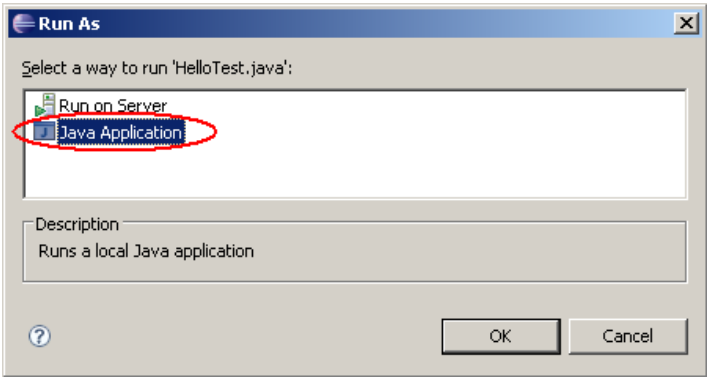
12) В методе `main` запишите команду вывода сообщения на экран:

```
public static void main(String[] args) {
   System.out.println("Hello Java");
}
```

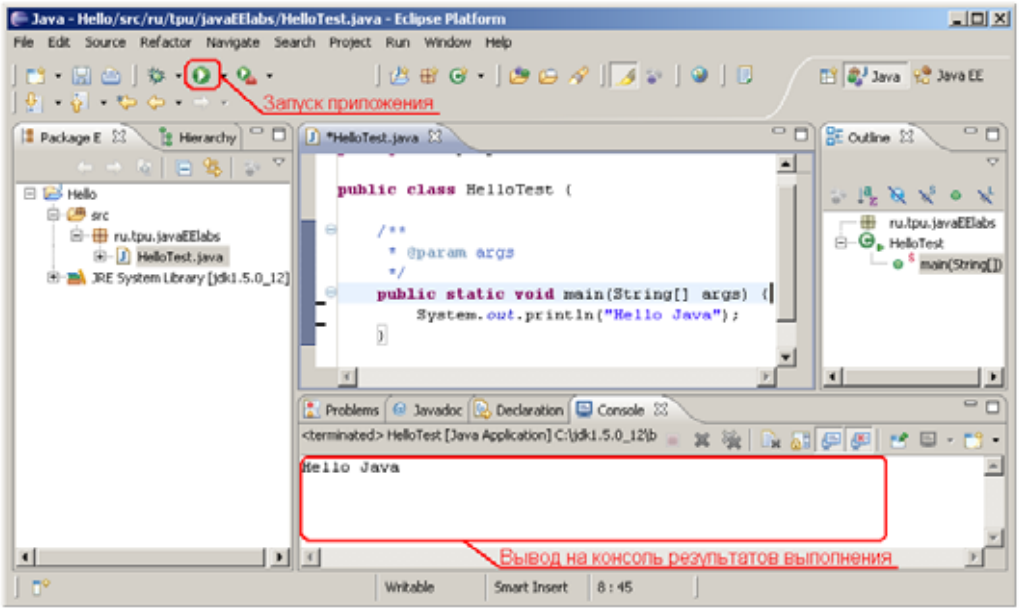
13) Сохраните файл, нажав на клавиши Ctrl-S.

5. Запуск приложения

1) Для запуска приложения выделите файл Hello.java в представлении Package Explorer, выберите команду меню Run/Run или нажмите на соответствующую кнопку панели инструментов. Запустить приложение можно также нажав правой кнопкой мыши на Hello.java в представлении Package Explorer и выбрав команду Run As/Java Application.
2) В появившемся окне Run As… выберите Java Application и нажмите OK.



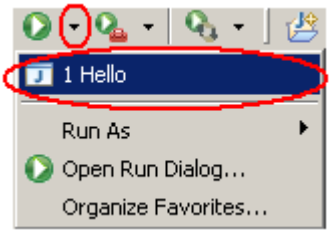
Если появилось окно сохранения файлов перед запуском Save and Launch, выберите флажок Always save resources… и нажмите OK.



3) После запуска приложения в нижней части экрана в окне Console отображается результат выполнения программы.

При первом запуске приложения создается конфигурация запуска, которая используется при последующих запусках программы. В дальнейшем, при выполнении лабораторных работ, рабочая область будет содержать несколько проектов, и соответственно несколько Java-классов, содержащих метод `main()`. Для того, чтобы выбрать приложение для запуска, следует выделить соответствующий java-файл в представлении Package Explorer и выполнить Run/Run, либо нажав правой кнопкой мыши на java-файл в представлении Package Explorer и выполнить команду Run As/Java Application. Для повторных запусков приложения можно нажать на кнопку Run панели инструментов для запуска

последнего запускавшегося приложения, либо открыть список конфигураций запуска и выбрать другое приложение:

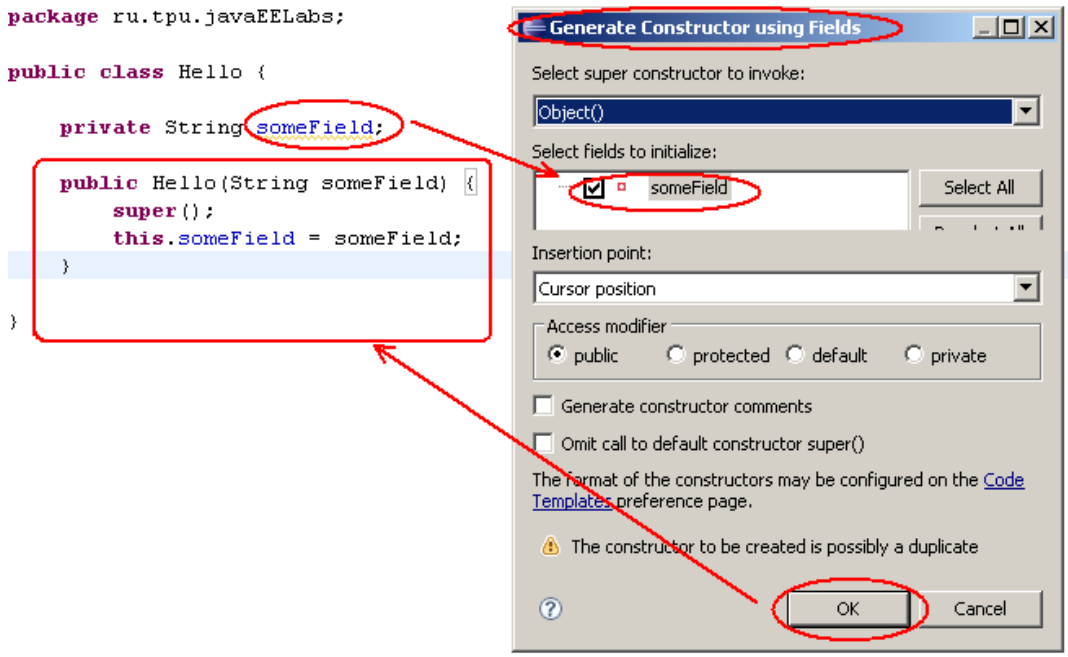


6. Автогенерация и автоформатирование кода

Ниже приведены некоторые полезные функции по генерации стандартных конструкций в Java-коде, а также автоматическое форматирование кода. Функции автогенерации кода сосредоточены в меню Source и начинаются со слова Generate (кроме Override/Implement Methods). Эти функции отрабатывают только на коде активного редактора java-кода.

Override/Implement Methods	переопределить/реализовать метод базового класса/интерфейса
Generate Getters And Setters	создать методы get и/или set для полей класса
Generate Delegate Methods	создать методы-делегаты для методов объекта, определенного в классе
Generate hashCode() and equals()	переопределить и сгенерировать содержимое методов hashCode() и equals()
Generate Constructors using Fields	создать конструктор, принимающий поля класса в качестве параметров
Generate Constructors using Superclass	создать конструктор используя суперкласс

Ниже приведен пример работы команды Generate Constructors using Fields:



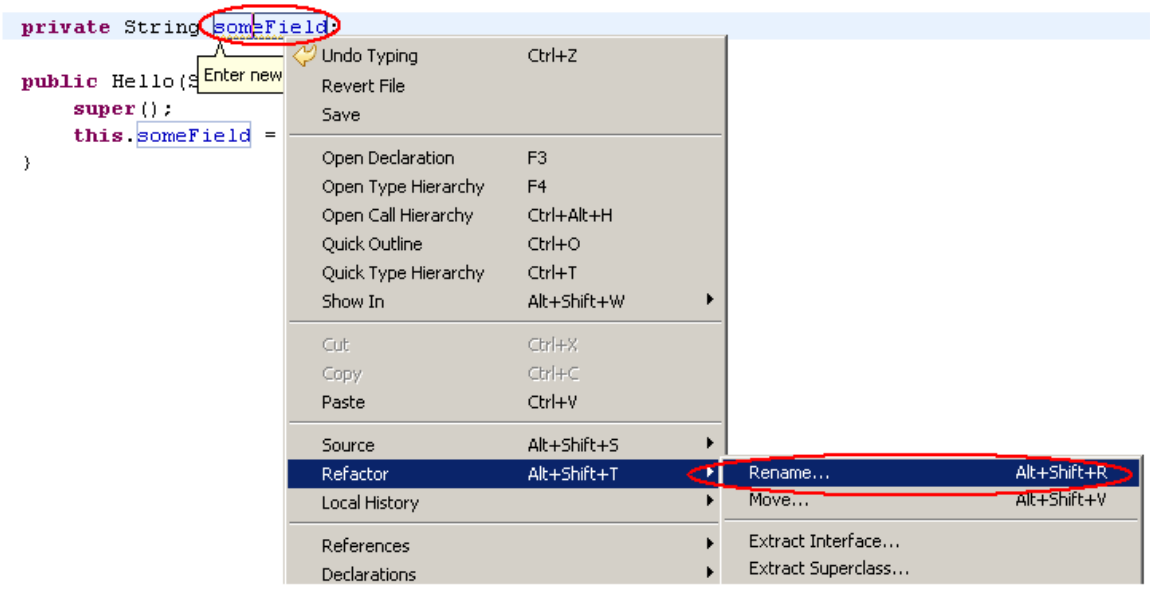
Автоформатирование кода выполняется с помощью команды меню Source/Format, либо с помощью комбинации клавиш Ctrl-Shift-F. Автоформатирование выполняется над текущим файлом открытым в окне редактора.

7. Рефакторинг

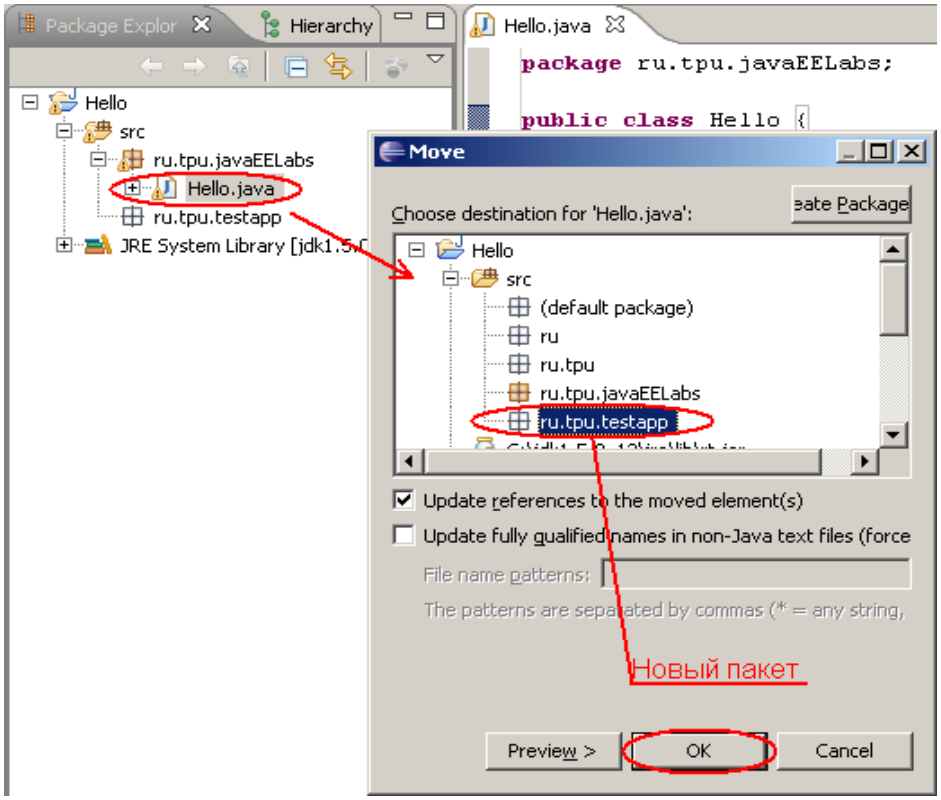
Рефакторинг – это безопасное изменение структуры ресурсов проекта. Действия по рефакторингу включают в себя изменение названий классов, полей и методов, изменение принадлежности классов пакетам, извлечение интерфейсов из описания класса и пр. При выполнении действия рафактор (refactor) обеспечивает сохранение целостности кода. Например, изменение имени класса приводит к изменению этого имени во всех элементах кода, где используется этот класс.

Команды рефакторинга вызываются над определенным элементом – классом, полем, методом и т. д. Функции рафактора (refactor) удобнее всего вызывать путем выбора элемента, вызова щелчком правой кнопкой мыши контекстного меню и выбора в нем одной из команд меню Refactor. Также команды рефакторинга можно вызвать, используя основное меню Refactor.

Ниже приведен пример вызова команды переименования над полем класса. Перед вызовом меню курсор редактора предварительно устанавливается на изменяемое поле `someField`.



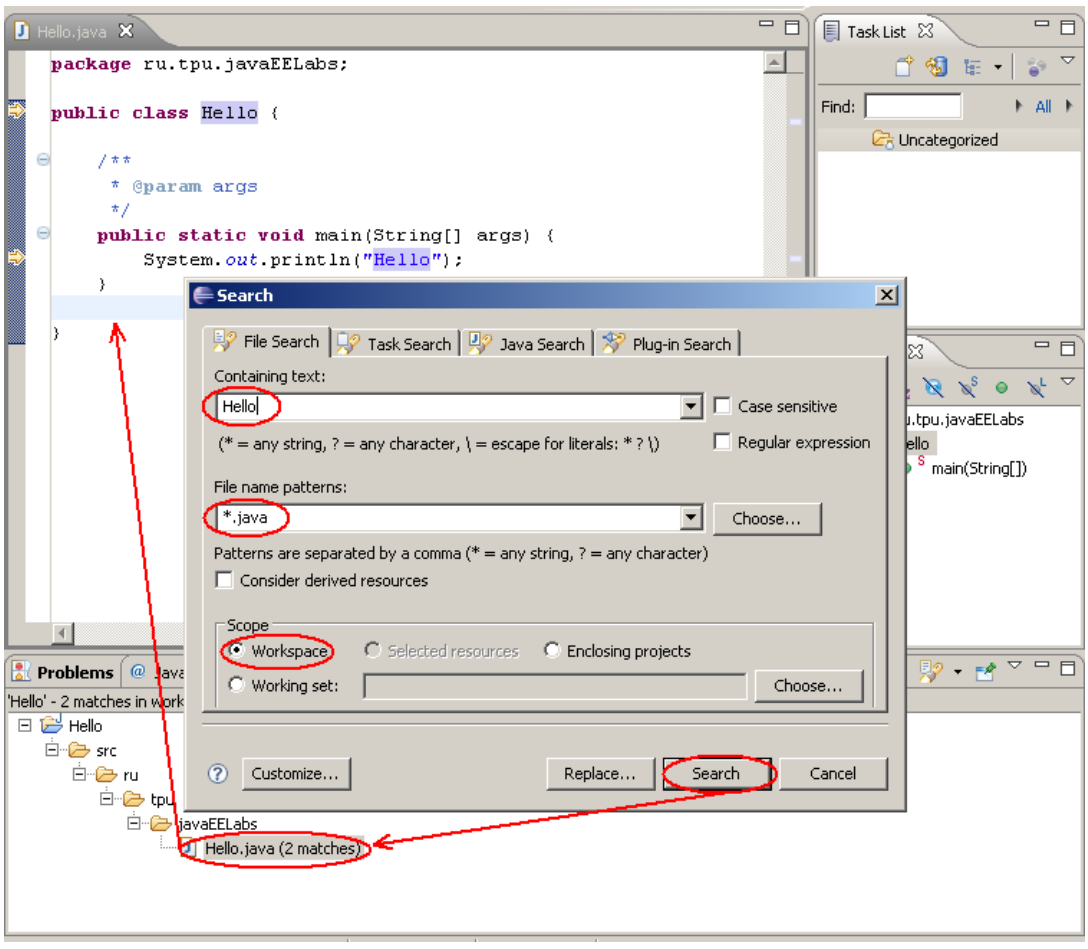
Следующий пример демонстрирует возможность смены пакета, в котором размещается класс. Действие рефактора Move вызывается над классом `Hello` в представлении Package Explorer. В результате выполнения действия класс перемещается из пакета `ru.tpu.javaEELabs` в пакет `ru.tpu.testapp`.



8. Поиск

Eclipse Workbench позволяет производить поиск файлов и строк, а также замену (без соблюдения целостности проекта) одной строки на другую. Рассмотрим пример поиска файлов в рабочей области, содержащих слово “Hello”.

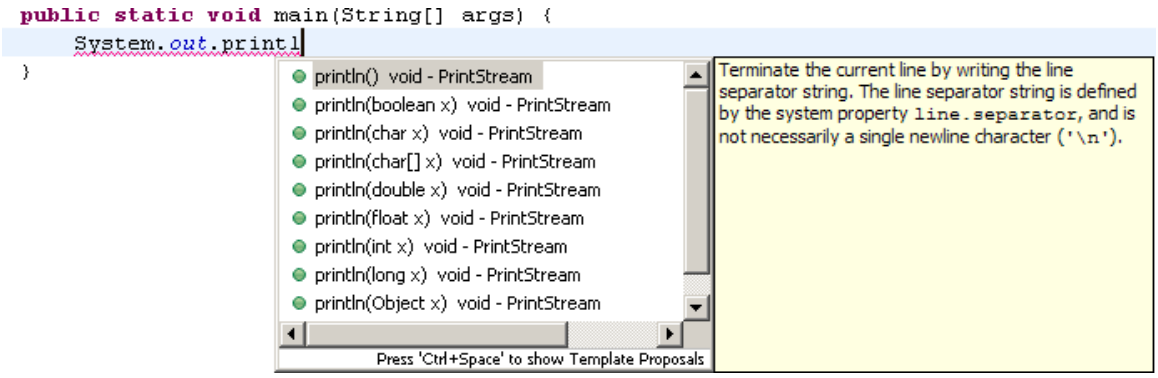
1) Нажмите на кнопку поиска или сочетание клавиш Ctrl-H
2) В появившемся окне вберите закладку File Search и в поле Containing Text введите строку Hello.
3) Опция File Name Patterns определяет шаблон имен файлов для поиска. Изменить шаблон можно путем прямого ввода нового значения в поле, либо с помощью кнопки Choose. По умолчанию выбран шаблон *.java, т. е. все файлы с расширением java.
4) Опция Scope определяет область поиска. Убедитесь, что в этом разделе выбран пункт Workspace (по всей рабочей области).
5) Нажмите Search.
6) В результате выполнения поиска отображается представление Search, в котором показаны результаты поиска. Щелчок на найденный файл открывает его в окне редактора.



10. Основные горячие клавиши

Ниже приведен список некоторых горячих клавиш Eclipse Workbench:

Ctrl-N	создать новый ресурс
Ctrl-S	сохранить файл, открытый в активном редакторе
Ctrl-Shift-S	сохранить все файлы проекта
Ctrl-F11	запуск последней конфигурации
F11	запуск последней конфигурации в режиме отладки
Ctrl-Shift-F	автоформатирование текущего файла
Ctrl-H	вызов окна сложного поиска/замены
Ctrl-F	вызов окна простого текстового поиска/замены по текущему файлу
Ctrl-Shift-O	автоматический импорт класса
Ctrl-Space	вызов контекстной подсказки редактора



Ctrl-щелчок мыши на классе, ссылке или вызове метода осуществляет переход к объявлению этого элемента.

Лабораторная работа 1. Создание приложения для доступа к базе данных с использованием технологии JDBC

Постановка задачи

Необходимо разработать приложение для автоматизации учета товаров на складе. Приложение должно иметь диалоговый пользовательский интерфейс и позволять пользователю просматривать, добавлять, изменять и удалять информацию о товарах из таблицы products. Таблица products должна содержать следующие поля: идентификатор товара (id), описание (description), цена (rate), количество (quantity).

Для решения поставленной задачи необходимо выполнить следующие шаги:

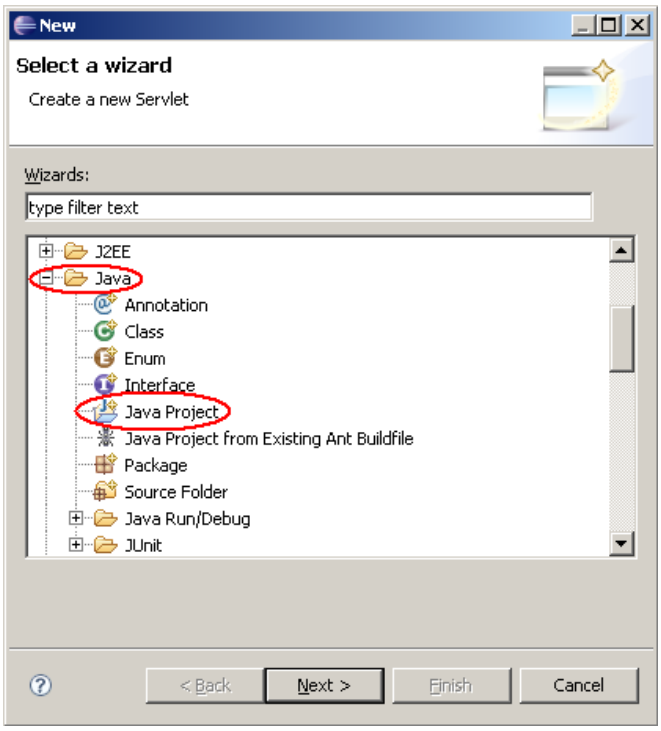
1. Создать новый проект.
2. Создать в БД таблицу `products` для хранения данных о товарах.
3. Реализовать Java-класс `Product` для представления и обработки данных о товаре.
4. Реализовать Java-класс обработки данных `ProductDAO`, в котором обеспечить получение соединения БД и реализовать методы выборки, создания, обновления и удаления товара.
5. Разработать графический интерфейс приложения (Swing) и обеспечить вызов методов обработки данных.
6. Запустить и проверить работу приложения.

Подготовительный этап

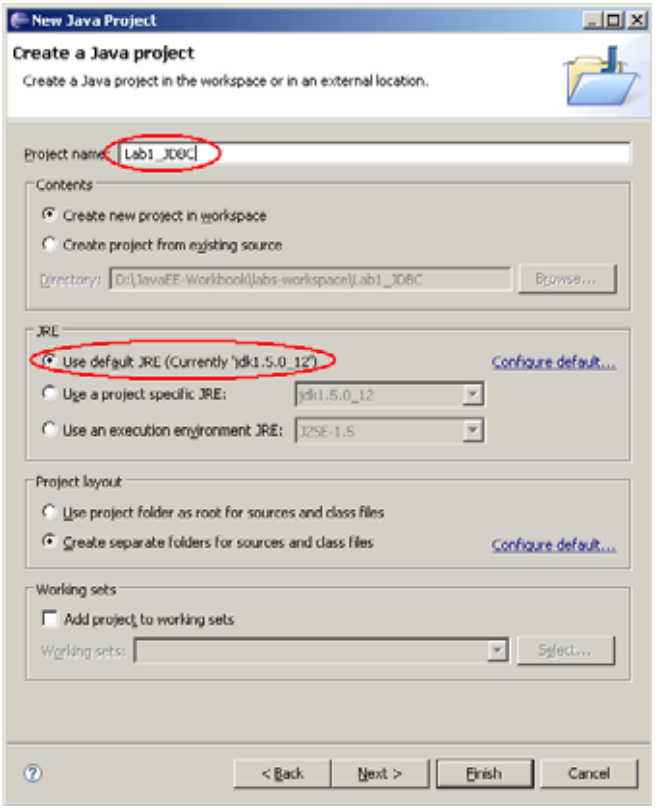
Для реализации проекта необходимо установить и настроить среду разработки Ecplise, Apache Derby и Derby Plugins (см. п. «Установка и настройка программного обеспечения»).

Создание нового проекта

1) Выберите пункт меню File/New/Project, в окне выбора типа проекта укажите other/Java Project и нажмите Next.

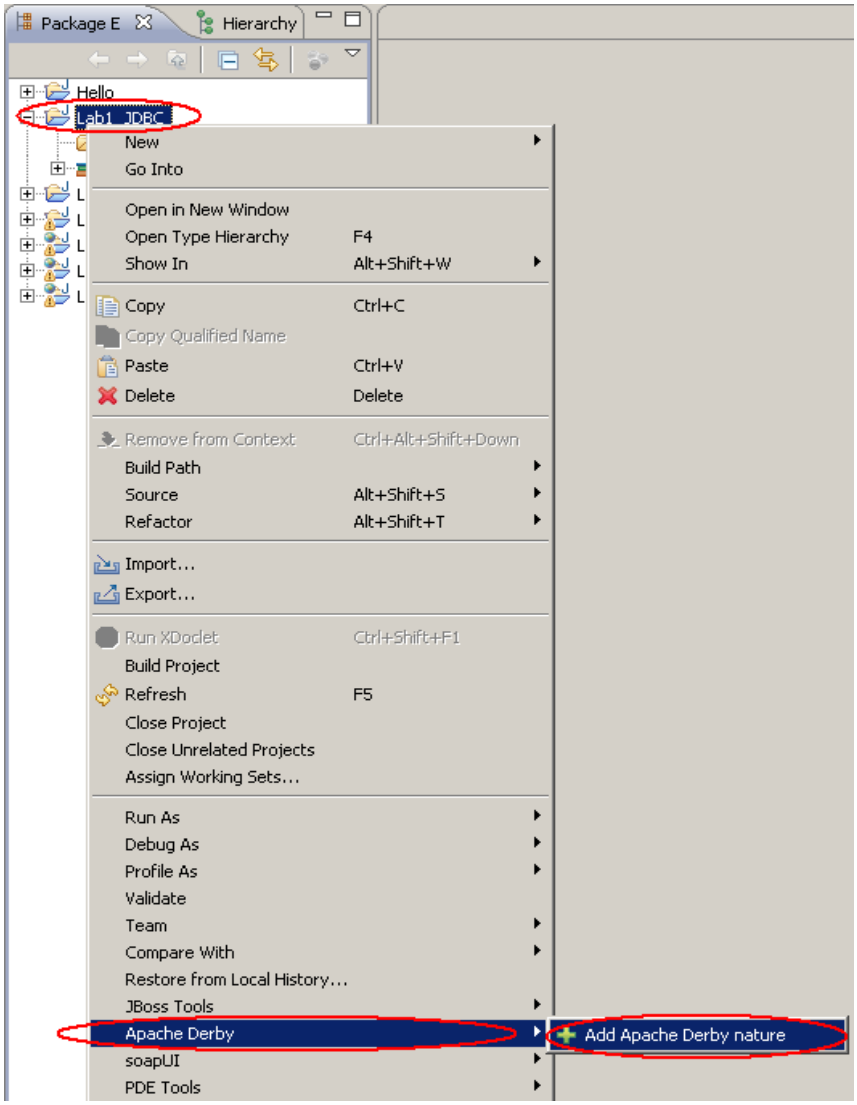


2) Укажите имя проекта Lab1_JDBC, проверьте параметры подраздела JRE – должна быть указана JRE 1.5. Нажмите Finish.

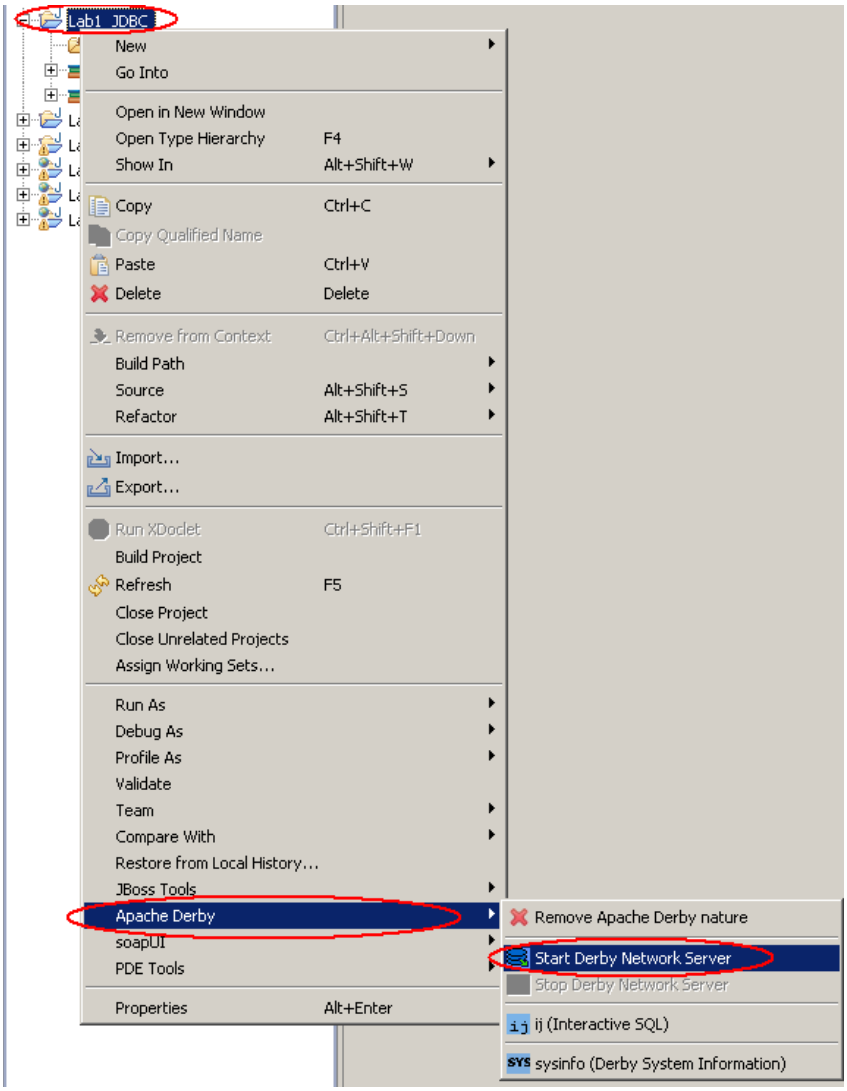


Создание таблицы `products`

1) Подключите к проекту окружение БД Derby и запустите сервер БД. Для этого, щелкните правой кнопкой мыши на созданный проект в представлении Package Explorer и выберите пункт меню Apache Derby/Add Apache Derby Nature.



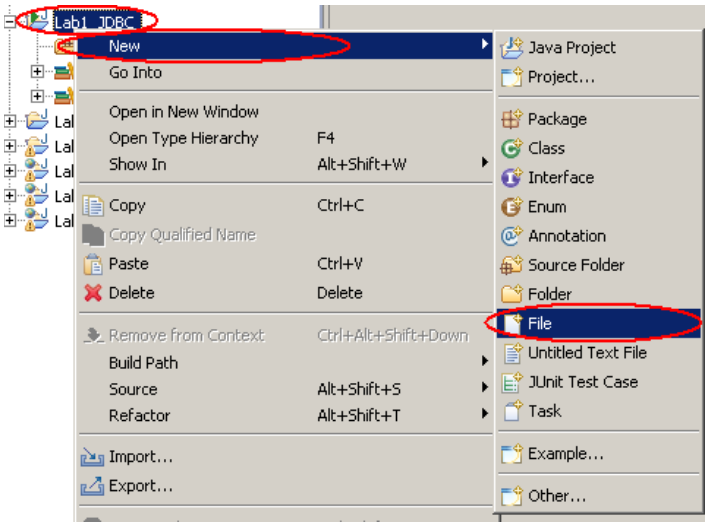
Затем повторно щелкните правой кнопкой мыши на проект и выберите Apache Derby/Start Derby Network Server



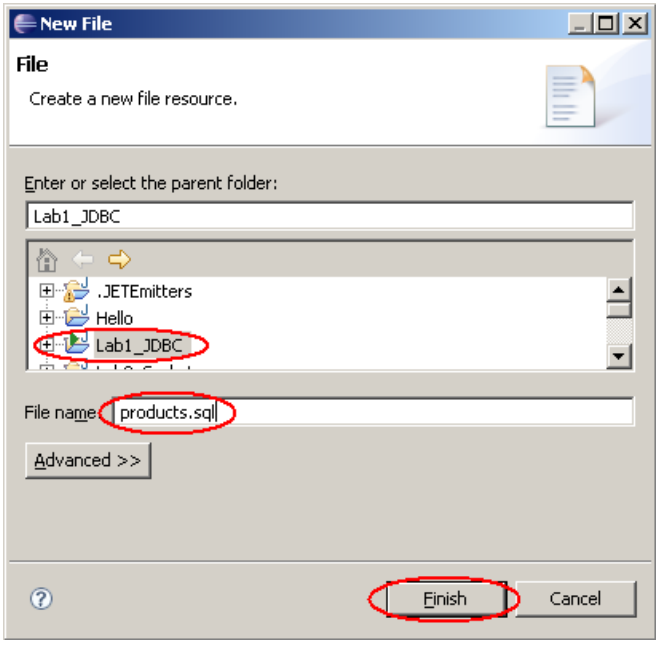
В случае успешного запуска сервера БД в консоли (представление Concole) появляется следующее сообщение:

```
DRDA_SecurityInstalled.I
Сетевой сервер Apache Derby Network Server - 10.3.2.1 - (599110) запущен и готов принимать соединения на порту 1527
```

2) Для хранения SQL-скриптов создадим новый файл products.sql. В окне Package Explorer щелкните правой кнопкой мыши на значок проекта и выберите New/File.



3) В появившемся окне укажите имя файла products.sql и нажмите Finish.

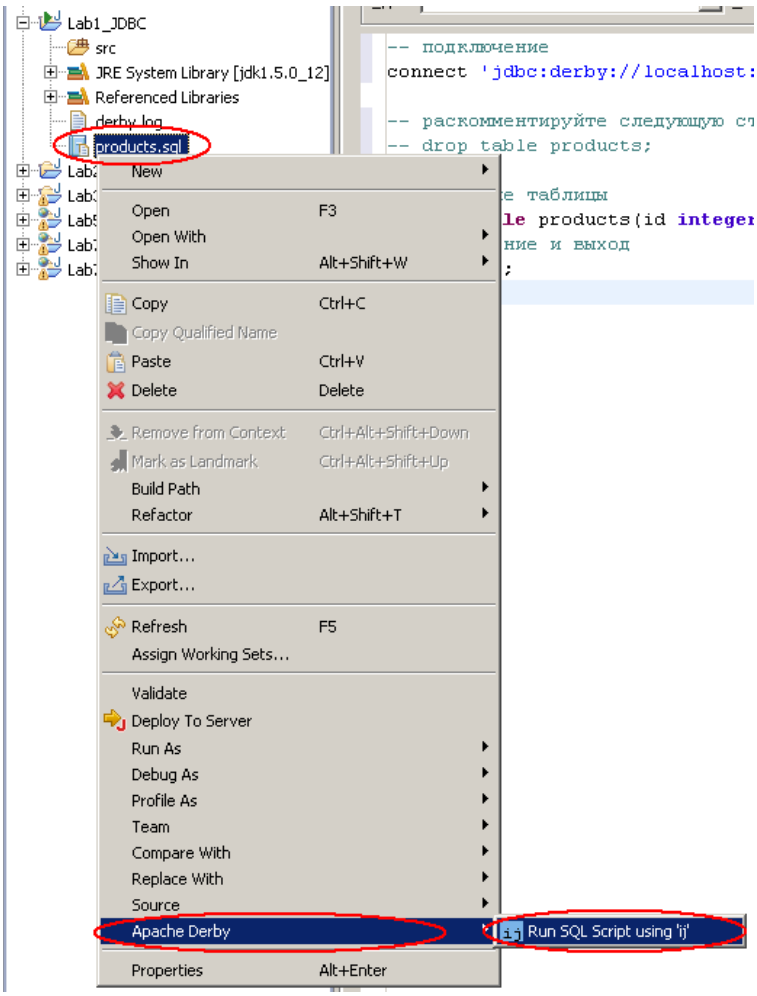


4) SQL-скрипт должен содержать команду создания таблицы `products`. Скопируйте в файл следующие команды:

```
-- подключение
connect 'jdbc:derby://localhost:1527/myDB;
           create=true;user=me;password=mine';

-- раскомментируйте следующую строку, если требуется пересоздать таблицу
-- drop table products;

-- создание таблицы
```

```
create table products(id integer, description varchar(100), rate double,
quantity integer);

-- отключение и выход
disconnect;
exit;
```

5) Сохраните файл нажатием на Ctrl-S.
6) Щелкните правой кнопкой мыши на файл products.sql в окне Package Explorer и выберите Apache Derby/Run SQL Script using ‘ij’.



7) В случае успешного выполнения скрипта в консоли выводится следующее:

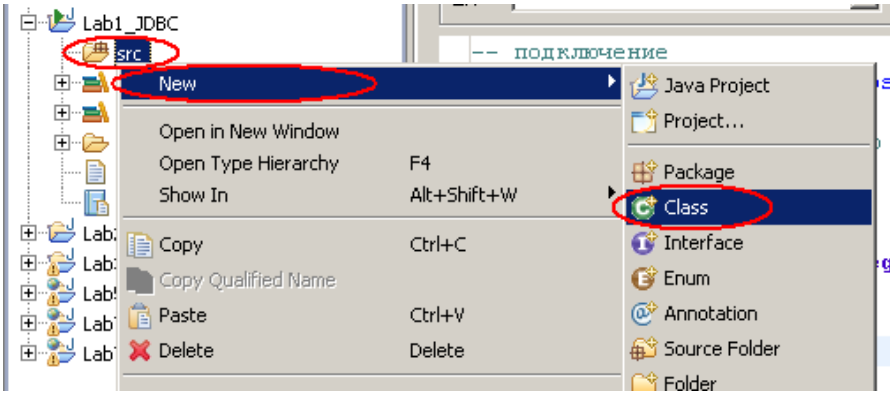
```
ij version 10.3
ij> -- подключение
connect
'jdbc:derby://localhost:1527/myDB;create=true;user=me;password=mine';
```

```
      ij> -- раскомментируйте следующую строку, если требуется пересоз-
дать таблицу
      -- drop table products;

      -- создание таблицы
      create table products(id integer, description varchar(100), rate
double, quantity integer);
      0 rows inserted/updated/deleted
      ij> -- отключение и выход
      disconnect;
      ij> exit;
```

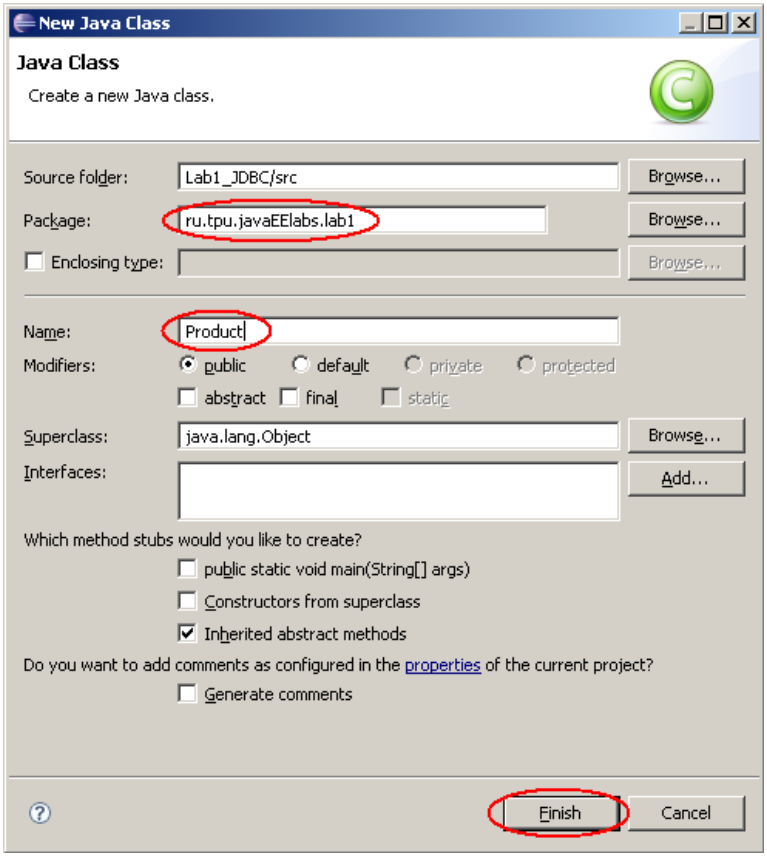
Создание класса Product

Класс Product является обычным Java-классом и будет использоваться для передачи и обработки данных о товаре. Класс содержит четыре поля – `id`, `description`, `rate`, `quantity`, соответствующие полям таблицы `products`.

1) В окне Package Explorer щелкните правой кнопкой мыши на каталог src проекта Lab1_JDBC и выбрав пункт меню New/Class.



2) Назовите класс Product и задайте имя пакета (Package) `ru.tpu.javaEElabs.lab1`. Нажмите Finish.



3) В созданном классе необходимо реализовать четыре поля `id`, `description`, `rate`, `quantity`, создать конструктор и методы get/set для доуступа к этим полям. Полный код класса `Product` приведен ниже:

```
package ru.tpu.javaEElabs.lab1;

public class Product {

private int id;
private String description;
private float rate;
private int quantity;

public Product(int id, String description, float rate, int quantity) {
	this.id = id;
	this.description = description;
	this.rate = rate;
	this.quantity = quantity;
}

public int getId() {
	return id;
}

	public void setId(int id) {
```

```
            this.id = id;
      }

      public String getDescription() {
            return description;
      }

      public void setDescription(String description) {
            this.description = description;
      }

      public float getRate() {
            return rate;
      }

      public void setRate(float rate) {
            this.rate = rate;
      }

      public int getQuantity() {
            return quantity;
      }

      public void setQuantity(int quantity) {
            this.quantity = quantity;
      }
}
```

Создание класса обработки данных `ProductDAO`

На следующем этапе создадим класс `ProductDAO` (DAO – Data Access Object – объект доступа к данным), который скрывает детали реализации логики работы с БД – получение соединения, выполнение операций выборки, создания, обновления и удаления.

1) В окне Package Explorer щелкните правой кнопкой мыши на пакет `ru.tpu.javaEElabs.lab1` проекта Lab1_JDBC и выбрав пункт меню New/Class.
2) Назовите класс ProductDAO и убедитесь, что имя пакета (Package) задано в `ru.tpu.javaEElabs.lab1`. Нажмите Finish.

   Код класса `ProductDAO` приведен ниже:

```
package ru.tpu.javaEElabs.lab1;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.List;

public class ProductDAO {
      // Вспомогательный метод получения соединения
      private Connection getConnection() throws Exception {
```

```
            // Подгрузка драйвера БД Derby
      Class.forName("org.apache.derby.jdbc.ClientDriver").newInstance();
            // Получение соединения с БД
            return DriverManager.getConnection(
"jdbc:derby://localhost:1527/myDB;create=true;user=me;password=mine");
      }

      /**
       * Возвращает список идентификаторов товаров
       *
       * @return
       */
      public List<Integer> getProductIds() throws Exception {
            List<Integer> productIds = new ArrayList<Integer>();
            // Получение соединения с БД
            Connection con = getConnection();

            // Выполнение SQL-запроса
            ResultSet rs = con.createStatement().executeQuery(
                        "Select id From products");
            // Перечисляем результаты выборки
            while (rs.next()) {
                  // Из каждой строки выборки выбираем
                  // результат и помещаем в коллекцию
                  productIds.add(rs.getInt(1));
            }
            // Закрываем выборку и соединение с БД
            rs.close();
            con.close();
            return productIds;
      }

      /**
       * Возвращает товар по идентификатору
       *
       * @return
       */
      public List<Product> getProductById(int id) throws Exception {
            List<Product> products = new ArrayList<Product>();
            // Получение соединения с БД
            Connection con = getConnection();

            // Подготовка SQL-запроса
            PreparedStatement st = con.prepareStatement(
                              "Select description, rate, quantity " +
                              "From products " +
                              "Where id = ?");
            // Указание значений параметров запроса
            st.setInt(1, id);

            // Выполнение запроса
            ResultSet rs = st.executeQuery();

            Product product = null;
            // Перечисляем результаты выборки
            while (rs.next()) {
                  // Из каждой строки выборки выбираем результаты,
                  // формируем новый объект Product
```

```
            // и помещаем его в коллекцию
            product = new Product(
                        id,
                        rs.getString(1),
                        rs.getFloat(1),
                        rs.getInt(1));
            products.add(product);
      }
      // Закрываем выборку и соединение с БД
      rs.close();
      con.close();
      return products;
}

/**
 * Добавляет новый товар
 * @param product
 * @throws Exception
 */
public void addProduct(Product product) throws Exception {
      // Получение соединения с БД
      Connection con = getConnection();

      // Подготовка SQL-запроса
      PreparedStatement st = con.prepareStatement(
                  "Insert into products " +
                  "(id, description, rate, quantity) " +
                  "values (?, ?, ?, ?)");
      // Указание значений параметров запроса
      st.setInt(1, product.getId());
      st.setString(2, product.getDescription());
      st.setFloat(3, product.getRate());
      st.setInt(4, product.getQuantity());

      // Выполнение запроса
      st.executeUpdate();

      con.close();
}

/**
 * Обновляет данные о товаре
 * @param product
 * @throws Exception
 */
public void setProduct(Product product) throws Exception {
      // Получение соединения с БД
      Connection con = getConnection();

      // Подготовка SQL-запроса
      PreparedStatement st = con.prepareStatement(
                  "Update products " +
                  "Set description=?, rate=?, quantity=? " +
                  "Where id=?");
      // Указание значений параметров запроса
      st.setString(1, product.getDescription());
      st.setFloat(2, product.getRate());
      st.setInt(3, product.getQuantity());
```

```
            st.setInt(4, product.getId());

            // Выполнение запроса
            st.executeUpdate();
            con.close();
      }

      public void removeProduct(int id) throws Exception {
            // Получение соединения с БД
            Connection con = getConnection();

            // Подготовка SQL-запроса
            PreparedStatement st = con.prepareStatement(
                        "Delete from products " +
                        "Where id = ?");
            // Указание значений параметров запроса
            st.setInt(1, id);

            // Выполнение запроса
            st.executeUpdate();
            con.close();
      }
}
```

Разработка клиентского приложения

Клиентское приложения для просмотра/редактирования товаров будет представлять собой диалоговое приложение Swing, содержащее список для выбора товара по идентификатору и полей, отображающих информацию по выбранному товару. Эти же поля будут использоваться для ввода/редактирования информации по добавляемому/изменяемому товару. Ниже будут располагаться кнопки «Добавить», «Обновить» и «Удалить». Обработчики нажатия на эти кнопки обращаются к полям формы и методам класса `ProductDAO` для выполнения соответствующих действий.

1) Создайте новый Java-класс с именем ProductInfoClient в пакете `ru.tpu.javaEElabs.lab1`. Класс наследует `JDialog`.

   Код класса `ProductInfoClient` приведен ниже:

```
package ru.tpu.javaEElabs.lab1;

import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.List;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JTextField;

/**
```

```
 * Диалог для просмотра/обработки данных о товарах
 */
public class ProductInfoClient extends JDialog  {

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;

	// Создаем DAO-объект
	ProductDAO productDAO = new ProductDAO();

	// Объявление элементов управления
	private JLabel lbSelectId = new JLabel("Выбор товара по Id");
	private JLabel lbId = new JLabel("Id");
	private JLabel lbDescription = new JLabel("Описание");
	private JLabel lbRate = new JLabel("Цена");
	private JLabel lbQuantity = new JLabel("Остаток");

	private JComboBox comboId = new JComboBox();
	private JTextField txtId = new JTextField();
	private JTextField txtDescription = new JTextField();
	private JTextField txtRate = new JTextField();
	private JTextField txtQuantity = new JTextField();

	private JButton btnAdd = new JButton("Добавить");
	private JButton btnUpdate = new JButton("Обновить");
	private JButton btnRemove = new JButton("Удалить");
	private JButton btnClear = new JButton("Очистить");


	/**
	 * Создает экземпляр диалога
	 * @param args
	 */
	public static void main(String[] args) {
		new ProductInfoClient();
	}
	/**
	 * Конструктор диалога
	 */
	public ProductInfoClient() {
		this.setTitle("Информация о товарах");
		this.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
		this.setLayout(new GridLayout(8, 2));
		this.setBounds(100, 50, 400, 200);

		// Добавление элементов управления в диалог
		this.add(lbSelectId);
		this.add(comboId);
		this.add(lbId);
		this.add(txtId);
		this.add(lbDescription);
		this.add(txtDescription);
		this.add(lbRate);
		this.add(txtRate);
		this.add(lbQuantity);
		this.add(txtQuantity);
```

```
        this.add(btnAdd);
        this.add(btnUpdate);
        this.add(btnRemove);
        this.add(btnClear);

        // Описание обработчиков событий
        comboId.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent e) {
                    showProductData();
              }
        });

        btnAdd.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent e) {
                    addProduct();
              }
        });

        btnUpdate.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent e) {
                    updateProduct();
              }
        });

        btnRemove.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent e) {
                    removeProduct();
              }
        });

        btnClear.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent e) {
                    clearProductInfo();
              }
        });

        // Обновляем список идентификаторов товаров
        refreshIdList();

        // Отображаем диалог на экране
        this.setVisible(true);
  }

  /**
   * Считывает список идентификаторов товаров
   * и заполняет список
   */
  private void refreshIdList() {
        try {
              // получаем список идентификаторов
              List<Integer> productIds = productDAO.getProductIds();
              // очищаем список
              comboId.removeAllItems();
              // заполняем список полученными данными
              for (Integer productId: productIds) {
                    comboId.addItem(productId);
              }
        } catch (Exception e) {
```

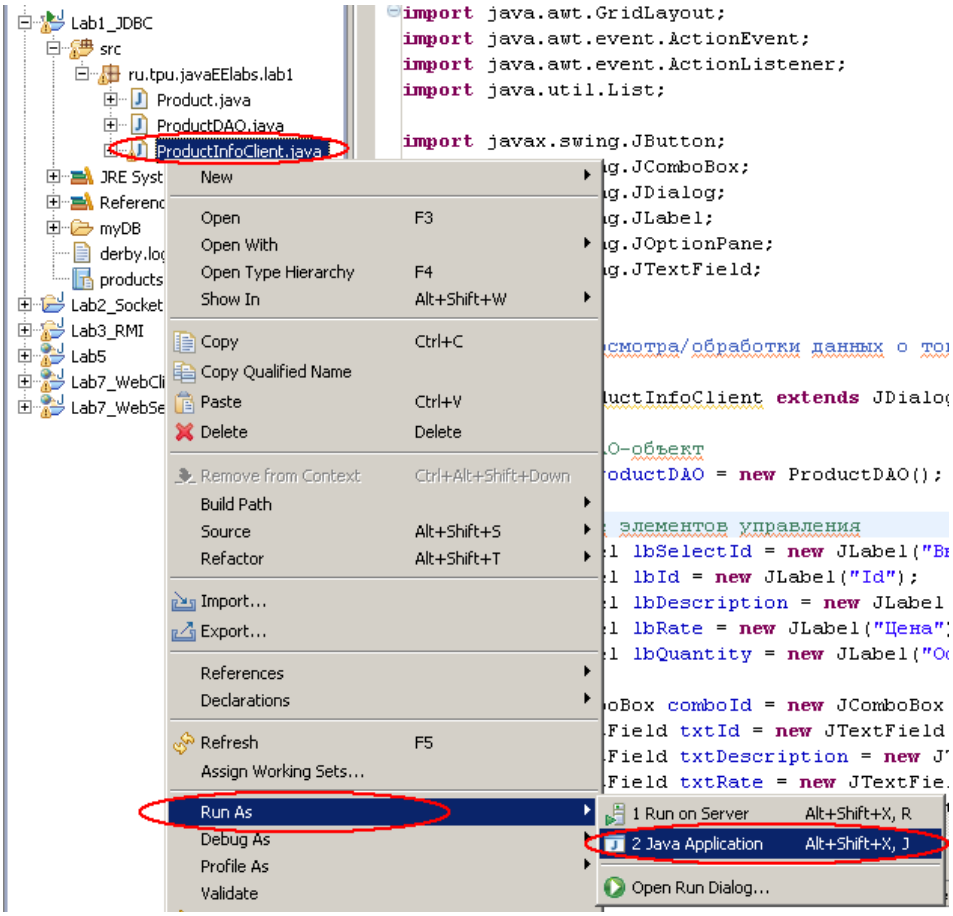
```
                    e.printStackTrace();
                    JOptionPane.showMessageDialog(this, e.getMessage());
                }
        }

        /**
         * Отображает данные о товаре по
         * выбранному в списке идентификатору
         */
        protected void showProductData() {
          try {
                // забираем значение, выбранное в списке
// идентификаторов товаров
                Integer productId = (Integer)comboId.getSelectedItem();

                if (productId != null) {
                    // получаем товар по идентификатору

                    List<Product> product = product-
DAO.getProductById(productId);
                    // заполняем текстовые поля значениями
// параметров товара
                    for(int i=0; i<product.size(); i++){
                          txt-
Id.setText(String.valueOf(product.get(i).getId()));
                          txtDescrip-
tion.setText(product.get(i).getDescription());

        txtRate.setText(String.valueOf(product.get(i).getRate()));
                          txtQuan-
tity.setText(String.valueOf(product.get(i).getQuantity()));
                    }
                }
          } catch (Exception e) {
                e.printStackTrace();
                JOptionPane.showMessageDialog(this, e.getMessage());
          }
        }

        /**
         * Добавляет новый товар на основе
         * данных текстовых полей
         */
        protected void addProduct() {
                try {
                    // создаем новый объект-товар
                    // на основе данных диалога
                    Product product = new Product(
                          Integer.parseInt(txtId.getText()),
                          txtDescription.getText(),
                          Float.parseFloat(txtRate.getText()),
                          Integer.parseInt(txtQuantity.getText()));
                    // сохраняем товар в БД
                    productDAO.addProduct(product);
                    // обновляем список идентификаторов
                    refreshIdList();
                    // устанавливаем текущим добавленный товар
comboId.setSelectedItem(
```

```
Integer.parseInt(txtId.getText()));
            } catch (Exception e) {
                  e.printStackTrace();
                  JOptionPane.showMessageDialog(this, e.getMessage());
            }
      }

      /**
       * Обновляет информацию о товаре на основе
       * данных текстовых полей
       */
      protected void updateProduct() {
            try {
                  // формируем объект-товар
                  // на основе данных диалога
                  Product product = new Product(
                              Integer.parseInt(txtId.getText()),
                              txtDescription.getText(),
                              Float.parseFloat(txtRate.getText()),
                              Integer.parseInt(txtQuantity.getText()));
                  // обновляем данные о товаре в БД
                  productDAO.setProduct(product);
            } catch (Exception e) {
                  e.printStackTrace();
                  JOptionPane.showMessageDialog(this, e.getMessage());
            }
      }

      /**
       * Удаляет выбранный товар
       */
      protected void removeProduct() {
            try {
                  // удаляем товар из БД
                  productDAO.removeProduct(
Integer.parseInt(txtId.getText()));
                  // обновляем список идентификаторов товаров
                  refreshIdList();
                  // отображаем данные по первому товару в списке
                  showProductData();
            } catch (Exception e) {
                  e.printStackTrace();
                  JOptionPane.showMessageDialog(this, e.getMessage());
            }
      }

      /**
       * Очищает данные в тектовых полях
       */
      protected void clearProductInfo() {
            try {
                  txtId.setText("");
                  txtDescription.setText("");
                  txtRate.setText("");
                  txtQuantity.setText("");
            } catch (Exception e) {
                  e.printStackTrace();
                  JOptionPane.showMessageDialog(this, e.getMessage());
```

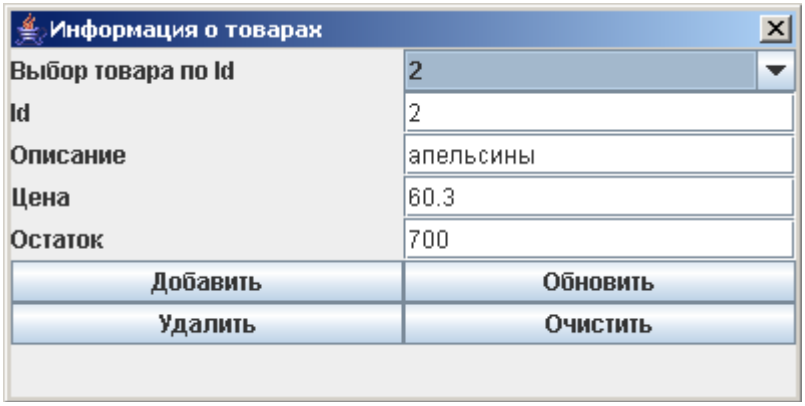
```
            }
        }
}
```

Запуск и тестирование приложения

1) Щелкните правой кнопкой мыши на класс ProductInfoClient в окне Package Explorer и выберите команду Run As/Java Application.



2) Проверьте работу приложения – добавьте несколько товаров, проверьте функции обновления и удаления.



Варианты заданий

1. Необходимо разработать приложение для учета компакт-дисков в музыкальном салоне. Приложение должно позволять пользователю просматривать, добавлять, изменять и удалять информацию о дисках. Описание диска должно содержать следующие поля: идентификатор (id), жанр, исполнитель, название, год. Необходимо обеспечить отображение списка дисков в виде таблицы и возможность просмотра дисков по жанру, выбранному пользователем из выпадающего списка.

2. Необходимо разработать приложение для учета книг в книжном магазине. Приложение должно позволять пользователю просматривать, добавлять, изменять и удалять информацию о книгах. Описание книги должно содержать следующие поля: идентификатор (id), жанр, ФИО автора, название, год. Необходимо обеспечить отображение списка книг в виде таблицы и возможность фильтрации книг по ФИО автора, введенной пользователем в текстовом поле.

Лабораторная работа 2. Отправка и прием сообщений с использованием протоколов UDP и TCP/IP

Часть 1. Создание приложения UDP

Постановка задачи

Необходимо разработать клиент/серверное приложение, в котором сервер может распространять сообщения всем клиентам, зарегистрированным в группе 233.0.0.1, порт 1502. Пользователь сервера должен иметь возможность ввода и отправки текстовых сообщений, а пользователь-клиент просматривает полученные сообщения.

Для решения поставленной задачи необходимо выполнить следующие шаги:

1. Создать новый проект.
2. Реализовать класс сервера для ввода и отправки сообщений.
3. Реализовать класс клиента для получения и просмотра сообщений.
4. Протестировать приложение – запустить сервер и клиент, и отправить сообщение.

Подготовительный этап

Для реализации проекта необходимо установить и настроить среду разработки Ecplise (см. п. «Установка и настройка программного обеспечения»).

Создание нового проекта

1) Выберите пункт меню File/New/Project, в окне выбора типа проекта укажите other/Java Project и нажмите Next.
2) Укажите имя проекта Lab2 и нажмите Finish.

Создание класса Server

Класс `Server` предназначен для отсылки сообщений всем клиентам, зарегистрированным в группе 233.0.0.1. Создание класса Server включает в себя следующие основные задачи:

1. Создание сокета с помощью класса `DatagramSocket`. Сокет сервера выполняет задачу отправки сообщения.
2. Создание объекта `InetAddress`, представляющего адрес сервера. Адреса для групповой (multicast) передачи сообщений выбираются из диапазона 224.0.0.0 – 239.255.255.255. В нашем приложении будет указан адрес 233.0.0.1.

3. Организация ввода строки сообщения с клавиатуры и создание объекта packet класса `DatagramPacket`, который хранит введенные данные и использует метод `send()` объекта класса `DatagramSocket`, для отсылки пакета всем клиентам группы.

1) Для создания класса сервера щелкните правой кнопкой мыши на каталог src в окне Package Explorer и выберите New/Class
2) В появившемся окне в качестве имени пакета (Package) укажите `ru.tpu.javaEELabs.lab2`, а в качестве имени класса (Name) задайте Server. Нажмите Finish.
   Код класса `Server` приведен ниже:

```
package ru.tpu.javaEELabs.lab2;

import java.io.*;
import java.net.*;

public class Server {

private BufferedReader in = null;
private String str = null;
private byte[] buffer;
private DatagramPacket packet;
private InetAddress address;
private DatagramSocket socket;

public Server() throws IOException {
      System.out.println("Sending messages");

// Создается объект DatagramSocket, чтобы
// принимать запросы клиента
      socket = new DatagramSocket();

// Вызов метода transmit(), чтобы передавать сообщение всем
// клиентам, зарегистрированным в группе
      transmit();
}

public void transmit() {
      try {
      // создается входной поток, чтобы принимать
      // данные с консоли
            in = new BufferedReader(new InputStreamReader(System.in));
            while (true) {
                  System.out.println(
                        "Введите строку для передачи клиентам: ");
                  str = in.readLine();
                  buffer = str.getBytes();
                  address = InetAddress.getByName("233.0.0.1");
                  // Посылка пакета датаграмм на порт номер 1502
                  packet = new DatagramPacket(
                        buffer,
                        buffer.length,
```

```
                        address,
                        1502);

                  //Посылка сообщений всем клиентам в группе
                  socket.send(packet);
            }
      } catch (Exception e) {
            e.printStackTrace();
      } finally {
            try {
                  // Закрытие потока и сокета
                  in.close();
                  socket.close();
            } catch (Exception e) {
                  e.printStackTrace();
            }
      }
}

public static void main(String arg[]) throws Exception {
      // Запуск сервера
      new Server();
}
}
```

Создание класса `Client`

Класс `Client` позволяет клиенту присоединиться к группе 233.0.0.1 для получения сообщений от сервера. Создание класса `Client` включает в себя следующие основные задачи:

1. Создание сокета для просмотра групповых сообщений с помощью класса `MulticastSocket`. Сокет клиента выполняет задачу приема сообщения.
2. Создание объекта `InetAddress`, представляющего адрес сервера и присоединение к группе этого сервера с помощью метода сокета `joinGroup`.
3. Организация чтения пакетов датаграмм (DatagramPacket) из сокета и отображение полученных данных на экране.

1) Для создания класса клиента щелкните правой кнопкой мыши на пакет `ru.tpu.javaEELabs.lab2` в каталоге src окна Package Explorer и выберите New/Class.
2) В появившемся окне в качестве имени класса (Name) задайте Client. Нажмите Finish.
   Код класса `Client` приведен ниже:

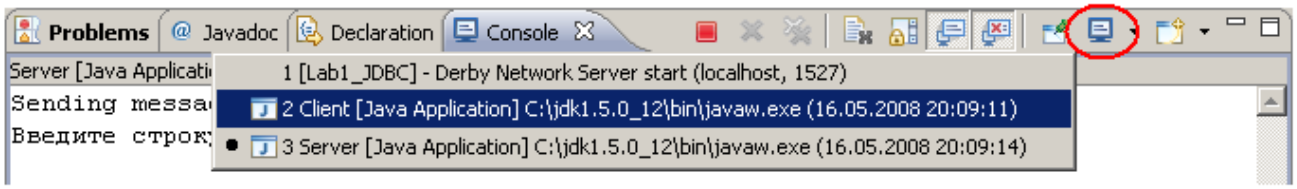
```
package ru.tpu.javaEELabs.lab2;

import java.net.*;
```

```
public class Client {
      private static InetAddress address;
      private static byte[] buffer;
      private static DatagramPacket packet;
      private static String str;
      private static MulticastSocket socket;

      public static void main(String arg[]) throws Exception {
            System.out.println("Ожидание сообщения от сервера");
            try {

                  // Создание объекта MulticastSocket, чтобы получать
                  // данные от группы, используя номер порта 1502
                  socket = new MulticastSocket(1502);

                  address = InetAddress.getByName("233.0.0.1");

                  // Регистрация клиента в группе
                  socket.joinGroup(address);
                  while (true) {
                        buffer = new byte[256];
                        packet = new DatagramPacket(
                              buffer, buffer.length);
                        // Получение данных от сервера
                        socket.receive(packet);
                        str = new String(packet.getData());
                        System.out.println(
                              "Получено сообщение: " + str.trim());
                  }
            } catch (Exception e) {
                  e.printStackTrace();
            } finally {
                  try {
                        // Удаление клиента из группы
                        socket.leaveGroup(address);

                        // Закрытие сокета
                        socket.close();
                  } catch (Exception e) {
                        e.printStackTrace();
                  }
            }
      }
}
```

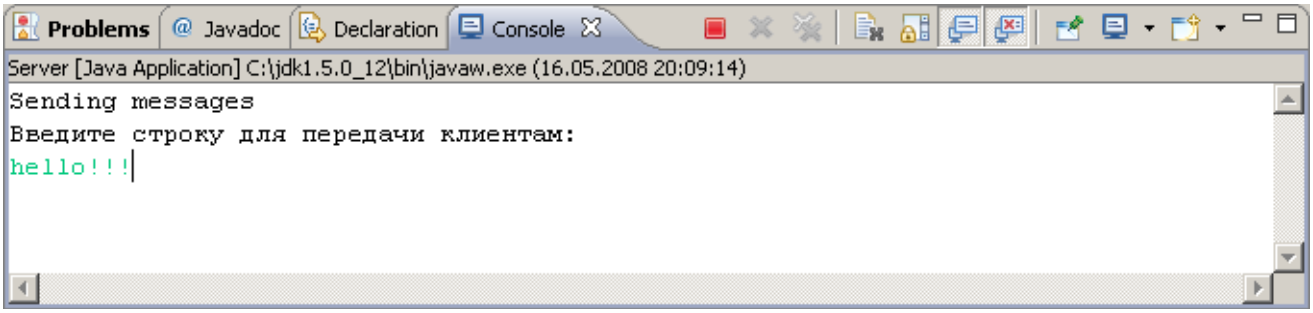
Запуск и тестирование

Каждый из построенных классов `Client` и `Server` содержит метод `main()` и является, по сути, отдельным приложением, которое может быть запущено на отдельной машине, подключенной к сети, при этом по умолчанию область видимости передачи групповых сообщений (multicasting scope) ограничивается подсетью сервера. В нашем случае роль клиента и сервера будет выполнять один и тот же компьютер.

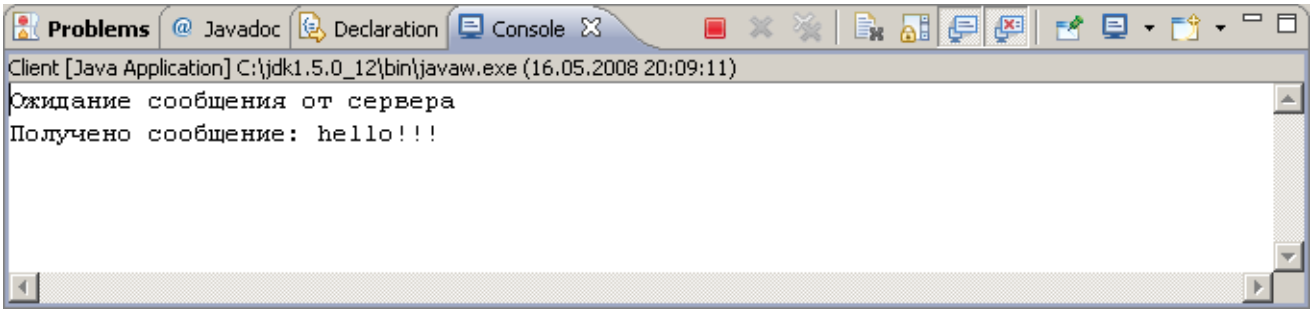
1) Щелкните правой кнопкой мыши на класс Client в окне Package Explorer и выберите команду Run As/Java Application. Проделайте то же самое с классом Server.
2) В результате будут запущены два приложения, переключаться между которыми можно с помощью кнопки-списка Display Selected Console представления Console:



3) Выберите консоль сервера, введите строку hello и нажмите Enter для подтверждения отправки.



4) Просмотрите консоль клиента и убедитесь, что клиент успешно приял сообщение.



5) Остановка приложения осуществляется с помощью кнопки Terminate представления Console.

Часть 2. Создание приложения TCP/IP

Постановка задачи

Необходимо разработать клиент/серверное приложение, в котором сервер слушает запросы клиентов на порт 1500 и отправляет объект-сообщение содержащий текущую дату/время сервера и строку сообщения. Пользователь-клиент должен иметь возможность просмотра полученного сообщения.

Для решения поставленной задачи необходимо выполнить следующие шаги:

1. Создать класс `DateMessage` с двумя полями: дата и строка – для хранения и передачи сообщения клиенту.
2. Реализовать класс сервера для прослушивания соединений на порту 1500 и отправки сообщений. Задача класса сервера должна выполняться в отдельном потоке.
3. Реализовать класс клиента для получения и просмотра сообщений
4. Протестировать приложение – запустить сервер и клиент, и проверить передачу и получение сообщения.

Подготовительный этап

Для реализации проекта необходимо установить и настроить среду разработки Ecplise (см. п. «Установка и настройка программного обеспечения»).

Создание класса `DateMessage`

1) Создайте новый Java-класс `DateMessage` в пакете `ru.tpu.javaEELabs.lab2`.
2) Скопируйте следующее содержимое класса:

```
package ru.tpu.javaEELabs.lab2;

import java.io.Serializable;
import java.util.Date;

public class DateMessage implements Serializable {

      private Date date;
      private String message;

      public DateMessage(Date date, String message) {
            this.date = date;
```

```
            this.message = message;
      }
      public Date getDate() {
            return date;
      }
      public void setDate(Date date) {
            this.date = date;
      }
      public String getMessage() {
            return message;
      }
      public void setMessage(String message) {
            this.message = message;
      }
}
```

Создание класса `ServerTCP`

Создание класса `ServerTCP` включает в себя следующие основные задачи:

1. Создание серверного сокета с помощью класса `ServerSocket`.
2. Ожидание запроса от клиента с помощью метода `accept()` серверного сокета.
3. Формирование объекта-сообщения и отправка его с помощью выходного потока клиентского сокета.

1) Создайте новый Java-класс `ServerTCP` в пакете `ru.tpu.javaEELabs.lab2`.
   Код класса `Server` приведен ниже:

```
package ru.tpu.javaEELabs.lab2;

import java.io.ObjectOutputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Calendar;

/**
 * Класс сервера (выполняется в отдельном процессе)
 */
public class ServerTCP extends Thread {
      // Объявляется ссылка
      // на объект - сокет сервера
      ServerSocket serverSocket = null;
      /**
       * Конструктор по умолчанию
       */
      public ServerTCP() {
            try {
                  // Создается объект ServerSocket, который получает
                  // запросы клиента на порт 1500
                  serverSocket = new ServerSocket(1500);
                  System.out.println("Starting the server ");
```

```
                // Запускаем процесс
                start();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        /**
         * Запуск процесса
         */
        public void run() {
            try {
                while (true) {
                    // Ожидание запросов соединения от клиентов
                    Socket clientSocket = serverSocket.accept();

                    System.out.println("Connection accepted from " +
                    clientSocket.getInetAddress().getHostAddress());

                    // Получение выходного потока,
                    // связанного с объектом Socket
                    ObjectOutputStream out =
                        new ObjectOutputStream(
                            clientSocket.getOutputStream());

                    // Создание объекта для передачи клиентам
                    DateMessage dateMessage = new DateMessage(
                            Calendar.getInstance().getTime(),
                            "Текущая дата/время на сервере");
                    // Запись объекта в выходной поток
                    out.writeObject(dateMessage);
                    out.close();
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        public static void main(String args[]) {
            // Запуск сервера
            new ServerTCP();
        }
}
```

Создание класса `ClientTCP`

Класс `ClientTCP` позволяет клиенту присоединиться к серверу, используя его IP-адрес (в нашем случае localhost) и получить от него сообщение. Создание класса `ClientTCP` включает в себя следующие основные задачи:

1. Создание сокета для доступа к серверу localhost на порт 1500.
2. Получение входного потока сокета.
3. Чтение объекта-сообщения из потока и отображение полученных данных на экране.

1) Создайте новый Java-класс `ClientTCP` в пакете `ru.tpu.javaEELabs.lab2`.

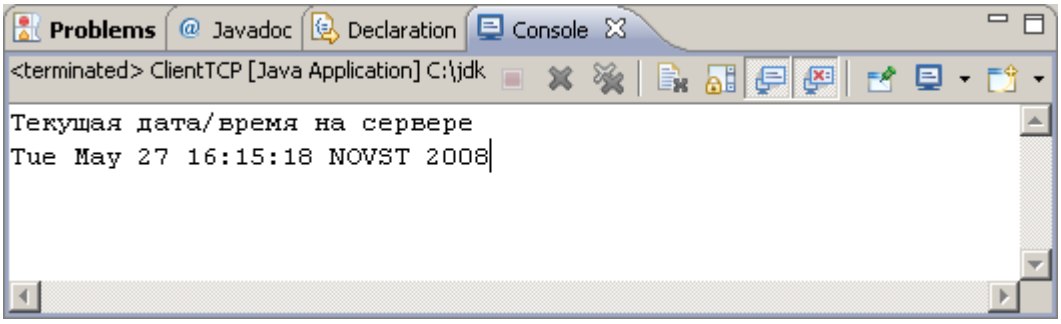
Код класса `Client` приведен ниже:

```
package ru.tpu.javaEELabs.lab2;

import java.io.ObjectInputStream;
import java.net.Socket;

public class ClientTCP {

	public static void main(String args[]) {
		try {
			// Создается объект Socket
			// для соединения с сервером
			Socket clientSocket = new Socket("localhost", 1500);

			// Получаем ссылку на поток, связанный с сокетом
			ObjectInputStream in =
			  new ObjectInputStream(clientSocket.getInputStream());

			// Извлекаем объект из входного потока
			DateMessage dateMessage =
				(DateMessage) in.readObject();

			// Выводим полученные данные в консоль
			System.out.println(dateMessage.getMessage());
			System.out.println(dateMessage.getDate());
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
}
```

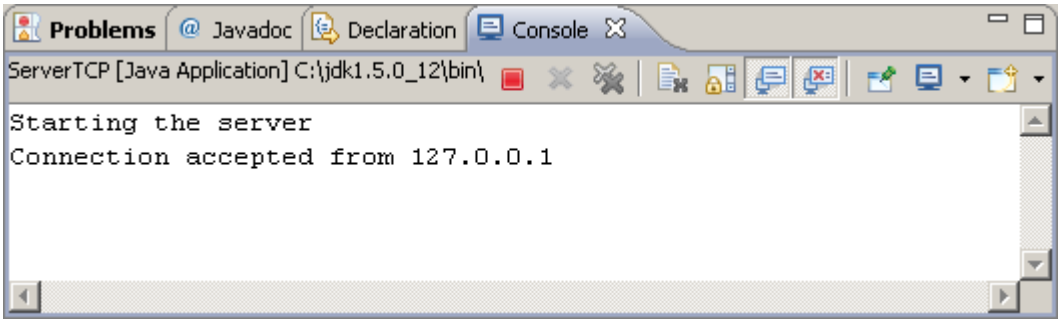
Запуск и тестирование

Щелкните правой кнопкой мыши на класс ServerTCP в окне Package Explorer и выберите команду Run As/Java Application. В консоли отображается сообщение `Starting the server`.

Проделайте то же самое с классом ClientTCP. При запуске клиент пытается соединиться с сервером и обрабатывает полученное сообщение. В результате в консоли клиента выводится следующее:



Выберите консоль сервера и просмотрите сообщение о приеме соединения от клиента:



Варианты заданий

1. Необходимо разработать клиент/серверное приложение, в котором сервер каждые 10 секунд распространяет некоторое текстовое сообщение, например, о погоде, всем *промежуточным* клиентам, зарегистрированным в группе 233.0.0.1, порт 1502 с помощью UDP. Текст сообщения хранится в текстовом файле на сервере. Промежуточный клиент фильтрует полученные сообщения и в случае изменения содержимого отображает его в консоли. Для *конечного* клиента промежуточный клиент выступает сервером. Конечный клиент присоединяется к промежуточному и получает тексты последних пяти отфильтрованных сообщений (с помощью протокола TCP/IP). Необходимо снабдить приложение конечного клиента графическим интерфейсом.

2. Разработать приложение для широковещательного общения пользователей (чат). Клиент отсылает сообщение серверу (с помощью протокола TCP/IP). Сервер накапливает порции сообщений и каждые 10 секунд распространяет очередную порцию сообщений всем клиентам, зарегистрированным в группе 233.0.0.1, порт 1502 с помощью UDP. Если за указанный период не поступило ни одного нового сообщения, то рассылка не производится. Клиент принимает сообщения и отображает их на экране. Клиентское приложение должно иметь удобный графический интерфейс для ввода новых и просмотра полученных сообщений.

Лабораторная работа 3. Создание приложения RMI

Постановка задачи

Необходимо разработать клиент/серверное приложение для удаленной регистрации участников научной конференции. Сервер организаторов конференции содержит базу данных (БД) и RMI-сервис для приема и записи регистрационных сведений в БД. Участникам конференции предоставляется приложение с графическим интерфейсом для ввода и отсылки данных на сервер с помощью вызова удаленного метода RMI.

Для решения поставленной задачи необходимо выполнить следующие шаги:

1. Создать новый проект.
2. Создать в БД таблицу `registration_info` для хранения данных регистрации участников конференции.
3. Создать сериализуемый Java-класс `RegistrationInfo` для представления и передачи данных регистрации.
4. Реализовать интерфейс `ConfServer` и класс реализации `ConfServerImpl` удаленных методов сервера.
5. Создать клиентское приложение – разработать графический интерфейс (Swing) и обеспечить вызов удаленного метода сервера.
6. Выполнить приложение.

Подготовительный этап

Для реализации проекта необходимо установить и настроить среду разработки Ecplise, Apache Derby и Derby Plugins (см. п. «Установка и настройка программного обеспечения»).

Создание нового проекта

1) Выберите пункт меню File/New/Project, в окне выбора типа проекта укажите other/Java Project и нажмите Next
2) Укажите имя проекта Lab3 и нажмите Finish.

Создание таблицы `registration_info`

1) Подключитесь к БД Derby и запустите сервер БД (см. п. «Установка и настройка программного обеспечения», пп. 8 «Запуск и остановка Apache Derby»).

2) Для хранения SQL-скриптов создадим новый файл registration_info.sql. В окне Package Explorer щелкните правой кнопкой мыши на значок проекта и выберите New/File, укажите имя файла registration_info.sql и нажмите Finish.
3) Скопируйте в файл следующие команды:

```
-- подключение
connect
'jdbc:derby://localhost:1527/myDB;create=true;user=me;password=mine';

-- создание таблицы
create table registration_info(first_name varchar(20), last_name
varchar(20), organization varchar(100), report_theme varchar(300), email
varchar(20));

-- отключение и выход
disconnect;
exit;
```

4) Сохраните файл нажатием на Ctrl-S
5) Щелкните правой кнопкой мыши на файл registration_info.sql в окне Package Explorer и выберите Apache Derby/Run SQL Script using 'ij'
6) В случае успешного выполнения скрипта в консоли выводится следующее:

```
ij version 10.3
ij> -- подключение
connect
'jdbc:derby://localhost:1527/myDB;create=true;user=me;password=mine';
ij> -- создание таблицы
create table registration_info(first_name varchar(20), last_name
varchar(20), organization varchar(100), report_theme varchar(300), email
varchar(20));
0 rows inserted/updated/deleted
```

Создание класса `RegistrationInfo`

Перед тем как начать создание интерфейса и класса реализации, создадим обычный сериализуемый Java-класс `RegistrationInfo`, который будет использоваться для представления и передачи данных об участнике конференции.

1) Создайте новый Java-класс, нажав правой кнопкой мыши на каталог src и выбрав пункт меню New/Class. Назовите класс RegistrationInfo и разместите его в пакете `ru.tpu.javaEElabs.lab3`.

2) В классе Employee создайте пять полей, соответствующих столбцам таблицы registration_info, добавьте конструкторы и набор get/set методов. Полный код класса `RegistrationInfo` приведен ниже:

```
package ru.tpu.javaEElabs.lab3;

import java.io.Serializable;

public class RegistrationInfo implements Serializable {

	private String firstName;
	private String lastName;
	private String organization;
	private String reportTheme;
	private String email;

	public RegistrationInfo() {}

	public RegistrationInfo(String firstName, String lastName,
		String organization, String reportTheme, String email) {
		super();
		this.firstName = firstName;
		this.lastName = lastName;
		this.organization = organization;
		this.reportTheme = reportTheme;
		this.email = email;
	}

	public String getFirstName() {
		return firstName;
	}
	public void setFirstName(String firstName) {
		this.firstName = firstName;
	}
	public String getLastName() {
		return lastName;
	}
	public void setLastName(String lastName) {
		this.lastName = lastName;
	}
	public String getOrganization() {
		return organization;
	}
	public void setOrganization(String organization) {
		this.organization = organization;
	}
	public String getReportTheme() {
		return reportTheme;
	}
	public void setReportTheme(String reportTheme) {
		this.reportTheme = reportTheme;
	}
	public String getEmail() {
		return email;
	}
```

```
    public void setEmail(String email) {
          this.email = email;
    }
}
```

Создание интерфейса `ConfServer` и класса реализации `ConfServerImpl`

Интерфейс `ConfServer` объявляет удаленные методы, которые могут быть вызваны клиентом RMI. В нашем случае интерфейс будет содержать один метод `registerConfParticipant`, принимающий характеристики участника конференции, и сохраняющий их в БД.

1) Для создания нового интерфейса щелкните правой кнопкой мыши на пакет `ru.tpu.javaEElabs.lab3` в окне Package Explorer и выберите New/Interface/
2) В появившемся окне в качестве имени класса (Name) задайте ConfServer и убедитесь что в качестве имени пакета (Package) указано `ru.tpu.javaEELabs.lab3`. Нажмите Finish.

   Код интерфейса `ConfServer` приведен ниже:

```
package ru.tpu.javaEElabs.lab3;

import java.rmi.*;

public interface ConfServer extends Remote {
  int registerConfParticipant(RegistrationInfo registrationInfo)
                          throws RemoteException;
}
```

Создание класса реализации `ConfServerImpl`

Класс `ConfServerImpl` содержит реализацию удаленного метода регистрации участников концференции. Объект класса `ConfServerImpl` представляет собой удаленный сервис и должен быть зарегистрирован под определенным именем в регистре RMI, входящем в состав Java Virtual Machine и запускаемый командой rmiregitry. Регистр RMI, обеспечивает хранение, поиск и выполнение методов объекта удаленными клиентами.

Перед регистрацией объекта в регистре RMI необходимо во-первых, указать путь к откомпилированному классу реализации `ConfServerImpl` (каталог bin в каталоге проекта). Во-вторых, необходимо настроить параметры менеджера безопасности (security manager), таким образом, чтобы виртуальная машина сервера могла запускать код объектов, пришедших (например, по сети) в качестве аргументов вызова удаленных методов. Эти настройки могут быть указаны с помощью файлов конфигурации, параметров запуска приложения, либо в самом

коде метода. В приведенном ниже примере используется последний способ.

Создание класса `ConfServerImpl` включает в себя следующие основные задачи:

1. Реализацию интерфейса `ConfServerImpl`.
2. Создание конструктора.
3. Обеспечение реализации удаленного метода `registerConfParticipant`.
4. Создание метода `main()`, выполняемого при запуске сервера, где выполняется:

   - указание регистру RMI пути к файлу класса реализации сервера путем установки значения системного свойства `java.rmi.server.codebase`;

   - создание и настройка менеджера безопасности `RMISecurityManager`;

   - создание и регистрация в регистре RMI удаленного объекта `ConfServer`.

1) Для создания класса щелкните правой кнопкой мыши на пакет `ru.tpu.javaEELabs.lab3` в каталоге src окна Package Explorer и выберите New/Class.
2) В появившемся окне в качестве имени класса (Name) задайте ConfServerImpl. Нажмите Finish.

   Код класса `ConfServerImpl` приведен ниже:

```
package ru.tpu.javaEElabs.lab3;

import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
import java.security.Permission;
import java.sql.*;

public class ConfServerImpl extends UnicastRemoteObject
      implements ConfServer {
      /* Определяется конструктор по умолчанию */
      public ConfServerImpl() throws RemoteException {
            super();
      }

      /* Определение удаленного метода */
      public int registerConfParticipant(RegistrationInfo
                        registrationInfo) throws RemoteException {
            try {
                  // Регистрация драйвера БД Derby
      Class.forName("org.apache.derby.jdbc.ClientDriver").newInstance();

                  // Получение соединения с БД
```

```
                Connection con = DriverManager.getConnection(
"jdbc:derby://localhost:1527/myDB;create=true;user=me;password=mine");

                // Запись полученных данных в БД
                PreparedStatement st = con.prepareStatement(
                            "insert into registration_info " +
                            "(first_name, last_name, organization, " +
                            "report_theme, email) " +
                            "values (?, ?, ?, ?, ?)");
                st.setString(1, registrationInfo.getFirstName());
                st.setString(2, registrationInfo.getLastName());
                st.setString(3, registrationInfo.getOrganization());
                st.setString(4, registrationInfo.getReportTheme());
                st.setString(5, registrationInfo.getEmail());
                st.executeUpdate();
                st.close();
                // Получение количества зарегистрированных участников
                Statement st1 = con.createStatement();
                int count = 0;
                ResultSet rs = st1.executeQuery(
                            "Select count(*) from registration_info");
                if (rs.next()) {
                      count = rs.getInt(1);
                }
                st1.close();

                return count;
          } catch (Exception e) {
                e.printStackTrace();
                throw new RemoteException(e.getMessage(), e);
          }
    }

    /* Метода main() */
    public static void main(String args[]) {
          try {
          // Указание расположения классов RMI
          System.setProperty("java.rmi.server.codebase",
          "file:///D:/JavaEE-Workbook/labs-workspace/Lab3_RMI/bin/");

          // Установка менеджера безопасности (если не установлен):
          // Создается новый объект анонимного
    //класса RMISecurityManager
          // и переопределяется метод checkPermission.
          // Метод не содержит кода, следовательно, не определяет
          // никаких ограничений

          if (System.getSecurityManager() == null) {
                System.setSecurityManager(new RMISecurityManager() {

                public void checkConnect(String host, int port,
                            Object context) {}
          public void checkConnect(String host, int port) {}
                public void checkPermission(Permission perm) {}
            });
          }

          // Создание экземпляра класса ConfServerImpl
```

```
            ConfServerImpl instance = new ConfServerImpl();
            // Регистрация объекта RMI под именем ConfServer
            Naming.rebind("ConfServer", instance);

            System.out.println("Сервис зарегистрирован");
            } catch (Exception e) {
                  e.printStackTrace();
            }
      }
}
```

Создание клиента

Класс `ConfClient` обращается к удаленному хосту (в нашем примере localhost) и получает ссылку на удаленный объект из регистра RMI. После этого клиент получает возможность вызова удаленных методов.

1) Щелкните правой кнопкой мыши на пакет `ru.tpu.javaEELabs.lab3` в каталоге src окна Package Explorer и выберите New/Class.
2) В появившемся окне в качестве имени класса (Name) задайте ConfClient. Нажмите Finish.

   Код класса `ConfClient` приведен ниже:

```
package ru.tpu.javaEElabs.lab3;

import javax.swing.*;
import java.rmi.*;
import java.awt.event.*;
import java.awt.*;

public class ConfClient {
      /* Объявляются переменные */
      static JFrame frame;
      static JPanel panel;

      JLabel lbLastName;
      JLabel lbFirstName;
      JLabel lbOrganization;
      JLabel lbReportTheme;
      JLabel lbEmail;

      JTextField txtLastName;
      JTextField txtFirstName;
      JTextField txtOrganization;
      JTextField txtReportTheme;
      JTextField txtEmail;

      JButton submit;

      /* Определяется конструктор по умолчанию */
      public ConfClient() {
            /* Создается JFrame */
            frame = new JFrame("Регистрация участника конференции");
```

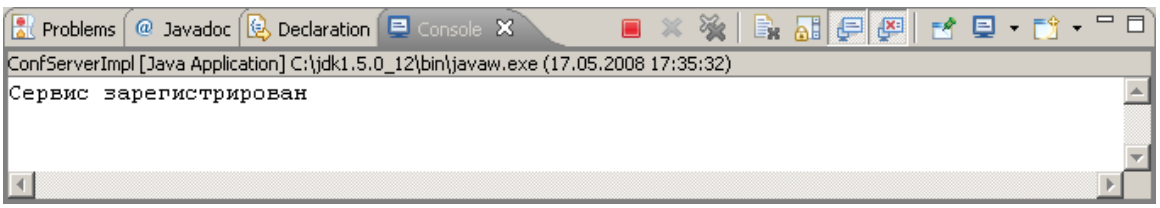
```
      panel = new JPanel();
      /* Набор менеджеров разметки */
      panel.setLayout(new GridLayout(5, 2));
      frame.setBounds(100, 100, 400, 200);
      frame.getContentPane().setLayout(new BorderLayout());
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      /* Define the swing components on the JFrame */
      lbLastName = new JLabel("Фамилия");
      lbFirstName = new JLabel("Имя");
      lbReportTheme = new JLabel("Тема доклада");
      lbOrganization = new JLabel("Организация");
      lbEmail = new JLabel("Емайл");

      txtLastName = new JTextField(15);
      txtFirstName = new JTextField(15);
      txtOrganization = new JTextField(70);
      txtReportTheme = new JTextField(100);
      txtEmail = new JTextField(15);

      submit = new JButton("Отправить");

      /* Добавление в панель компонентов swing */
      panel.add(lbLastName);
      panel.add(txtLastName);
      panel.add(lbFirstName);
      panel.add(txtFirstName);
      panel.add(lbOrganization);
      panel.add(txtOrganization);
      panel.add(lbReportTheme);
      panel.add(txtReportTheme);
      panel.add(lbEmail);
      panel.add(txtEmail);
      frame.getContentPane().add(panel, BorderLayout.CENTER);
      frame.getContentPane().add(submit, BorderLayout.SOUTH);
      frame.setVisible(true);

      submit.addActionListener(new ButtonListener());
}

/* Создание класса ButtonListener */
class ButtonListener implements ActionListener {
      /* Определение метода actionPerformed() */
      public void actionPerformed(ActionEvent evt) {
            try {
                  // Получение удаленного объекта
                  // Если сервер размещен на удаленном компьютере,
                  // то вместо localhost указывается имя
                  // хоста сервера
                  ConfServer server = (ConfServer) Naming.lookup(
                        "rmi://localhost/ConfServer");

                  // Формирование сведений о регистрации для
                  //отправки на сервер
                  RegistrationInfo registrationInfo =
                        new RegistrationInfo(
                                    txtFirstName.getText(),
                                    txtLastName.getText(),
```

```
                                        txtOrganization.getText(),
                                        txtReportTheme.getText(),
                                        txtEmail.getText());
                        // Вызов удаленного метода
                        int count = server.
                              registerConfParticipant(registrationInfo);

                        JOptionPane.showMessageDialog(frame,
                        "Регистрация выполнена успешно" +
                        "\nКоличество зарегистрированных участников - " +
                        count +
                        "\nСпасибо за участие");

                  } catch (Exception e) {
                        JOptionPane.showMessageDialog(frame, "Ошибка");
                        System.out.println(e);
                  }
            }
      }

      // Определение метода main()
      public static void main(String args[]) {
            // Создание объекта класса Client
            new ConfClient();
      }
}
```

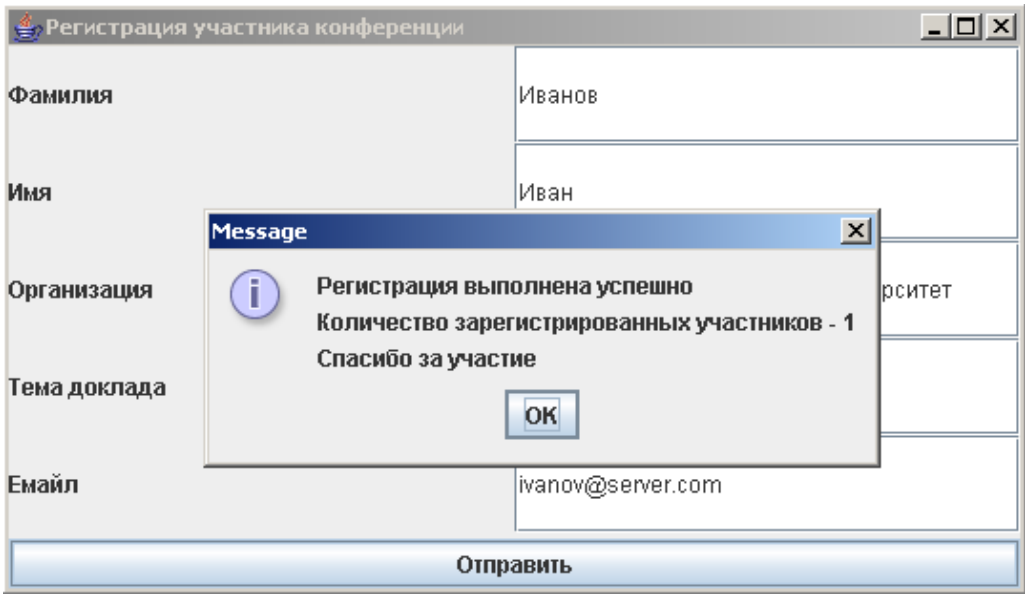
Запуск и тестирование

Каждый из классов `ConfServerImpl` и `ConfClient` содержит метод `main()` и является независимым приложением, которое может быть запущено на отдельном компьютере. В нашем случае роль клиента и сервера будет выполнять один и тот же компьютер.

1) Запустите службу регистра RMI с помощью команды `rmiregistry`. В Windows это действие может быть выполнено с помощью команды Пуск/Выполнить. Служба регистра RMI обеспечивает хранение удаленных объектов и доступ к ним клиентов и должна быть запущена на протяжении всего времени работы приложений с удаленными объектами.
2) Щелкните правой кнопкой мыши на класс ConfServerImpl в окне Package Explorer и выберите команду Run As/Java Application. В результате выполнения в службе RMI регистрируется объект ConfServer. В случае успешной регистрации выводится сообщение:



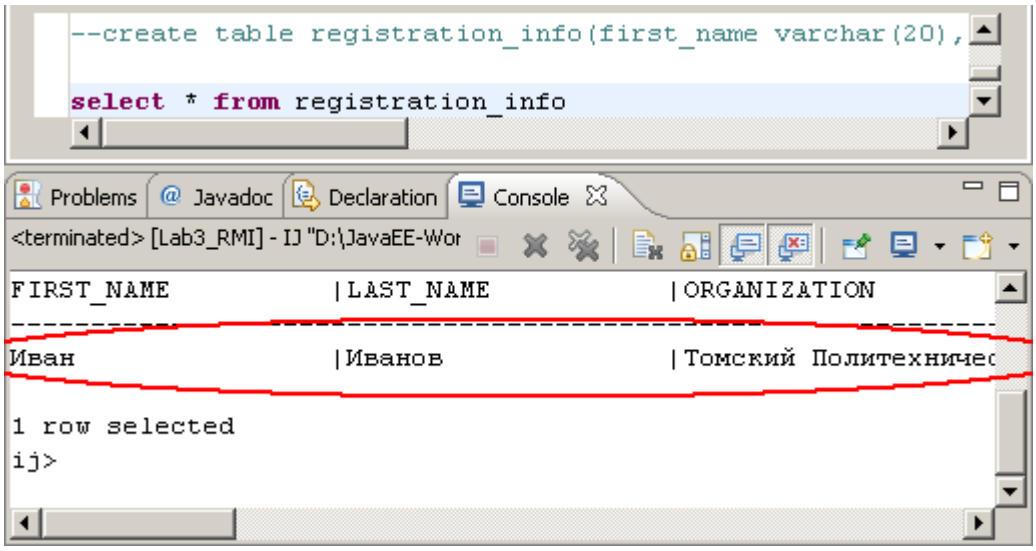
3) Аналогичным образом запустите класс `ConfClient`. В появившемся окне укажите данные регистрации нового участника и нажмите Отправить. Результаты успешного выполнения программы приведены на следующем рисунке:



4) Проверим появилась ли запись о новом участнике в таблице БД `registration_info`. Для этого откройте файл registration_info.sql, закоментируйте строку `create table registration_info` и добавьте следующий запрос:

```
select * from registration_info
```

5) Выполните скрипт и просмотрите результаты Select-запроса:



Варианты заданий

1. На удаленном сервере хранится база данных документов. Необходимо разработать клиент/серверное приложение для обеспечения возможности поиска и загрузки документов. Каждый документ описывается в виде набора следующих атрибутов: название, дата создания, автор, путь к файлу. Пользователь должен иметь возможность просмотра списка документов, и загрузки необходимого файла документа на свой компьютер. Обеспечить графический интерфейс для клиентского приложения. Рекомендация: содержимое файла можно передавать в виде массива байт (byte[]).

2. На удаленном сервере хранится база данных изображений. Необходимо разработать клиент/серверное приложение для обеспечения возможности их просмотра. Каждое изображение представляет собой файл на сервере и описывается с помощью следующих атрибутов: краткое описание, дата создания, автор, путь к файлу. Пользователь должен иметь возможность просмотра списка изображений, и просмотра выбранного изображения на своем компьютере. Обеспечить графический интерфейс для клиентского приложения. Рекомендация: содержимое файла можно передавать в виде массива байт (byte[]).

Лабораторная работа 4. Создание веб-приложения с использованием технологий JSP и Servlet

Часть 1. Разработка сервлета

Постановка задачи

Необходимо разработать веб-приложение, использующее сервлет для поиска информации о сотрудниках организации. Данные о сотрудниках хранятся в таблице Employee. Для осуществления поиска пользователь указывает фамилию сотрудника и просматривает информацию о найденных сотрудниках (возможно существование нескольких сотрудников с одинаковыми фамилиями).

Для решения поставленной задачи необходимо выполнить следующие шаги:

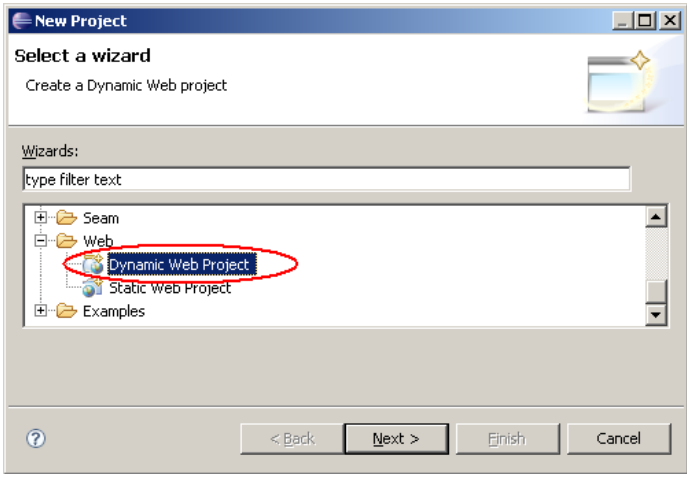
1. Создать новый проект
2. Создать таблицу `employee` и заполнить ее данными
3. Разработать сервлет, который выбирает из БД записи, соответствующие запросу пользователя и отображает результат.
4. Упаковать приложение и развернуть на сервере.
5. Протестировать работу приложения в браузере

Подготовительный этап

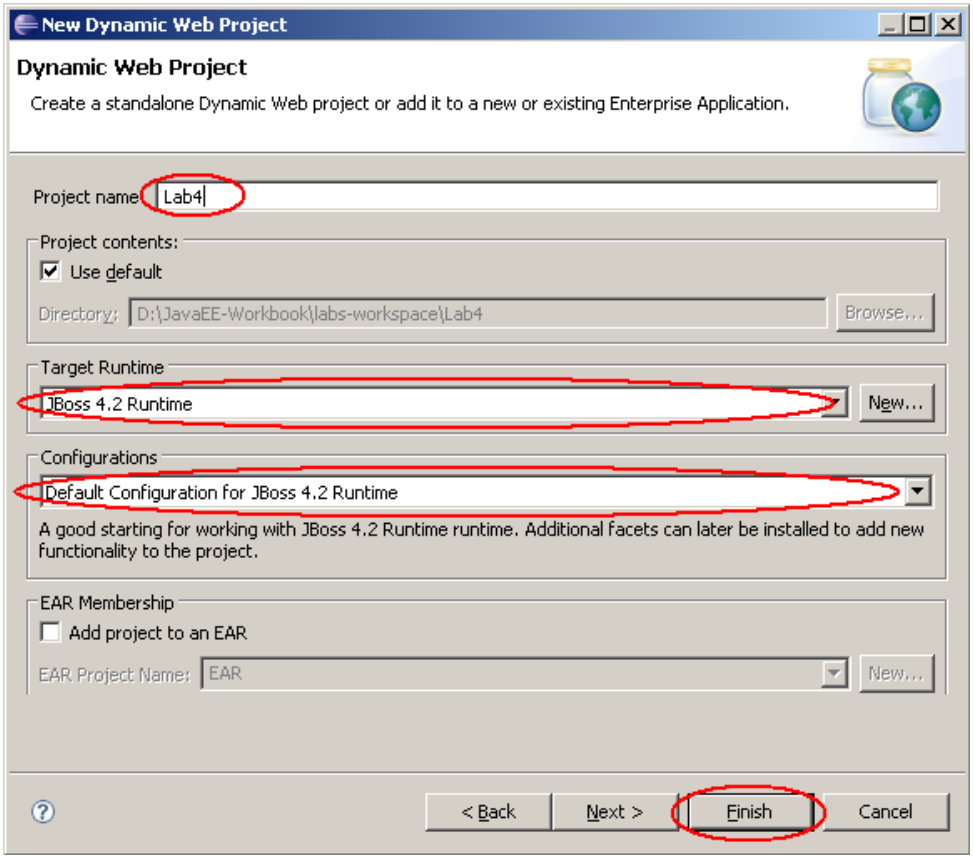
Для реализации проекта необходимо установить и настроить Ecplise, JBoss AS, JBoss Tools, Apache Derby и Derby Plugins (см. п. «Установка и настройка программного обеспечения»).

Создание нового проекта

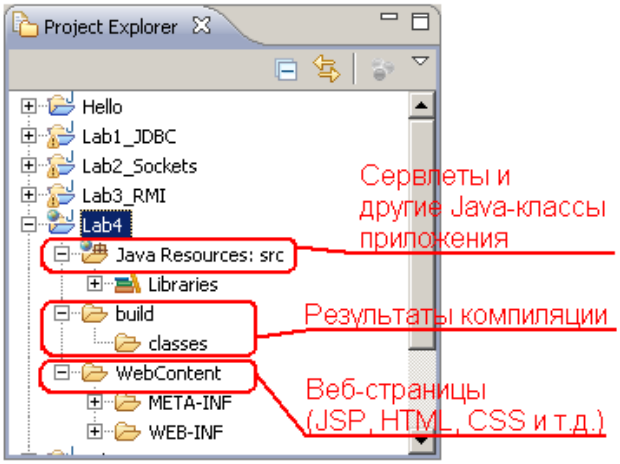
1) Выберите пункт меню File/New/Project, в окне выбора типа проекта укажите Web/Dynamic Web Project и нажмите Next.



2) Укажите имя проекта Lab4. Остальные настройки, связанные с сервером приложений, на котором будет разворачиваться веб-приложение, должны отобразиться автоматически. Нажмите Finish.

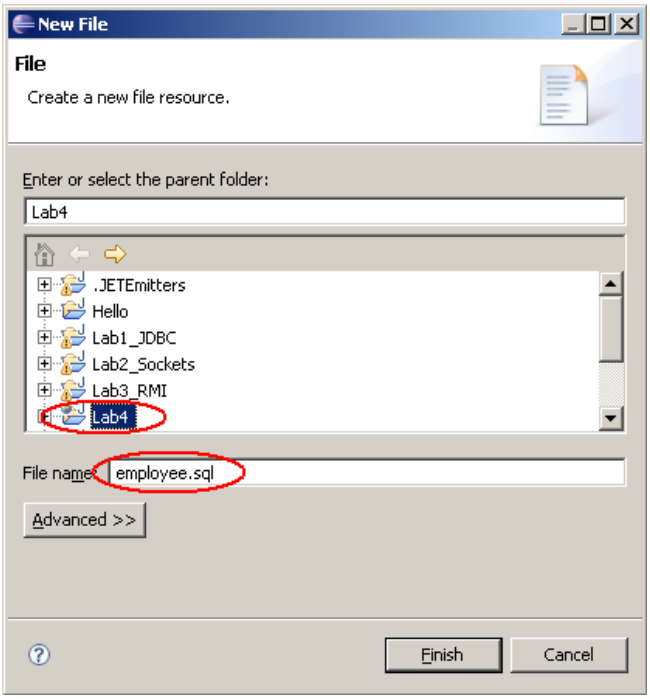


3) В результате будет создан проект следующей структуры:



Создание таблицы `employee` и вставка тестовых данных

1) Подключитесь к БД Derby и запустите сервер БД (см. п. «Установка и настройка программного обеспечения», пп. 8 «Запуск и остановка Apache Derby»).
2) Для хранения SQL-скриптов создадим новый файл employee.sql. В окне Project Explorer щелкните правой кнопкой мыши на значок проекта Lab4 и выберите New/File, укажите имя файла employee.sql и нажмите Finish.



3) Созданный файл автоматически открывается для редактирования. Скопируйте в файл следующие команды:

```
-- подключение
connect
'jdbc:derby://localhost:1527/myDB;create=true;user=me;password=mine';

-- раскомментируйте следующую строку, если требуется пересоздать таблицу
-- drop table employee;

-- создание таблицы
create table employee(id integer, first_name varchar(20), last_name var-
char(20), designation varchar(20), phone varchar(20));

--вставка тестовых данных
insert into employee values (1, 'Ivan', 'Ivanov', 'Manager', '11-22-
33');
insert into employee values (2, 'Nikolay', 'Ivanov', 'Programmer', '33-
44-55');
insert into employee values (3, 'Sergey', 'Petrov', 'System administra-
tor', '12-34-56');
insert into employee values (4, 'Alexey', 'Petrov', 'Manager', '56-78-
90');
insert into employee values (5, 'Vitaliy', 'Kuznetsov', 'Technician',
'55-66-77');

-- выбрать все из таблицы для проверки
select * from employee;

-- отключение и выход
disconnect;
exit;
```

4) Сохраните файл нажатием на Ctrl-S.
5) Щелкните правой кнопкой мыши на файл employee.sql в окне Project Explorer и выберите Apache Derby/Run SQL Script using ‘ij’.
6) В случае успешного выполнения скрипта в консоли выводится следующая информация:

```
ij version 10.3
ij> -- подключение
connect
'jdbc:derby://localhost:1527/myDB;create=true;user=me;password=mine';
ij> -- раскоментируйте следующую строку, если требуется пересоз-
дать таблицу
 drop table employee;
0 rows inserted/updated/deleted
ij> -- создание таблицы
create table employee(id integer, first_name varchar(20),
last_name varchar(20), designation varchar(20), phone varchar(20));
0 rows inserted/updated/deleted
ij> --вставка тестовых данных
```

```
      insert into employee values (1, 'Ivan', 'Ivanov', 'Manager', '11-
22-33');
      1 row inserted/updated/deleted
      ij> insert into employee values (2, 'Nikolay', 'Ivanov', 'Program-
mer', '33-44-55');
      1 row inserted/updated/deleted
      ij> insert into employee values (3, 'Sergey', 'Petrov', 'System
administrator', '12-34-56');
      1 row inserted/updated/deleted
      ij> insert into employee values (4, 'Alexey', 'Petrov', 'Manager',
'56-78-90');
      1 row inserted/updated/deleted
      ij> insert into employee values (5, 'Vitaliy', 'Kuznetsov', 'Tech-
nician', '55-66-77');
      1 row inserted/updated/deleted
      ij> -- выбрать все из таблицы для проверки
      select * from employee;
      ID          |FIRST_NAME          |LAST_NAME           |DESIGNATION
|PHONE
      ------------------------------------------------------------------
------------------------------
      1           |Ivan                  |Ivanov                |Manager
|11-22-33
      2           |Nikolay               |Ivanov                |Programmer
|33-44-55
      3           |Sergey                |Petrov                |System ad-
ministrator|12-34-56
      4           |Alexey                |Petrov                |Manager
|56-78-90
      5           |Vitaliy               |Kuznetsov             |Technician
|55-66-77

      5 rows selected
      ij> -- отключение и выход
      disconnect;
      ij> exit;
```

Реализация сервлета

Перед тем как начать программировать сервлет, создадим обычный Java-класс Employee, который будет использоваться для представления и передачи данных о сотруднике.

1) Создайте новый Java-класс, нажав правой кнопкой мыши на проект Lab4 и выбрав пункт меню New/Class. Назовите класс Employee и разместите его в пакете `ru.tpu.javaEElabs.lab4`.
2) В классе `Employee` создайте пять полей, соответствующих столбцам таблицы `employee`, добавьте конструкторы и набор get/set методов. Полный код класса `Employee` приведен ниже:

```
package ru.tpu.javaEElabs.lab4;

import java.io.Serializable;

public class Employee implements Serializable {
```

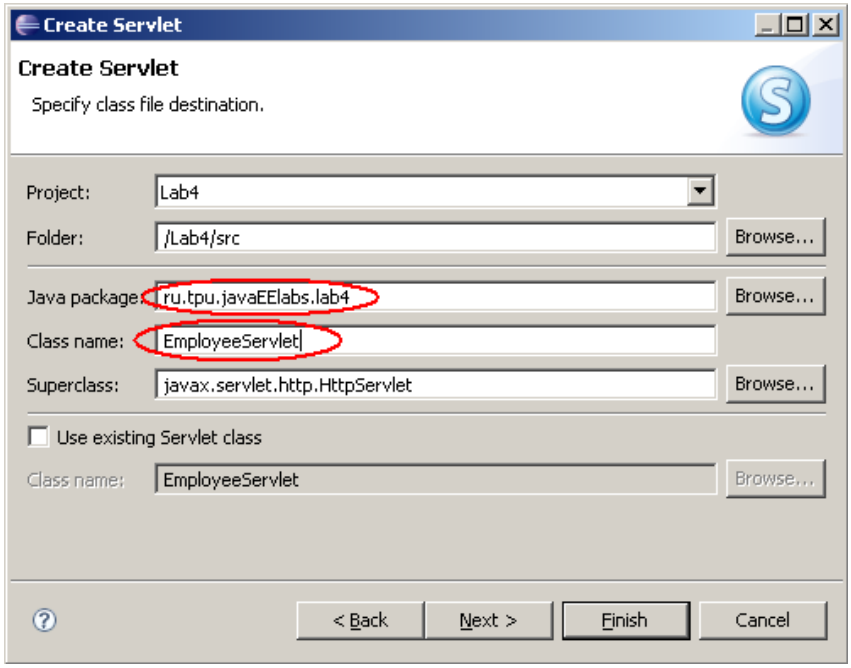
```
private Long id;
private String firstName;
private String lastName;
private String designation;
private String phone;

public Employee() {}

public Employee(Long id, String firstName, String lastName,
            String designation, String phone) {
      super();
      this.id = id;
      this.firstName = firstName;
      this.lastName = lastName;
      this.designation = designation;
      this.phone = phone;
}

public Long getId() {
      return id;
}

public void setId(Long id) {
      this.id = id;
}

public String getFirstName() {
      return firstName;
}

public void setFirstName(String firstName) {
      this.firstName = firstName;
}

public String getLastName() {
      return lastName;
}

public void setLastName(String lastName) {
      this.lastName = lastName;
}

public String getDesignation() {
      return designation;
}

public void setDesignation(String designation) {
      this.designation = designation;
}

public String getPhone() {
      return phone;
}

public void setPhone(String phone) {
      this.phone = phone;
}
```

```
}
```

Прежде чем приступить к разработке сервлета, отключите пункт Project/Build Automatically в главном меню Eclipse. Это позволит избежать автоматической развертки приложения на сервер при сохранении исходных файлов.

Теперь можно переходить к созданию сервлета.

3) Для создания нового сервлета щелкните правой кнопкой мыши на проект Lab4, выберите пункт меню New/Other, укажите Web/Servlet и нажмите Next.
4) Укажите пакет (Java package), в котором будет размещен сервлет, например, `ru.tpu.javaEElabs.lab4`.
5) Укажите имя класса сервлета (Class name) EmployeeServlet и нажмите Finish.



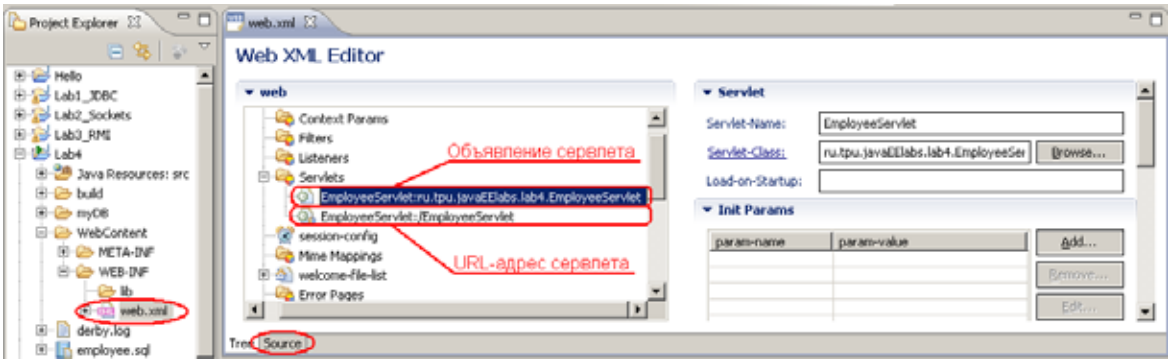
6) Программный код сервлета приведен ниже. Основная бизнес логика сосредоточена в методе `doGet()`, который выполняется на сервере после того, как пользователь запустил сервлет на выполнение. Скопируйте код и сохраните сервлет, нажав Ctrl-S.

```
package ru.tpu.javaEElabs.lab4;

import java.io.IOException;
import java.io.PrintWriter;
```

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.ArrayList;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class EmployeeServlet extends javax.servlet.http.HttpServlet implements
                javax.servlet.Servlet {
       static final long serialVersionUID = 1L;

       public EmployeeServlet() {
              super();
       }

       protected void doGet(HttpServletRequest request,
                     HttpServletResponse response)
                     throws ServletException, IOException {
          try {
              response.setContentType("text/html;charset=UTF-8");
              // Получение из http-запроса значения параметра lasname
              String lastname = request.getParameter("lastname");

              // Коллекция для хранения найденных сотрудников
              ArrayList<Employee> employees = new ArrayList<Employee>();

              // Загрузка драйвера БД Derby
              Class.forName("org.apache.derby.jdbc.ClientDriver").newInstance();

              // Получение соединения с БД
       Connection          con          =          DriverManager.getConnection(
       "jdbc:derby://localhost:1527/myDB;create=true;user=me;password=mine");

              // Выполнение SQL-запроса
              ResultSet rs = con.createStatement().executeQuery(
              "Select id, first_name, last_name, designation, phone "
               + "From employee " + "Where last_name like '"
               + lastname + "'");
              // Перечисление результатов запроса
              while (rs.next()) {
                     // По каждой записи выборки формируется
                     // объект класса Employee.
                     // Значения свойств заполяются из полей записи
                     Employee emp = new Employee(
                     rs.getLong(1),
                     rs.getString(2),
                     rs.getString(3),
                     rs.getString(4),
                     rs.getString(5));
                     // Добавление созданного объекта в коллекцию
                     employees.add(emp);
              }
              // Закрываем выборку и соединение с БД
              rs.close();
              con.close();

              // Выводим информацию о найденных сотрудниках
              PrintWriter out = response.getWriter();
```

```
            out.println("Найденные сотрудники<br>");
            for (Employee emp: employees) {
                  out.print(emp.getFirstName() + " " +
                              emp.getLastName() + " " +
                              emp.getDesignation() + " " +
                              emp.getPhone() + "<br>");
            }
         } catch (Exception ex) {
            ex.printStackTrace();
            throw new ServletException(ex);
         }
      }

      protected void doPost(HttpServletRequest request,
                  HttpServletResponse response)
                  throws ServletException, IOException {
      }
}
```

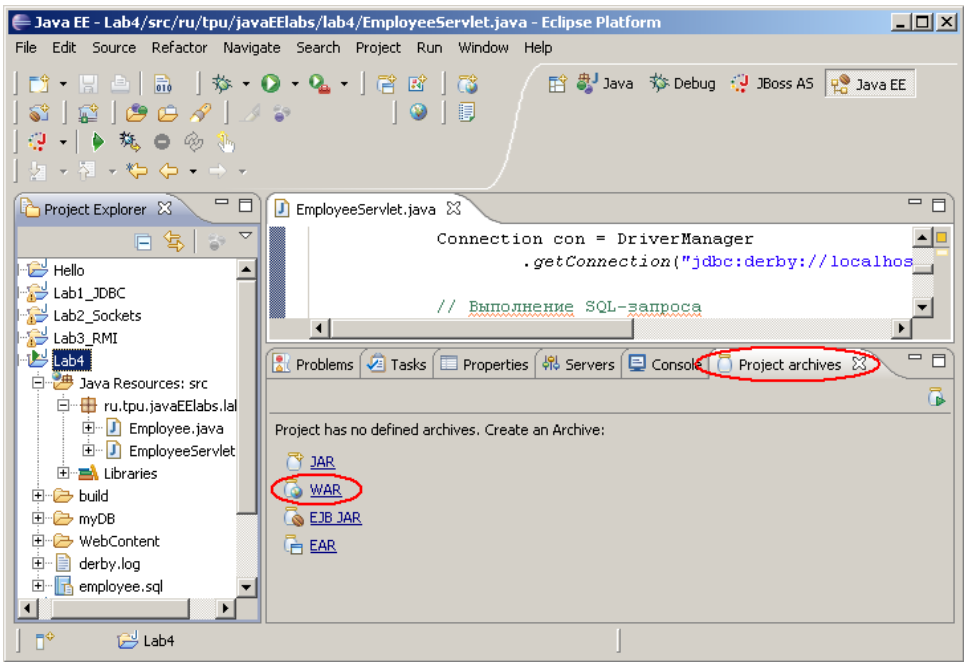
7) При создании сервлета с помощью мастера (как было описано выше), описание сервлета в конфигурационном файле web.xml генерируется автоматически. Для просмотра/редактирования этого файла в окне Project Explorer дважды щелкните на элемент Lab4/Web Content/WEB-INF/web.xml.
8) В редакторе Web XML Editor раскройте категорию Servlets и просмотрите параметры сервлета. Первый элемент объявляет сервлет веб-контейнеру, второй – описывает URL-ссылку на сервлет.
9) Содержимое файла web.xml также можно просмотреть в виде исходного XML-документа. Для этого следует выбрать закладку Source в нижней части окна редактора:



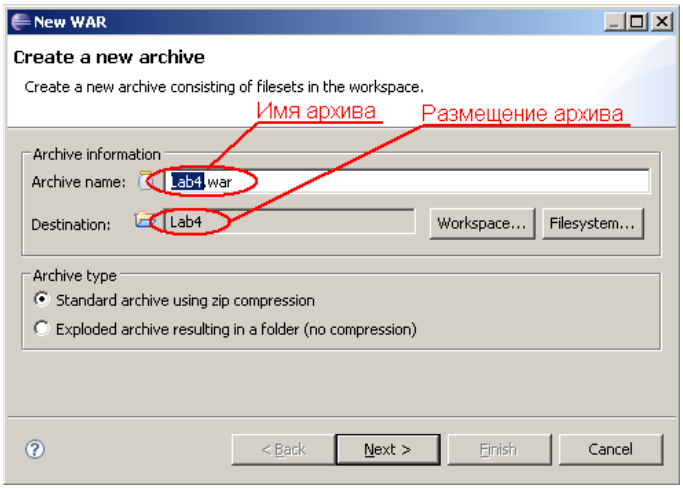
Упаковка приложения

Для упаковки веб-приложения в WAR-файл, пригодный для размещения на сервере приложений, необходимо создать конфигурацию архива, в которой перечисляются подкаталоги и файлы, которые будут включены в архив.

1) Перейдите на закладку Project Archives. В случае, если закладка отсутствует, подключите ее, выбрав пункт меню Window/Show View/Other и выбрав JBoss Tools/Project Archives.
2) В закладке Project Archives выберите WAR:



3) В результате отображается мастер создания нового архива. В первом окне мастера указывается имя архива, его размещение и тип. Оставьте параметры по умолчанию и нажмите Next.



4) В следующем окне настраивается содержимое архива – указывается какие подкаталоги должен содержать архив и какие

файлы должны быть в них помещены. По умолчанию, в веб-архиве будут созданы три подкаталога: WEB-INF и вложенные в него lib и classes:

- в WEB-INF помещаются веб-страницы и прочие веб-ресурсы проекта (html, jsp, css, файлы изображений и т. д.). В нашем проекте эти файлы содержатся в Lab4/WebContent.

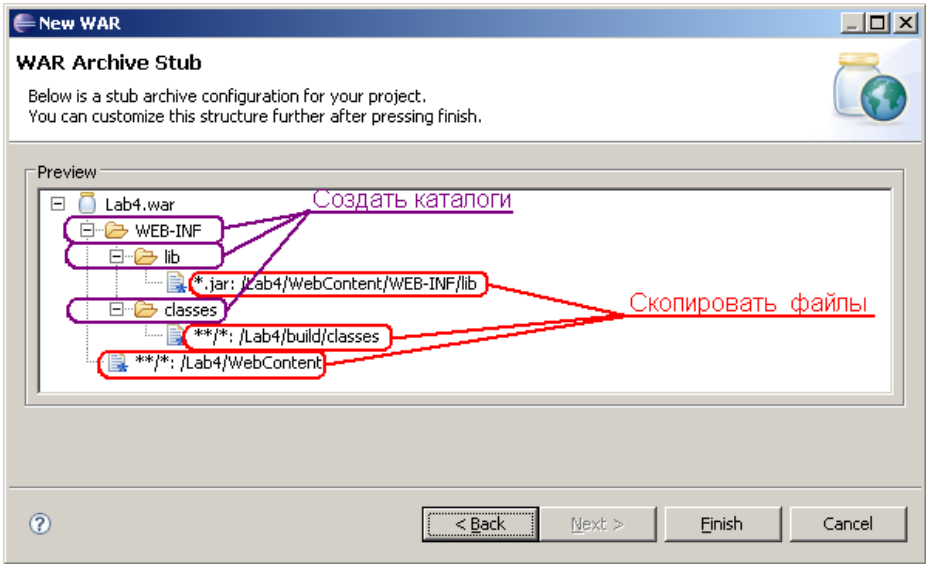
- в lib помещаются дополнительные библиотеки проекта. В нашем проекте они отсутствуют.

- в classes помещаются откомпилированные java-классы проекта, в том числе сервлеты. Каталог Lab4/build/classes, указанный в качестве источника файлов, содержит результаты компиляции исходных файлов проекта, содержащихся в src.

При указании списка файлов, помещаемых в тот или иной подкаталог архива могут использоваться маски. Например:

*.jar – скопировать все файлы с расширением jar

**/* – скопировать все файлы указанной директории и поддиректорий



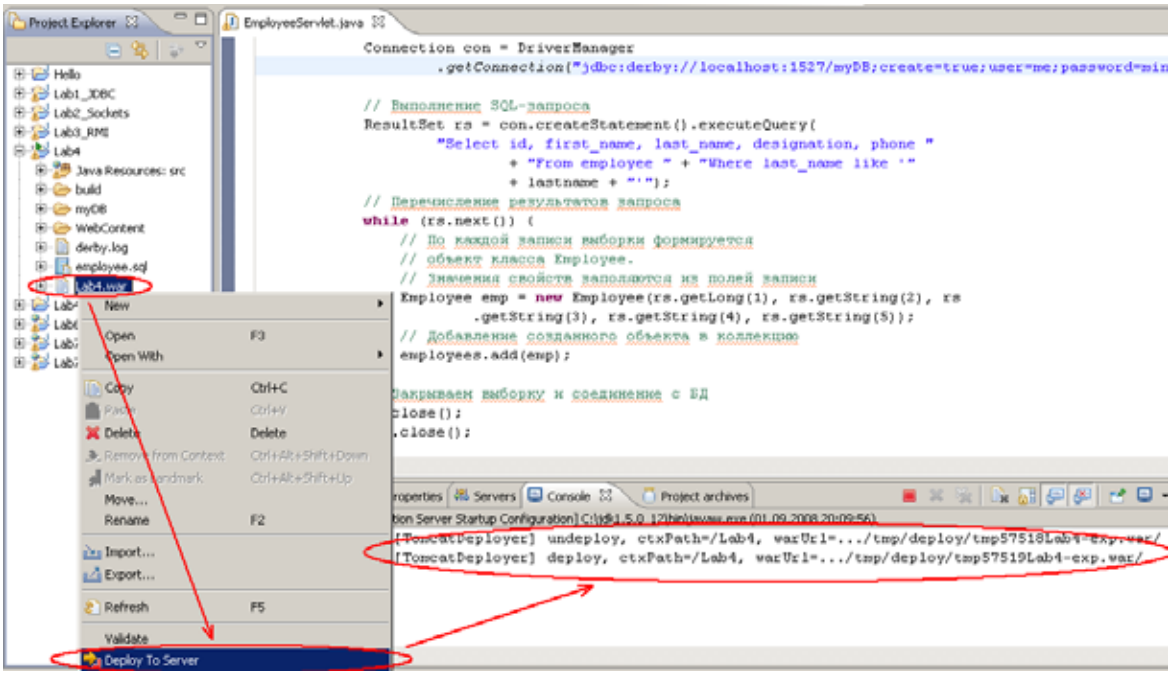
Оставьте настройки по умолчанию и нажмите Finish. В результате в окне Project Archives отображается вновь созданная конфигурация упаковки приложения.

Сборка и развертывание приложения

В процессе сборки выполняется компиляция исходных файлов и упаковка приложения. Eclipse в случае успешной сборки проекта автоматически выполняет и развертывание приложения.

1) Запустите сервер приложений JBoss AS с помощью кнопки панели инструментов Start JBoss 4.2 Server. Обратите внимание, что кнопки запуска/остановки сервера приложений доступны если активной является перспектива JBoss AS:

2) Щелкните правой кнопкой мыши на значок проекта Lab4 в окне Project Explorer и выберите Build Project. Если компиляция прошла успешно, выполняется создание/обновление архива Lab4.war. Файл архива отображается в конце дерева проекта в окне Project Explorer.
3) Для развертывания приложения на сервере, щелкните правой кнопкой мыши на архив Lab4.war в окне Project Explorer и выберите команду Deploy To Server. В случае если в консоли отображается сообщение `deploy...` и не появилось информации об ошибках, значит развертывание приложения прошло успешно. Более подробную информацию о процессе и результате развертывания приложения можно просмотреть в лог-файле сервера: C:\jboss-4.2.1.GA\server\default\log\ server.log.



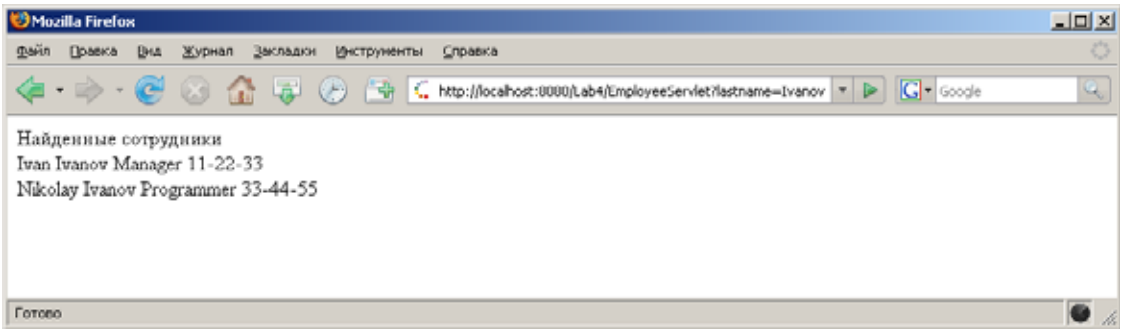
Примечание: в консоли Eclipse отражаются сокращенные сведения о действиях сервера, полный перечень которых содержится в лог-файле server.log. Для того чтобы консоль Eclipse полностью воспроизводила содержимое лог-файлов, откройте на редактирование файл C:\jboss-4.2.1.GA\server\default\conf\jboss-log4j.xml, найдите раздел `<appender name="CONSOLE" …>` и исправьте значение (`value`) параметра `Threshold` на `DEBUG`, как показано ниже:

```
<appender name="CONSOLE" class="org.apache.log4j.ConsoleAppender">
 …
    <param name="Threshold" value="DEBUG"/>
  …
 </appender>
```

Сохраните файл и перезапустите JBoss AS.

Тестирование приложения

Запустите браузер и перейдите по ссылке `http://localhost:8080/Lab4/EmployeeServlet?lastname=Ivanov`. Обратите внимание, что в запросе выполняется обращение к сервлету и передается параметр `lastname` со значением `Ivanov`. Просмотрите результаты выполнения запроса.



Часть 2. Разработка JSP страницы

Постановка задачи

Необходимо доработать веб-приложение, организовав веб-интерфейс приложения, в котором пользователь указывает фамилию сотрудника, выполняет поиск и просматривает информацию о найденных сотрудниках. Интерфейс требуется реализовать в виде jsp-страницы.

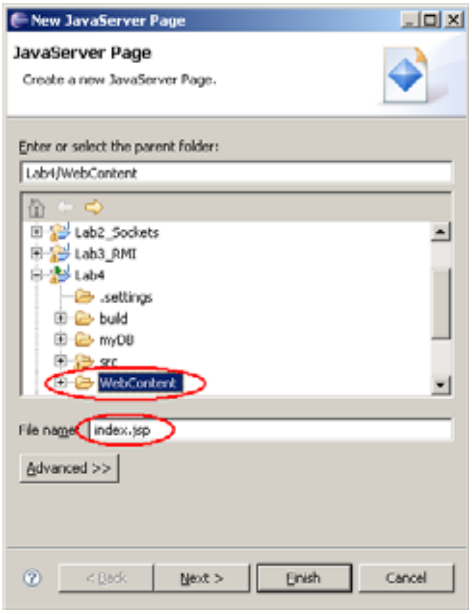
Для решения поставленной задачи необходимо выполнить следующие шаги:

1. Разработать JSP страницу для ввода исходных данных и отсылки сообщения сервлету.
2. Доработать код сервлета для отправки результатов поиска jsp-странице.
3. Упаковать приложение и развернуть на сервере.
4. Протестировать работу приложения в браузере.

Разработка JSP-страницы

Наш проект будет содержать всего одну страницу index.jsp, которая будет использоваться и для ввода фамилии и для отображения результатов поиска.

1) Для создания новой JSP-страницы, нажмите правой кнопкой мыши на проект Lab4 и выберите пункт меню New/Other, затем выберите Web/JSP и нажмите Next.
2) Укажите имя файла index.jsp и убедитесь, что в дереве структуры проекта выбран каталог WebContent. Нажмите Finish.



3) Созданный JSP-файл уже содержит базовый набор тэгов заголовка и тела веб-страницы. Нам необходимо добавить следующие элементы кода:

- поменять кодировку страницы на UTF-8 для корректного отображения символов кириллицы

- поменять заголовок страницы на «Поиск сотрудников»

- создать форму, включающую в себя текстовое поле lastname для ввода фамилии и кнопку подтверждения (Submit). При выполнении подтверждения форма передает значение параметра `lastname` сервлету `EmployeeServlet`.

- сигналом того, что поиск был выполнен и имеются результаты для отображения, будет являтся наличие параметра http-запроса `employeesFound`, который будет передан сервлетом странице index.jsp в случае успешного выполнения поиска. Далее, проверив размер коллекции найденных сотрудников, мы определим, был ли найден хотя бы один сотрудник. В случае утвердительного ответа обработаем коллекцию, и отобразим все свойства каждого найденного сотрудника в виде таблицы.

Ниже приведен полный код JSP-страницы:

```
< %@ page language="java" contentType="text/html; charset=UTF-8"
         pageEncoding="UTF-8" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
< %@page import="java.util.ArrayList" %>
< %@page import="ru.tpu.javaEElabs.lab4.Employee" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Поиск сотрудников</title>
</head>
<body>
      <form action="EmployeeServlet">
            Фамилия сотрудника
            <input type="text" name="lastname">
            <input type=submit value="поиск">
      </form>
      < %
      // Получение значения параметра employeesFound
      ArrayList employees = (ArrayList)
      request.getAttribute("employeesFound");
      // Если параметр задан, начинаем обработку
      if (employees != null) {
            // Если не найдено ни одного сотрудника – вывод сообщения
            if (employees.size()==0)
                  out.print("Сотрудники не найдены");
            else {
```

```
            out.print("<TABLE border=\"1\">");
            // Заголовок таблицы
            out.print("<TR><TD>Id</TD><TD>Имя</TD><TD>Фамилия</TD>" +
                        "<TD>Должность</TD><TD>Телефон</TD></TR>");
            for (int i = 0; i < employees.size(); i++) {
                  // По каждому найденному сотруднику
                  // формируется строка таблицы
                  out.print("<TR>");
                  // Получение очередного сотрудника из коллекции
                  Employee emp = (Employee) employees.get(i);
                  // Заполнение строки таблицы свойствами сотрудника
                  out.print("<TD>" + emp.getId() + "</TD>");
                  out.print("<TD>" + emp.getFirstName() + "</TD>");
                  out.print("<TD>" + emp.getLastName() + "</TD>");
                  out.print("<TD>" + emp.getDesignation() + "</TD>");
                  out.print("<TD>" + emp.getPhone() + "</TD>");
                  out.print("</TR>");
            }
            out.print("</TABLE>");
            }
      }
       %>
</body>
</html>
```

Доработка сервлета

Метод сервлета `doGet()` выполняется после того как пользователь выполнил подтверждение (Submit) формы. Необходимо изменить код сервлета таким образом, чтобы результаты поиска отправлялись jsp-странице в виде параметра `employeesFound`. Для этого закомментируем часть кода, связанную с выводом коллекции сотрудников в поток вывода (`response.getWriter()`), и добавим код помещения этой коллекции в параметр запроса и перенаправление запроса к странице index.jsp:

```
      /*
      // Выводим информацию о найденных сотрудниках
      PrintWriter out = response.getWriter();
      out.println("Найденные сотрудники<br>");
      for (Employee emp: employees) {
            out.print(emp.getFirstName() + " " +
                        emp.getLastName() + " " +
                        emp.getDesignation() + " " +
                        emp.getPhone() + "<br>");
      }
      */

      // Помещение результатов в параметр запроса employeesFound
      request.setAttribute("employeesFound", employees);
      // Перенаправление http-запроса на страницу index.jsp
      RequestDispatcher dispatcher = getServletContext()
                  .getRequestDispatcher("/index.jsp");
      dispatcher.forward(request, response);
```

Упаковка и развертывание приложения

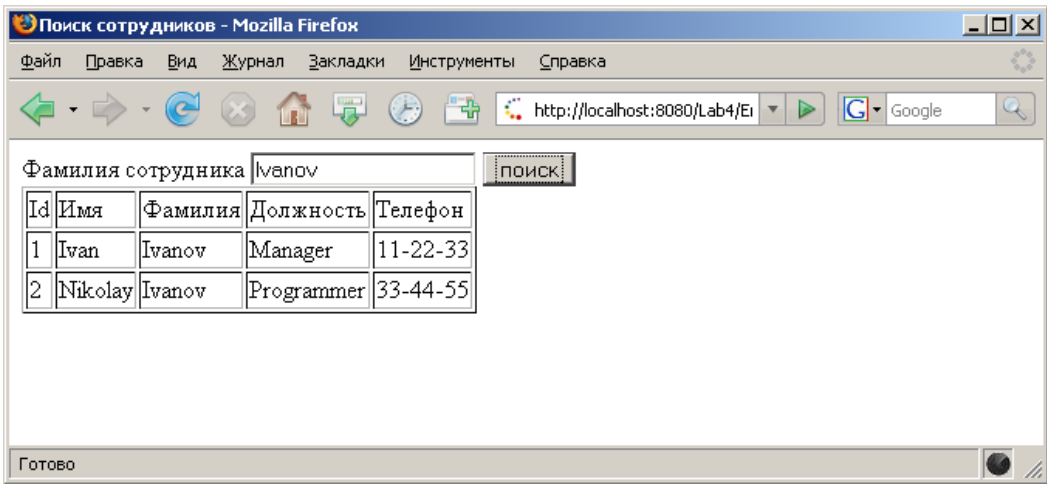
1) Щелкните правой кнопкой мыши на значок проекта Lab4 в окне Project Explorer и выберите Build Project.
2) Для развертывания приложения на сервере, щелкните правой кнопкой мыши на архив Lab4.war в окне Project Explorer и выберите команду Deploy To Server.

Тестирование приложения

Запустите броузер и перейдите по адресу `http://localhost:8080/Lab4`.

Введите фамилию сотрудника, например “Ivanov” и нажмите «Поиск». В результате на странице отображается информация о найденных сотрудниках.

Также попытайтесь выполнить поиск, оставив поле «Фамилия сотрудника» пустым.



Варианты заданий

1. Необходимо расширить функциональные возможности приложения поиска и просмотра сотрудников возможностью добавления новых сотрудников. Прежде всего, необходимо изменить код главной страницы таким образом, чтобы при отсутствии строки поиска в таблице отображались все сотрудники. Далее, необходимо создать новую страницу, содержащую форму для ввода информации о новом сотруднике и создать на нее ссылку на главной странице. Разработать сервлет для обработки параметров нового сотрудника и создания записи в БД. После создания сотрудника главная страница автоматически обновляется.

2. Необходимо расширить функциональные возможности приложения поиска и просмотра сотрудников возможностью изменения параметров сотрудников. Прежде всего, необходимо изменить код главной страницы таким образом, чтобы при отсутствии строки поиска в таблице отображались все сотрудники. Далее, необходимо создать новую страницу, содержащую форму для изменения информации о сотруднике. Эта страница открывается при щелчке на запись о сотруднике в таблице на главной странице. Разработать сервлет для обработки параметров сотрудника и обновления записи в БД. После изменения атрибутов сотрудника главная страница автоматически обновляется.

Лабораторная работа 5. Дополнительные возможности технологий Servlet и JSP: управление сессией пользователя, настройка страницы ошибок и создание клиентских тегов

Постановка задачи

Необходимо разработать веб-приложение для магазина по продаже книг, предоставляющее возможность просмотра списка книг и помещения выбранных книг в корзину. Корзина накапливает информацию о добавленных книгах и отображает общую сумму заказа. Данные о книге содержат четыре параметра: код, название, автор и цена. Вход в приложение является авторизованным, т. е. пользователь должен ввести логин/пароль для доступа к странице просмотра/выбора книг. При попытке прямого доступа к этой странице или в случае вода неверного логина/пароля осуществляется переход к странице ошибок.

Примечание: для упрощения задачи в примерах кода, приведенных в описании лабораторной работы проверка логина/пароля, а также формирование списка книг производится статически, непосредственно в коде программы. Использование для этих целей внешних хранилищ данных, например СУБД было описано в предыдущих лабораторных работах и не должно составить труда.

Для решения поставленной задачи необходимо выполнить следующие шаги:

1. Создать новый проект.
2. Разработать JSP страницу index.jsp для ввода имени пользователя и пароля и отсылки сообщения сервлету `LoginServlet`.
3. Разработать сервлет `LoginServlet` проверяющий параметры аутентификации и сохраняющий результат в сессии.
4. Разработать клиентский тег `BookTagHandler` для отображения информации о книге с кнопкой «Добавить в корзину».
5. Сформировать описание клиентского тега в библиотеке тегов BookTagLib.tld.
6. Разработать страницу ошибок errorPage.jsp.
7. Разработать страницу bookStore.jsp отображающую информацию о книгах и позволяющую формировать корзину выбранных книг. Связать страницу bookStore.jsp со страницей ошибок errorPage.jsp.
8. Упаковать приложение и развернуть на сервере.
9. Протестировать работу приложения в браузере.

Подготовительный этап

Для реализации проекта необходимо установить и настроить Ecplise, JBoss AS, JBoss Tools (см. п. «Установка и настройка программного обеспечения»).

Создание нового проекта

1) Выберите пункт меню File/New/Project, в окне выбора типа проекта укажите Web/Dynamic Web Project и нажмите Next.
2) Укажите имя проекта Lab5 и нажмите Finish.

Создание страницы index.jsp

Страница index.jsp должна содержать форму, включающую в себя два поля для ввода имени пользователя и пароля и кнопку «ОК» для подтверждения ввода. Данные формы отсылаются сервлету LoginServlet.

1) Создайте новую JSP-страницу index.jsp.
2) Скопируйте в нее следующий код:

```
< %@page contentType="text/html" pageEncoding="UTF-8" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
      charset=UTF-8">
        <title>Книжный магазин</title>
    </head>
    <body>
    <form action="LoginServlet" method="post">
      Имя пользователя: <input type="text" name="user">
    <br>
      Пароль: <input type="password" name="password">
    <br>
      <input type="submit" value="OK">
    </form>

    </body>
</html>
```

Разработка сервлета `LoginServlet`

Основной задачей сервлета `LoginServlet` является прием данных аутентификации, проверка этих данных и установка переменной user текущей сессии в случае успешной аутентификации. Наличие этой пе-

ременной будет сигнализировать другим страницам веб-приложения о том, что к ним обращается авторизованный пользователь.

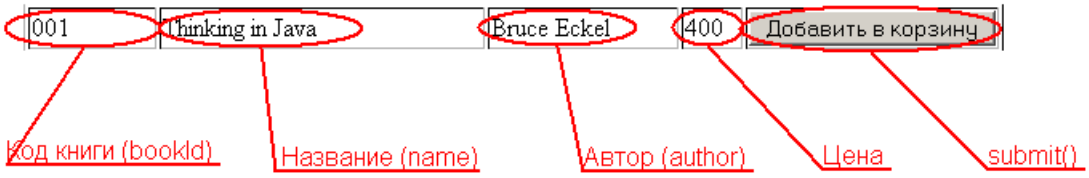
1) Создайте новый сервлет `LoginServlet` в пакете `ru.tpu.javaEELabs.lab5`.
2) Скопируйте следующий код сервлета:

```
package ru.tpu.javaEELabs.lab5;

import java.io.*;

import java.util.HashMap;
import javax.servlet.*;
import javax.servlet.http.*;

public class LoginServlet extends HttpServlet {

    protected void doPost(HttpServletRequest request,
            HttpServletResponse response)
                  throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");

        // Получаем параметры авторизации
        String username = request.getParameter("user");
        String password = request.getParameter("password");

        // Проверяем имя пользователя и пароль
        if (username.equals("someuser")
                  && password.equals("somepassword")) {
            // если логин и пароль верны,
            // получаем ссылку на текущую сессию
            HttpSession session = request.getSession(true);
            // и устанавливаем атрибут user
            session.setAttribute("user", username);
            // также создаем новую корзину товаров
            session.setAttribute("booksBasket", new HashMap());
        }
        // перенаправляем запрос на страницу выбора товаров
        RequestDispatcher dispatcher =
                getServletContext().getRequestDispatcher(
                        "/bookStore.jsp");
        dispatcher.forward(request, response);
    }
}
```

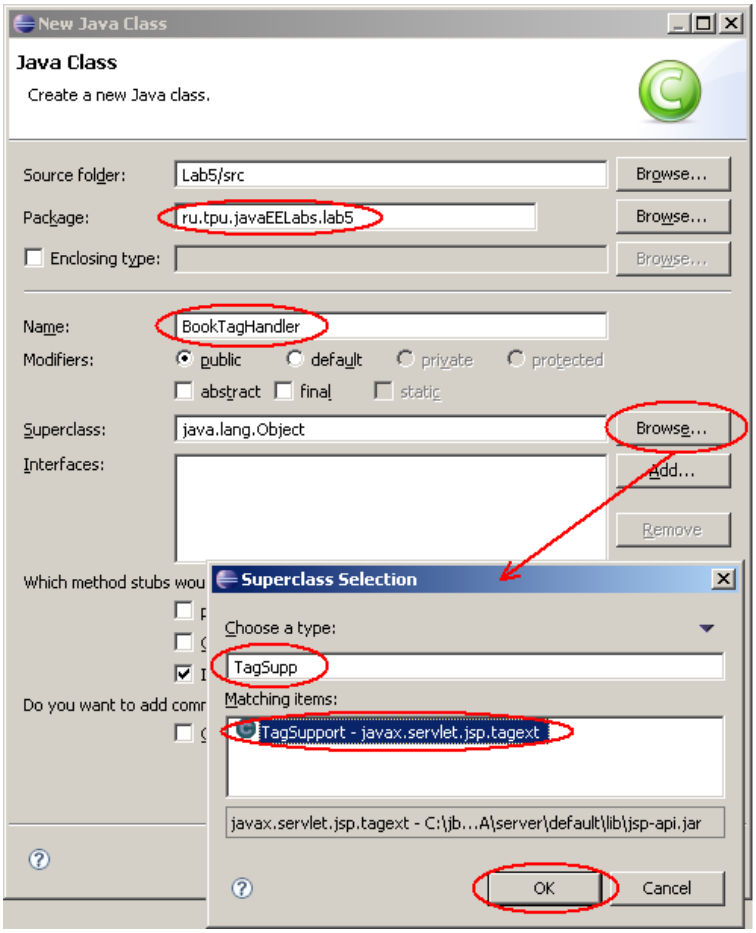
Разработка клиентского тега `BookTagHandler`

Клиентский тег `BookTagHandler` должен отображать строку таблицы (`<TR>`), поля которой отображают информацию о книге: код, название, автор, цена. Все эти параметры составят входные атрибуты клиентского тега: `bookId`, `name`, `author`, `price`.

Кроме того, тег должен представлять собой форму (<FORM>), которая отправляет код книги и ее цену при нажатии на кнопку «Добавить в корзину», расположенную в последней ячейке строки. Для того чтобы форма «знала», на какую страницу/сервлет отправлять данные формы, нам необходимо будет добавить еще один входной атрибут тега – назовем его `pageURL`.



1) Создайте новый Java-класс `BookTagHandler` в пакете `ru.tpu.javaEElabs.lab5`. При создании в поле Superclass укажите, что класс является наследником `javax.servlet.jsp.tagext.TagSupport`.



2) В классе `BookTagHandler` необходимо создать пять полей: `bookId`, `name`, `author`, `price` и `pageURL` и набор get/set методов. Затем необходимо переопределить метод `doStartTag()`, в котором с помощью потока ответа клиенту сформровать и передать содержимое тега.

   Полный код класса `BookTagHandler` приведен ниже:

```
package ru.tpu.javaEELabs.lab5;

import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.JspException;

public class BookTagHandler extends TagSupport {

    // Поля-свойства тега
    private String pageURL;
    private String bookId;
    private String name;
    private String author;
    private int price;

    public String getPageURL() {
        return pageURL;
    }

    public void setPageURL(String pageURL) {
        this.pageURL = pageURL;
    }

    public String getBookId() {
        return bookId;
    }

    public void setBookId(String bookId) {
        this.bookId = bookId;
    }

    public String getAuthor() {
        return author;
    }

    public void setAuthor(String author) {
        this.author = author;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getPrice() {
```

```
        return price;
    }

    public void setPrice(int price) {
        this.price = price;
    }

    public int doStartTag() throws JspException {
        try {
            // получаем поток ответа клиенту
            JspWriter out = pageContext.getOut();

            // заголовок формы
            out.print("<form action = \"" + pageURL +
                    "\" method=post>");
            // строка таблицы
            out.print("<tr>");

            // отображаем код книги
            out.print("<td>" + bookId + "</td>");

            // создаем скрытое поле формы для
            // передачи кода книги при подтверждении формы
            out.print("<input type = hidden name=\"bookId\" " +
                    "value=\"" + bookId + "\">");

            // отображаем название
            out.print("<td>" + name + "</td>");
            // отображаем автора
            out.print("<td>" + author + "</td>");
            // отображаем цену
            out.print("<td>" + price + "</td>");

            // создаем скрытое поле формы для
            // передачи цены книги при подтверждении формы
            out.print("<input type = hidden name=\"bookPrice\" " +
                    "value=\"" + price + "\">");

            // кнопка submit
            out.print("<td><input type=submit " +
                    "value=\"Добавить в корзину\"></td>");

            out.print("</tr>");
            out.print("</form>");
        } catch (Exception e) {
            e.printStackTrace();
        }
        return SKIP_BODY;
    }
}
```

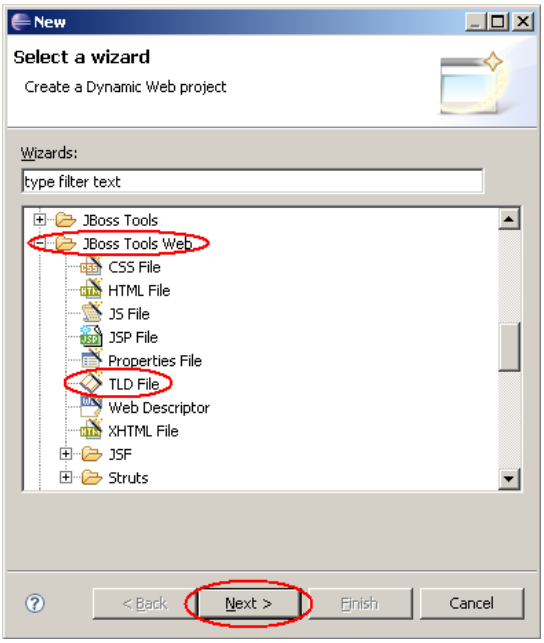
Формирование описания клиентского тега в библиотеке тегов BookTagLib.tld

Теперь нам необходимо описать параметры созданного тега в TLD-файле. Эти параметры включают в себя информацию о размеще-

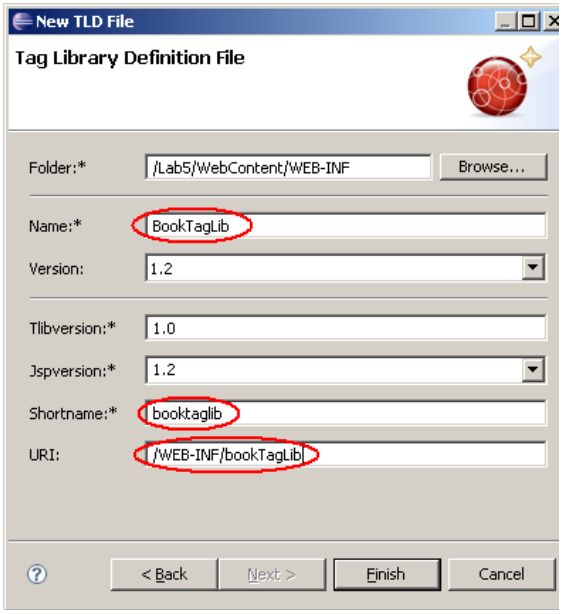
нии библиотеки тегов, имя и класс реализации клиентского тега, а также имена его входных атрибутов.

При создании TLD-файла мы будем использовать возможности мастера и редактора JBossTools.

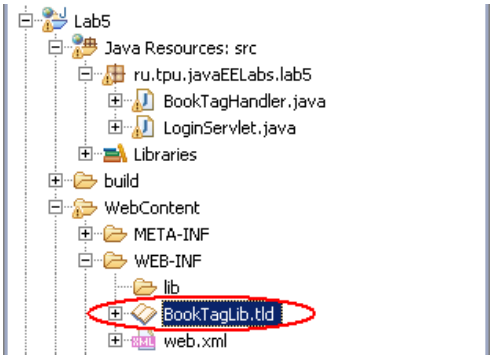
1) В окне Project Explorer щелкните правой кнопкой мыши на подкаталог WEB-INF, расположенный внутри каталога Lab5/WebContent и выберите команду New/Other.
2) В появившемся окне раскройте категорию JBoss Tools Web, выберите TLD File и нажмите Next.



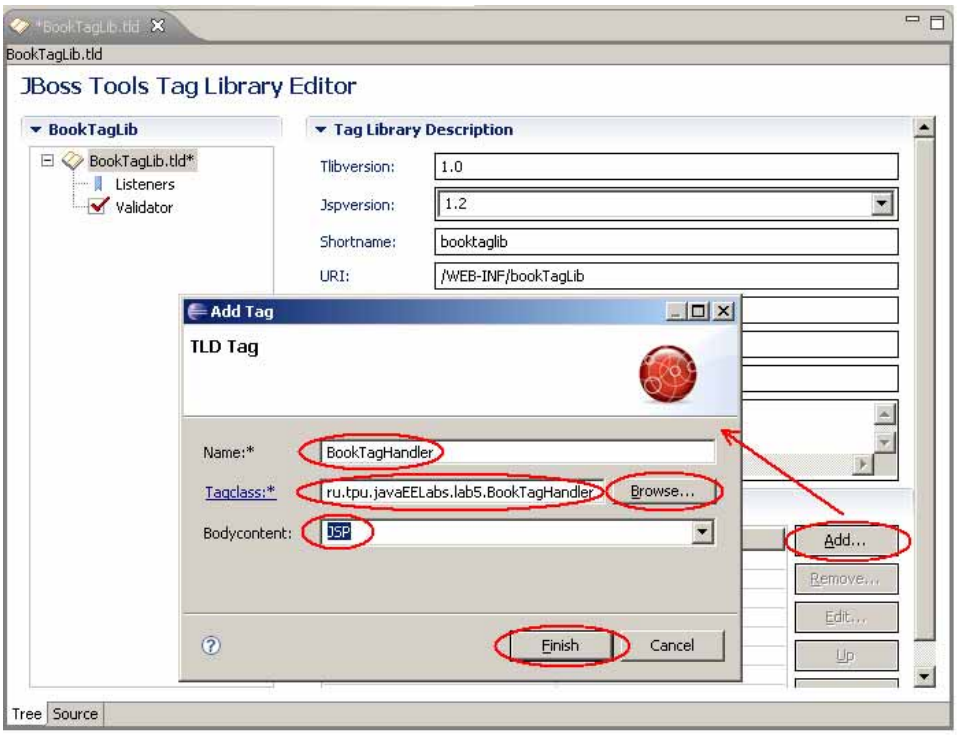
3) В следующем окне в поле имени TLD-файла (Name) укажите BookTagLib, в поле сокращенного имени (Shortname) укажите booktaglib и в поле адреса библиотеки (URI) задайте /WEB-INF/bookTagLib. Нажмите Finish.



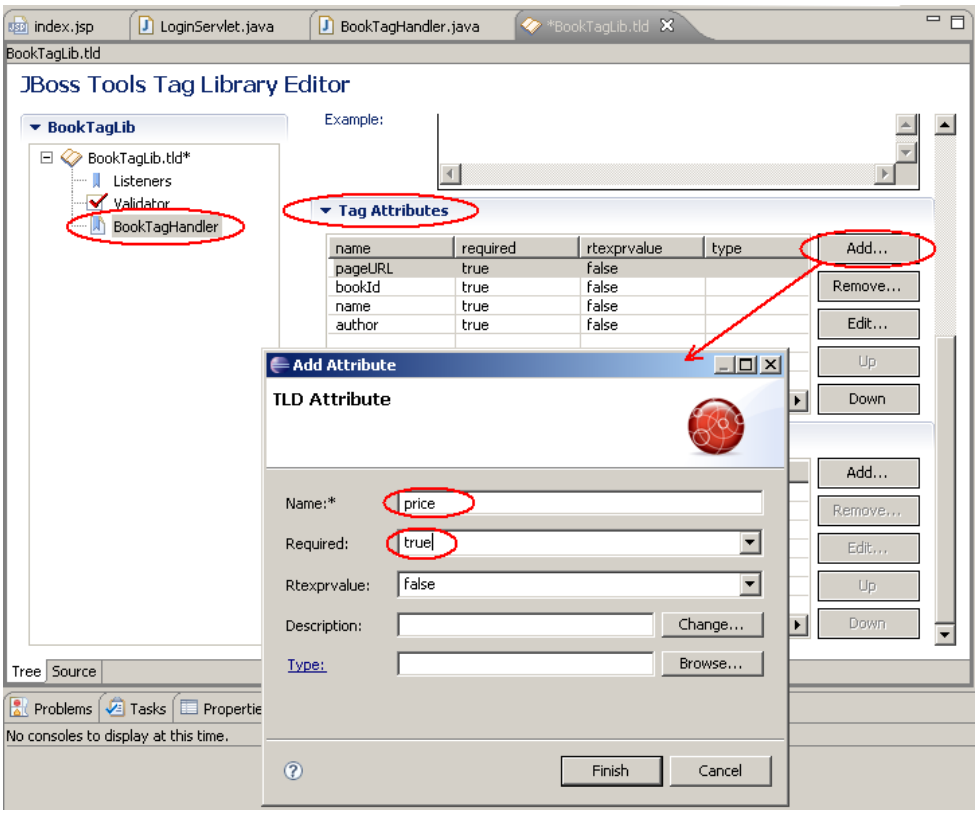
В результате созданный TLD-файл открывается в редакторе JBoss Tools Tag Library Editor. Редактор также можно открыть щелкнув дважды на TLD-файле в представлении Project Explorer.



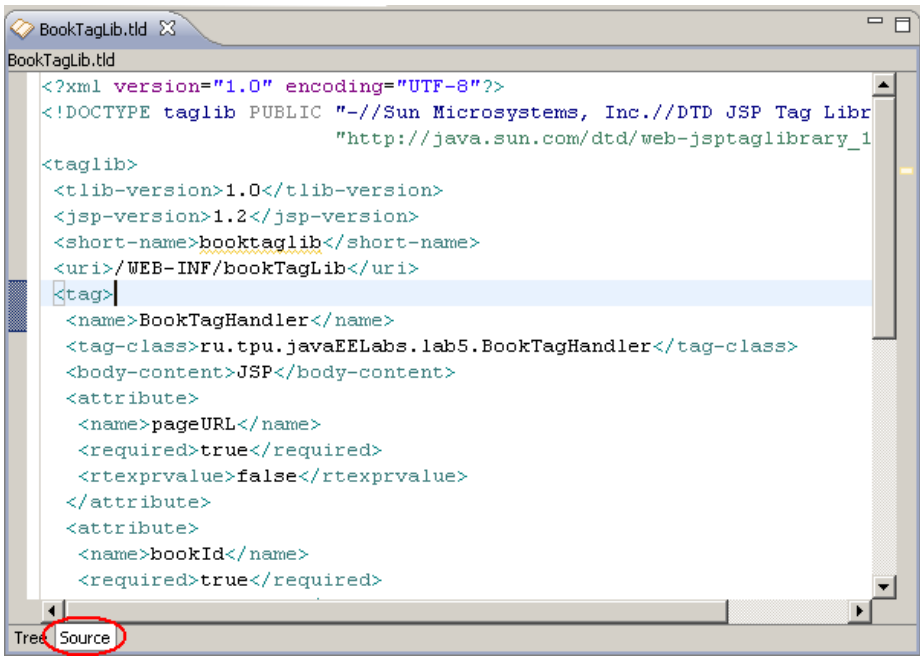
4) Для добавления в библиотеку описания тега нажмите на кнопку Add справа от списка Defined Tags.
5) В появившемся диалоге введите имя тега (Name) – BookTagHandler, затем с помощью кнопки Browse или прямым вводом укажите класс реализации тега (Tagclass) – `ru.tpu.javaEELabs.lab5.BookTagHandler` и установите тип содержимого тега (Bodycontent) в JSP. Нажмите Finish.



6) Далее нам необходимо добавить описания пяти атрибутов тега BookTagHandler: pageURL, bookId, name, author, price. Для добавления атрибута нажмите на кнопку Add справа от списка Tag Atributes. В появившемся окне укажите имя (Name) атрибута, а значение Required (обязательный) установите в true. Добавьте описания всех пяти атрибутов.



7) Нажмите Ctrl-S для сохранения TLD-файла.
8) Содержимое файла можно также просматривать и редактировать в виде исходного XML-кода. Для этого необходимо нажать на закладку Source в нижней части редактора.



Ниже приведен полный код файла BookTagLib.tld:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag
Library 1.2//EN"
"http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">
<taglib>
 <tlib-version>1.0</tlib-version>
 <jsp-version>1.2</jsp-version>
 <short-name>booktaglib</short-name>
 <uri>/WEB-INF/bookTagLib</uri>
 <tag>
  <name>BookTagHandler</name>
  <tag-class>ru.tpu.javaEELabs.lab5.BookTagHandler</tag-class>
  <body-content>JSP</body-content>
  <attribute>
   <name>pageURL</name>
   <required>true</required>
   <rtexprvalue>false</rtexprvalue>
  </attribute>
  <attribute>
   <name>bookId</name>
   <required>true</required>
   <rtexprvalue>false</rtexprvalue>
  </attribute>
  <attribute>
   <name>name</name>
   <required>true</required>
   <rtexprvalue>false</rtexprvalue>
  </attribute>
  <attribute>
   <name>author</name>
   <required>true</required>
   <rtexprvalue>false</rtexprvalue>
  </attribute>
  <attribute>
   <name>price</name>
   <required>true</required>
   <rtexprvalue>false</rtexprvalue>
  </attribute>
 </tag>
</taglib>
```

Разработка страницы ошибок errorPage.jsp

Страница ошибок представляет собой JSP-страницу, способную принимать исключения от других страниц и отображать информацию об этих исключениях. JSP-страница помечается как страница ошибок с помощью установки значения атрибута `isErrorPage` в `true`. После этого становится доступна переменная exception, содержащая ссылку на исключение, являющееся причиной вызова страницы.

1) Создайте новую jsp-страницу с именем errorPage.jsp.
2) Скопируйте в созданную страницу следующий код:

```
<%-- Параметр isErrorPage обозначает, что это страница ошибок --%>
<%@ page isErrorPage="true" pageEncoding="UTF-8" session="false" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
         "http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Ошибка</title>
</head>
<body>
      <h2>
            <%-- exception содержит ссылку на возникшее исключение --%>

            <%-- Выводим тип исключительной ситуации --%>
            <%= exception.getClass().getName() %>
            :
            <%-- Выводим текст сообщения исключительной ситуации --%>
            <%= exception.getMessage() %>
      </h2>
      <% exception.printStackTrace(); %>
</body>
</html>
```

Разработка страницы bookStore.jsp.

Разработка страницы bookStore.jsp включает в себя следующие этапы:

- связывание со страницей ошибок errorPage.jsp. Выброс исключения в случае попытки обращения к странице пользователя, не прошедшего аутентификацию (переменная сессии user не установлена);

- реализовать отображение списка книг с помощью клиентского тега `BookTagHandler`. Настроить теги таким образом, что при добавлении товара в корзину запрос отправляется этой же странице (атрибут pageURL);

- обеспечить добавление товаров в корзину, хранящуюся в сессии в виде объекта `HashMap` под именем `booksBasket`;

- обеспечить проверку состояния корзины. Если корзина не пуста, отображать количество и суммарную стоимость товаров в ней.

1) Создайте новую JSP-страницу bookStore.jsp.
2) Скопируйте в страницу следующий код:

```
<%-- Связывание со страницей ошибок --%>
<%@page errorPage="errorPage.jsp" %>

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="java.util.HashMap" %>
<%@page import="java.util.Iterator" %>
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
      "http://www.w3.org/TR/html4/loose.dtd">


<%
// Проверка наличия атрибута user в текущей сессии
if (session.getAttribute("user") == null) {
      // если атрибут отсутствует, выброс исключения,
      // которое перенаправляется странице ошибок
      throw new Exception(
      "Ошибка аутентификации: неверное имя пользователя/пароль");
}
%>

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Книжный магазин</title>
</head>
<body>
      <p><b>
      <%
      // Получаем из сессии объект - корзину товаров
      // (изначально корзина была создана в LoginServlet)
      HashMap booksBasket =
       (HashMap)session.getAttribute("booksBasket");

      // Получаем параметры запроса
      String bookId = request.getParameter("bookId");
      String bookPriceStr = request.getParameter("bookPrice");
      Integer bookPrice = bookPriceStr == null ? null :
            new Integer(bookPriceStr);

      // Если параметры не пустые
      // (т. е. произошло добавление товара в корзину)
      if (bookId != null && bookPrice != null) {
            // помещаем значения в корзину,
            // ключом элемента HashMap является код книги,
            // значением - цена книги
            // таким образом предполагается, что
            // каждая книга может быть добавлена только один раз
            booksBasket.put(bookId, bookPrice);
      }

      if (booksBasket.size() == 0) {
            out.print("Ваша корзина пуста");
      } else {
            int sum = 0;
            // если корзина не пуста, перечисляем значения HashMap
            // которые содержат цену книг и вычислем сумму
            Iterator iterator = booksBasket.values().iterator();
            while (iterator.hasNext()) {
                  Integer price = (Integer)iterator.next();
                  sum += price.intValue();
            }
            // отображаем количество и суммарную
            // стоимость товаров в корзине
            out.print("Количество товаров в корзине: " +
```

```
                    booksBasket.size());
            out.print("<br>");
            out.print("На сумму: " + sum);
        }
        %>
        </b></p>

        <!-- Подключение библиотеки тегов -->

        <%@ taglib uri="/WEB-INF/BookTagLib.tld" prefix="x" %>

        <!-- Отображение списка книг -->

        <!-- создаем таблицу -->
        <table border=1>
        <tr>
              <td>Код товара</td>
              <td>Название</td>
              <td>Автор</td>
              <td>Цена</td>
        </tr>
        <!--
        Используем клиентский тег для создания
        строки таблицы. При создании экземпляра тега
        указываем значения параметров тега.
        Параметр pageURL указывает на эту же страницу
        -->
        <x:BookTagHandler
              pageURL="bookStore.jsp"
              bookId="001"
              name="Thinking in Java"
              author="Bruce Eckel"
              price="400"/>
        <x:BookTagHandler
              pageURL="bookStore.jsp"
              bookId="002"
              name="Bitter Java"
              author="Bruce Tate"
              price="300"/>
        <x:BookTagHandler
              pageURL="bookStore.jsp"
              bookId="003"
              name="Object-Oriented Design Patterns"
              author="Erich Gamma et al."
              price="500"/>
        </table>
        </body>
</html>
```

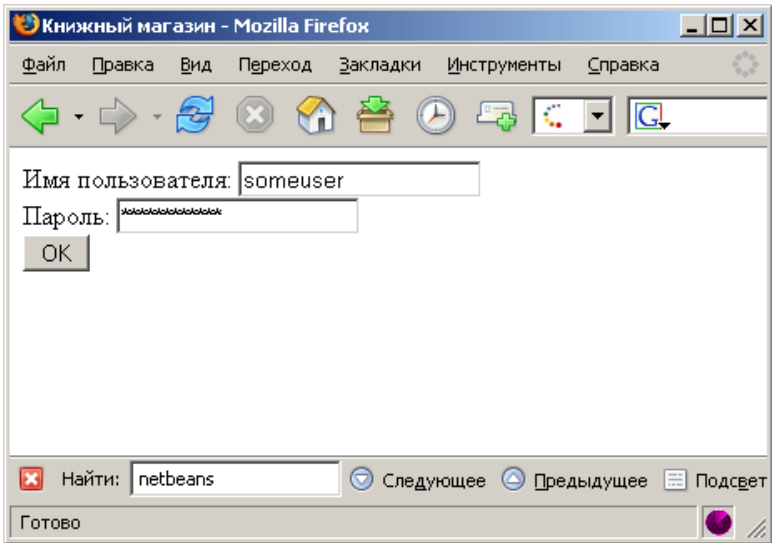
Сборка и развертывание приложения

1) Создайте WAR-архив приложения Lab5 с помощью представления Project Archives.
2) Выполните команду Build Project.
3) Запустите сервер приложений.
4) Разверните файл Lab5.war на сервере.

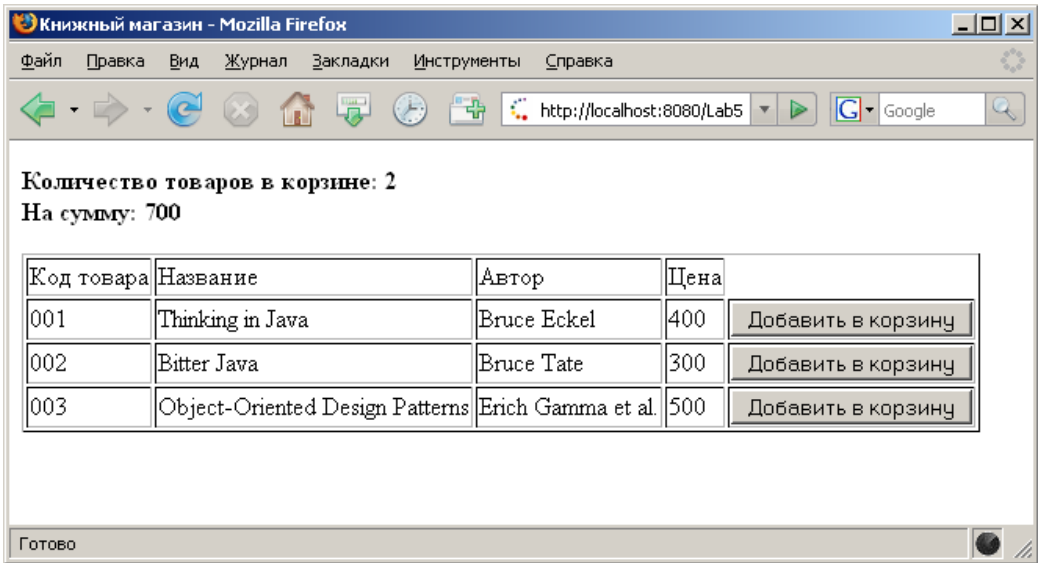
Тестирование приложения

Запустите браузер и перейдите по ссылке `http://localhost:8080/Lab5`.

Введите имя пользователя: someuser, пароль: somepassword и нажмите OK.

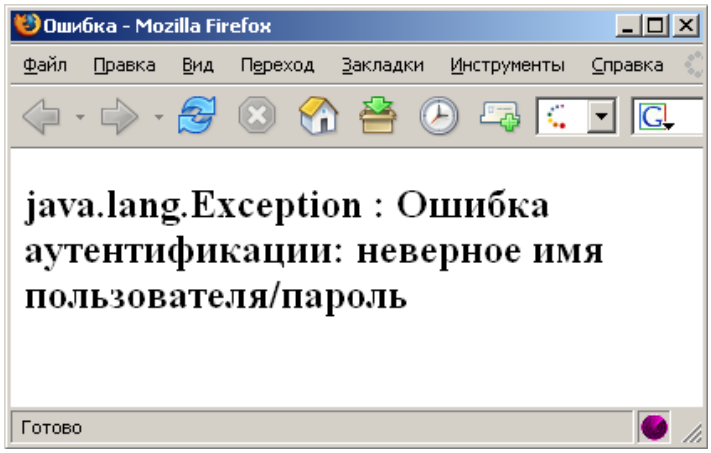


На странице выбора товаров проверьте функцию добавления товаров в корзину:



Также попробуйте ввести неверный логин/пароль при входе, либо обратиться к странице выбора товаров напрямую, минуя окно входа в систему, по ссылке `http://localhost:8080/Lab5/bookStore.jsp`.

При этом необходимо перезапустить браузер для завершения текущей сессии. В результате должна отобразиться страница ошибок следующего вида:



Варианты заданий

1. Необходимо расширить функциональные возможности приложения. Обеспечить хранение и проверку логинов и паролей пользователей в таблице БД. Таблица должна содержать три поля: логин, пароль и ФИО пользователя. Обеспечить возможность простой регистрации новых пользователей. Пользователь выбирает ссылку «Регистрация» и в новом окне вводит логин и пароль. При совпадении логина, запрос перенаправляется на страницу ошибок с соответствующим сообщением. Обеспечить хранение в сессии ФИО авторизированного пользователя и выводить в верхней части страницы приветствие формата «Добро пожаловать, <ФИО пользователя>». Реализовать приветствие в виде пользовательского тега.

2. Необходимо расширить функциональные возможности приложения. Во-первых, при добавлении товара в корзину реализовать возможность указания количества книг (ранее можно было добавить только один экземпляр каждой книги). Расширить приложение возможностями просмотра содержимого корзины и удаления товаров из корзины. Содержимое корзины должно отображаться на отдельной странице в виде списка, каждая строка которого содержит название, количество экземпляров книги, сумму и кнопку «Удалить»». Реализовать элемент списка корзины в виде пользовательского тега.

Лабораторная работа 6. Анализ и трансформация XML-документов

Постановка задачи

Компания-заказчик предоставляет сервис онлайновой продажи книг. Информация о книгах передается между клиентами и торговой организацией в виде XML-документов. Необходимо обеспечить обработку этих документов. Обработка включает в себя две задачи: анализ содержимого документа и трансформацию в файл HTML. Исходные данные для анализа предоставляются в виде файла-примера books.xml. Трансформация документа в HTML-код выполняется на основе XSL-документа books.xsl. Анализ содержимого документа в свою очередь может быть выполнен двумя способами – с помощью SAX API и DOM API. Необходимо реализовать оба способа.

Примечание: примеры программ, приведенные в этой лабораторной работе, обеспечивают лишь чтение и разбор структуры XML-документа и вывод результатов в консоль. Внешний результат работы такой программы в значительной степени повторяет содержимое исходного документа. Тем не менее, методы, используемые при реализации этих программ, являются основой решения большинства задач, связанных с чтением и анализом документов XML.

Часть 1. Анализ документа XML с помощью SAX API

Для решения поставленной задачи необходимо выполнить следующие шаги:

1. Создать новый проект.
2. Создать и заполнить содержимо исходного файла books.xml.
3. Создать Java-класс `ParseBooksSAX` для анализа содержимого документа и вывода результатов в консоль.
4. Выполнить приложение.

Подготовительный этап

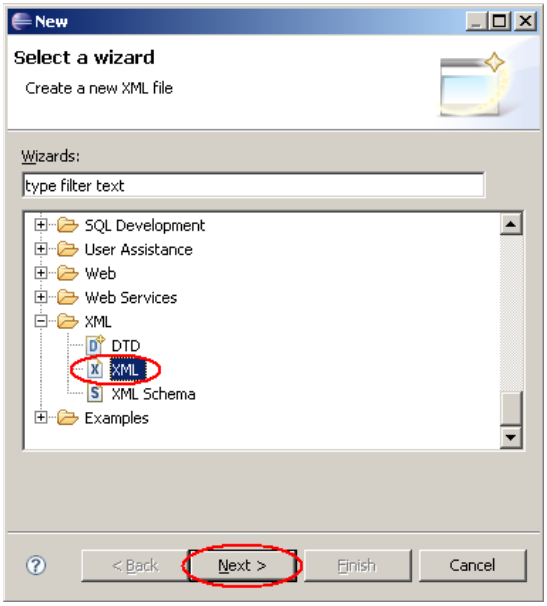
Для реализации проекта необходимо установить и настроить среду разработки Ecplise.

Создание нового проекта

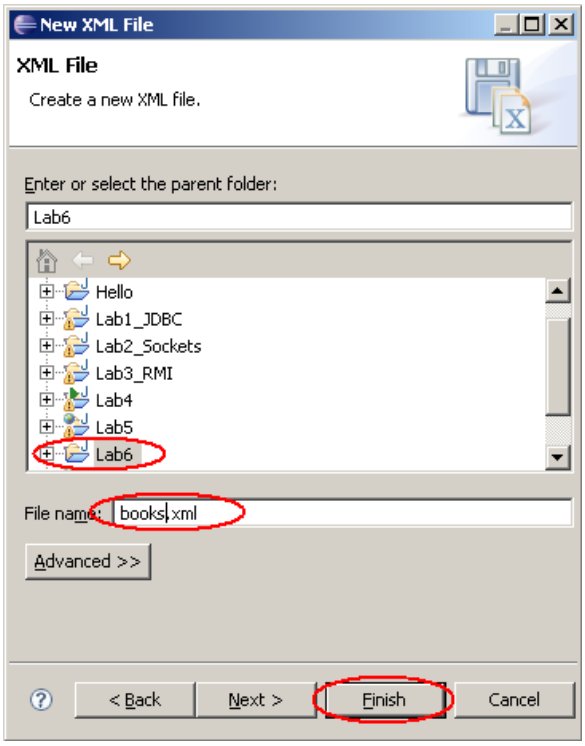
1) Выберите пункт меню File/New/Project, в окне выбора типа проекта укажите other/Java Project и нажмите Next.
2) Укажите имя проекта Lab6 и нажмите Finish.

Создание исходного файла books.xml

1) Для хранения исходного XML-документа создадим новый файл books.xml. В окне Package Explorer щелкните правой кнопкой мыши на значок проекта и выберите New/Other. В окне выбора типа ресурса укажите XML/XML и нажмите Next.



2) Укажите имя файла books.xml и нажмите Finish.



3) Скопируйте в файл следующее содержимое:

```
<?xml version="1.0" encoding="UTF-8"?>

<BookInfo>

      <Book quantity = "1">
            <BookName>Thinking in Java</BookName>
            <BookPrice>400</BookPrice>
            <BookAuthor>Bruce Eckel</BookAuthor>
      </Book>

      <Book quantity="2">
            <BookName>JavaEE Patterns</BookName>
            <BookPrice>700</BookPrice>
            <BookAuthor>Deepak Alur</BookAuthor>
      </Book>

 </BookInfo>
```

4) Сохраните файл нажатием на Ctrl-S.

Реализация класса `ParseBooksSAX`

1) Создайте новый Java-класс, нажав правой кнопкой мыши на подкаталог src и выбрав пункт меню New/Class. Назовите класс ParseBooksSAX и разместите его в пакете `ru.tpu.javaEElabs.Lab6`.

   Скопируйте код класса `ParseBooksSAX`:

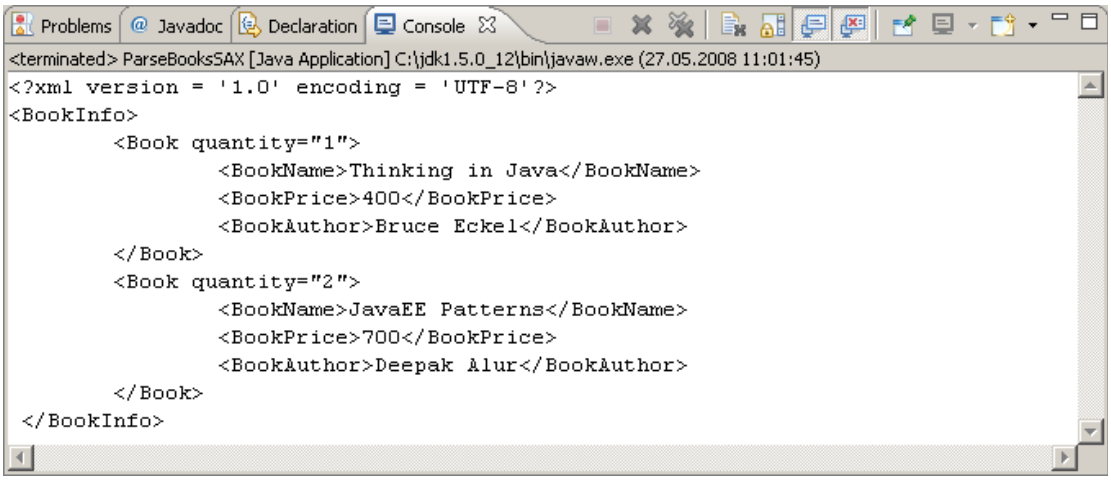
```
package ru.tpu.javaEElabs.Lab6;

import java.io.File;

import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

/**
 * Класс является наследником обработчика SAX
 * org.xml.sax.helpers.DefaultHandler
 */
public class ParseBooksSAX extends DefaultHandler {

      // Строковый буфер для накопления символов
      // текстовых элементов документа
      private StringBuffer stringBuffer;

      public static void main(String args[]) {
            String fileName = "books.xml";
```

```
      // Создаем экземпляр обработчика SAX
      DefaultHandler defaultHandler = new ParseBooksSAX();
      // Создаем экземпляр класса SAXParserFactory
      SAXParserFactory saxParserFactory =
            SAXParserFactory.newInstance();
      try {
            // Создаем экземпляр класса SAXParser
            SAXParser Sax_Parser =
                  saxParserFactory.newSAXParser();
            // Запуск парсера XML файла
            Sax_Parser.parse(new File(fileName), defaultHandler);
      } catch (Exception e) {
            e.printStackTrace();
      }
}

/**
 * Обрабатывает событие начала документа
 */
public void startDocument() throws SAXException {
      System.out.println(
      "<?xml version = '1.0' encoding = 'UTF-8'?>");
}

/**
 *  Обрабатывает событие конца документа
 */
public void endDocument() throws SAXException {}

/**
 *  Обрабатывает событие начала элемента <...>
 */
public void startElement(
            String namespaceURI,
            String localName,
            String qName,
            Attributes attrs) throws SAXException {
      displayBufferText();
      if ("".equals(localName))
            localName = qName;
      System.out.print("<" + localName);
      if (attrs != null) {
            // Получаем количество атрибутов элемента
            for (int i = 0; i < attrs.getLength(); i++) {
                  // Получаем локальное имя атрибута
                  String aName = attrs.getLocalName(i);
                  if ("".equals(aName))
                        aName = attrs.getQName(i);
                  System.out.print(" ");
                  // Получаем значение атрибута
                  System.out.print(aName +
                              "=\"" +
                              attrs.getValue(i) +
                              "\"");
            }
      }
      System.out.print(">");
```

```
	}

	/**
	 *  Обрабатывает событие конца элемента </...>
	 */
	public void endElement(
				String namespaceURI,
				String localName,
				String qName) throws SAXException {
		displayBufferText();
		if ("".equals(localName))
			localName = qName;
		System.out.print("</" + localName + ">");
	}

	/** Обрабатывает событие символьных данных
	 *  текстового элемента <...>some_text</...>
	 */
	public void characters(
				char buf[],
				int offset,
				int len) throws SAXException {
		String s = new String(buf, offset, len);
		// если строковый буфер равен null, создаем новый
		if (stringBuffer == null) {
			stringBuffer = new StringBuffer(s);
		} else {
			// Добавляем символы в строковый буфер
			stringBuffer.append(s);
		}
	}

	/**
	 *  Выводит текст, собранный в строковом буфере
	 */
	private void displayBufferText() {
		if (stringBuffer!=null) {
			System.out.print(stringBuffer.toString());
			stringBuffer = null;
		}
	}
}
```

Запуск приложения

1) Щелкните правой кнопкой мыши на класс ParseBooksSAX в окне Package Explorer и выберите команду Run As/Java Application.
2) Просмотрите результаты выполнения в представлении Console.


Problems
Javadoc
Declaration
Console
<terminated> ParseBooksSAX [Java Application] C:\jdk1.5.0_12\bin\javaw.exe (27.05.2008 11:01:45)
<?xml version = '1.0' encoding = 'UTF-8'?>
<BookInfo>
<Book quantity="1">
<BookName>Thinking in Java</BookName>
<BookPrice>400</BookPrice>
<BookAuthor>Bruce Eckel</BookAuthor>
</Book>
<Book quantity="2">
<BookName>JavaEE Patterns</BookName>
<BookPrice>700</BookPrice>
<BookAuthor>Deepak Alur</BookAuthor>
</Book>
</BookInfo>

Часть 2. Анализ документа XML с помощью DOM API

Для решения поставленной задачи необходимо выполнить следующие шаги:

1. В проекте Lab6 создать Java-класс `ParseBooksDOM` для анализа содержимого документа и вывода результатов в консоль.
2. Выполнить приложение.

Реализация класса ParseBooksDOM

1) Создайте новый Java-класс, нажав правой кнопкой мыши на пакет `ru.tpu.javaEElabs.Lab6` и выбрав пункт меню New/Class. Назовите класс ParseBooksDOM.
   Скопируйте код класса `ParseBooksDOM`:

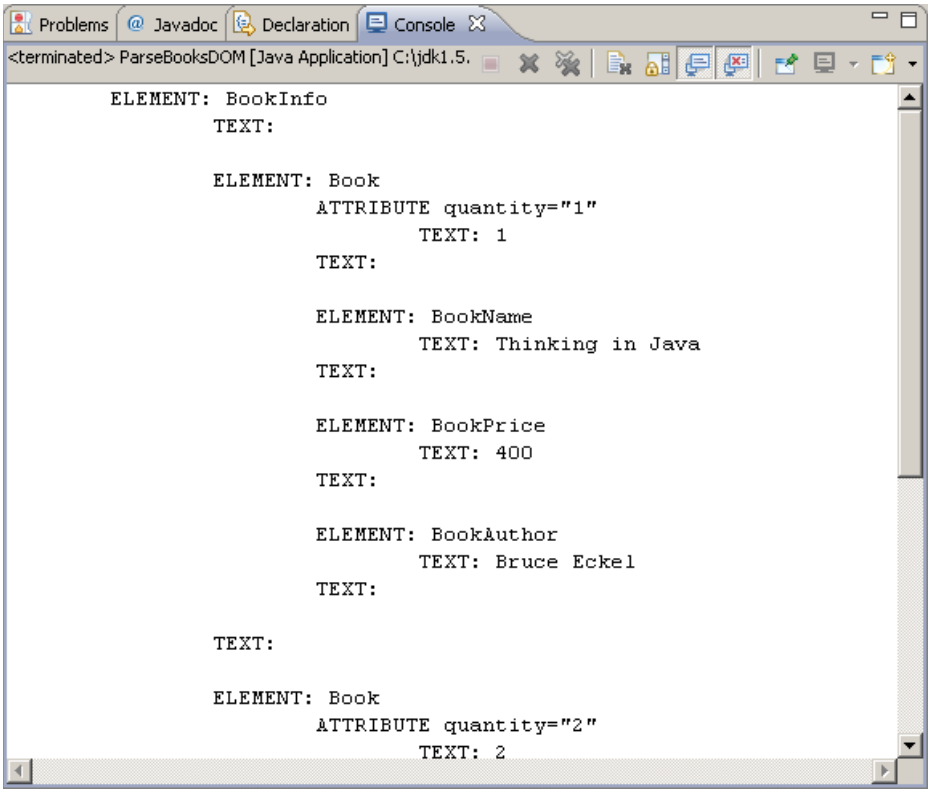
```
package ru.tpu.javaEElabs.Lab6;

import java.io.File;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Attr;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.w3c.dom.Text;

public class ParseBooksDOM {

	/**
	 * Обрабатывает элемент <some_element></some_element>
	 *
	 * @param node
	 * @param indent отступ
	 */
	private void dumpElement(Element node, String indent) {
		System.out.println(indent +
				"ELEMENT: " +
				node.getTagName());
		// Обрабатываем атрибуты элемента
		NamedNodeMap nm = node.getAttributes();
		for (int i = 0; i < nm.getLength(); i++)
			printNode(nm.item(i), indent + "    ");
	}

	/**
	 * Обрабатывает атрибут <... some_atribute = "some_value">
```

```
 *
 * @param node
 * @param indent отступ
 */
private void dumpAttributeNode(Attr node, String indent) {
      System.out.println(indent +
                  "ATTRIBUTE " +
                  node.getName() + "=\"" +
                  node.getValue() + "\"");
}

/**
 * Обрабатывает текстовое содержимое
 * элемента <...>some_text</...>
 *
 * @param node
 * @param indent отступ
 */
private void dumpTextNode(Text node, String indent) {
      System.out.println(indent + "TEXT: " + node.getData());
}

/**
 * Рекурсивная процедура обработки узлов
 * XML-документа
 *
 * @param node обрабатываемый узел
 * @param indent отступ
 */
private void printNode(Node node, String indent) {
      // Получаем тип узла
      int type = node.getNodeType();

      switch (type) {
      case Node.ATTRIBUTE_NODE:
            dumpAttributeNode((Attr) node, indent);
            break;
      case Node.ELEMENT_NODE:
            dumpElement((Element) node, indent);
            break;
      case Node.TEXT_NODE:
            dumpTextNode((Text) node, indent);
            break;

      }

      // Рекурсивно обрабатываем дочерние узлы
      NodeList list = node.getChildNodes();
      for (int i = 0; i < list.getLength(); i++)
            printNode(list.item(i), indent + "  ");
}

/**
 * Объявление метода main
 *
 * @param args
 * @throws Exception
 */
```

```
    public static void main(String[] args) throws Exception {
          try {
                String filename = "books.xml";

                // Создаем экземпляр класса DocumentBuilderFactory
                DocumentBuilderFactory factory =
                      DocumentBuilderFactory.newInstance();
                // Создаем экземпляр класса DocumentBuilder
                DocumentBuilder docBuilder =
                      factory.newDocumentBuilder();

                // Запускаем анализ входного файла
                Document doc = docBuilder.parse(new File(filename));

                // Выводим дерево DOM на экран
                new ParseBooksDOM().printNode(doc, "");

          } catch (Exception e) {
                e.printStackTrace();
          }
    }
}
```

Запуск приложения

1) Щелкните правой кнопкой мыши на класс `ParseBooksDOM` в окне Package Explorer и выберите команду Run As/Java Application.
2) Просмотрите результаты выполнения в представлении Console.



Часть 3. Трансформация XML-документа в HTML

Для решения поставленной задачи необходимо выполнить следующие шаги:

1. Создать XSL-документ books.xsl.
2. В проекте Lab6 реализовать Java-класс `TransformBooksToHTML` который читает содержимое документа books.xml и при помощи стилей описанных в books.xsl создает файл books.html.
3. Выполнить приложение и просмотреть содержимое сгенерированного файла books.html.

Создание исходного файла стилей books.xsl

1) Для хранения стилей XML-документа создадим новый файл books.xsl. В окне Package Explorer щелкните правой кнопкой мыши на значок проекта и выберите New/Other. В окне выбора типа ресурса укажите XML/XML и нажмите Next.
2) Укажите имя файла books.xsl и нажмите Finish.
3) Стили XSL описываю как выглядит тот или иной элемент XML-документа. Содержимое файла стилей books.xsl приведено ниже:

```
<?xml version="1.0"?>
<xsl:stylesheet  xmlns:xsl=  "http://www.w3.org/1999/XSL/Transform"
version= "1.0">
<xsl:output method="html" indent="yes"/>
<xsl:template match="/">
<html>
  <body>
       <xsl:apply-templates/>
  </body>
</html>
</xsl:template>
<xsl:template match="BookInfo">
   <table border="2" width="100 %" >
       <tr bgcolor="LIGHTBLUE">
            <td>Book Name</td>
            <td>Book Price</td>
            <td>Author Name</td>
            <td>Quantity</td>
       </tr>
      <xsl:for-each select="Book">
            <tr bgcolor="LIGHTYELLOW">
                 <td><i><xsl:value-of select="BookName"/></i></td>
                 <td><xsl:value-of select="BookPrice"/></td>
```

```
                        <td><xsl:value-of select="BookAuthor"/></td>
                        <td bgcolor="LIGHTGREEN">
                        <xsl:value-of select="@quantity"/>
                  </td>
                  </tr>
            </xsl:for-each>
          </table>
        </xsl:template>
      </xsl:stylesheet>
```

4) Сохраните файл нажатием на Ctrl-S.

Реализация класса TransformBooksToHTML

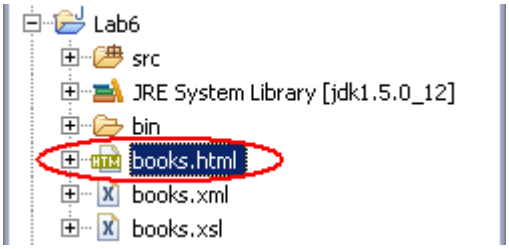
Создайте новый Java-класс, нажав правой кнопкой мыши на пакет `ru.tpu.javaEElabs.Lab6` и выбрав пункт меню New/Class. Назовите класс TransformBooksToHTML.

Скопируйте код класса `TransformBooksToHTML`:

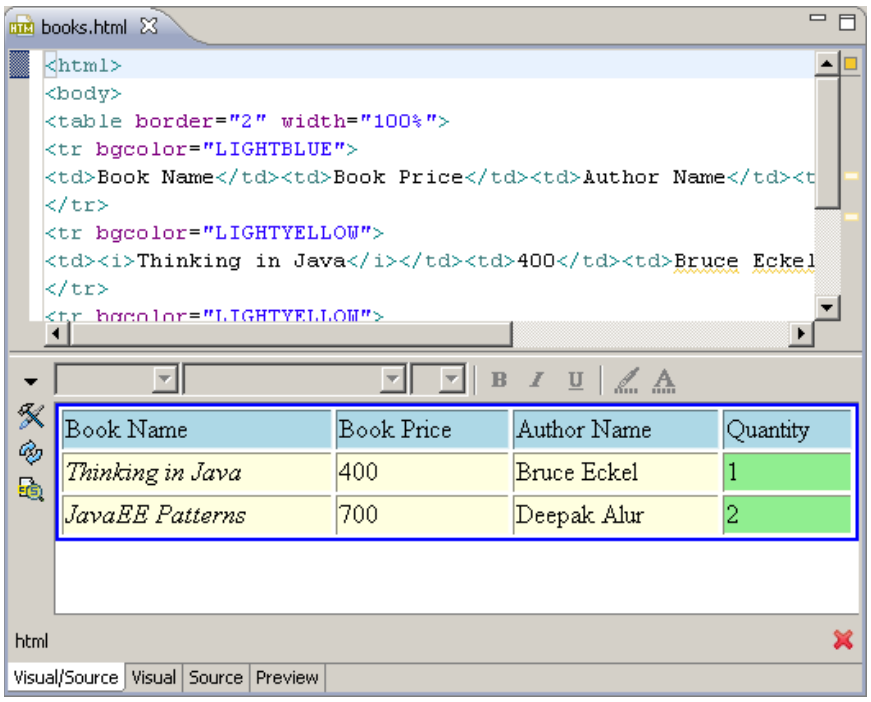
```
package ru.tpu.javaEElabs.Lab6;

import javax.xml.transform.*;
import javax.xml.transform.stream.*;
import java.io.*;

public class TransformBooksToHTML {
      public static void main(String[] args)
            throws TransformerException,
                  TransformerConfigurationException,
                  FileNotFoundException, IOException {

            String inputFile = "books.xml";
            String xslFile = "books.xsl";
            String outputFile = "books.html";

            // Создаем экземпляр TransformerFactory
            TransformerFactory transFactory =
                  TransformerFactory.newInstance();

            // Создаем экземпляр Transformer
            Transformer transformer =
                  transFactory.newTransformer(new StreamSource(xslFile));

            // Выполняем трансформацию
            transformer.transform(
                  new StreamSource(inputFile),
                  new StreamResult(new FileOutputStream(outputFile)));

            // Выводим сообщение о том, что результат записан в файл
            System.out.println("Генерация HTML-файла " +
                        outputFile + " завершена");
      }
}
```

Запуск приложения

1) Щелкните правой кнопкой мыши на класс TransformBooksToHTML в окне Package Explorer и выберите команду Run As/Java Application. В случае успешного выполнения в консоли должно отобразиться сообщение.
2) Выделите значок проекта Lab6 в представлении Package Explorer и нажмите F5 (Обновить). В результате среди элементов проекта отображается сгенерированный файл books.html.



3) Двойным щелчком откройте файл books.html во внутреннем html-редакторе/браузере Eclipse.



4) Сгенерированный файл размещается на диске в каталоге проекта Lab6 и может быть просмотрен в любом внешнем браузере.

Варианты заданий

1. Исходные данные для приложения поставляются в виде XML-документа, содержащего информацию о поездах, пункты отправления и назначения, времени отправления и прибытия. Приложение должно реализовывать поиск рейсов исходя из пункта отправления и пункта назначения. Необходимо создать исходный файл, получить данные о пунктах отправления и назначения, введенные пользователем, проанализировать исходный файл (выбор API анализа – на усмотрение разработчика), и представить результаты поиска в виде HTML-файла.

2. Исходные данные для приложения поставляются в виде XML-документа, содержащего информацию о расписании деловых встреч: дата и время встречи, с кем встреча и место встречи. Приложение должно реализовывать поиск встреч на указанную пользователем дату. Необходимо создать исходный файл, получить дату, введенную пользователем, проанализировать исходный файл (выбор API анализа – на усмотрение разработчика), и представить результаты поиска в виде HTML-файла.

Лабораторная работа 7. Создание веб-сервисов на базе спецификации JAX-WS

Постановка задачи

Необходимо разработать веб-сервис, предоставляющий пользователям информацию о погодных условиях в городах мира. Информация о погоде храниться в БД и включает в себя три характеристики: температура, облачность, осадки. Пользователю-клиенту предоставляется веб-интерфейс где он имеет возможность выбрать город и получить информацию о погоде в этом городе.

Таким образом, архитектура системы на логическом уровне состоит из четырех слоев, где каждый последующий слой является клиентом предыдущего. В скобках указаны соответствующее ПО для размещения и поддержки задачи:

1. База данных (Derby DB).
2. Веб-сервис JAX-WS (JBoss WS, встроенный в JBoss AS).
3. Веб-приложение, состоящее из JSP-страниц и сервлетов (Tomcat, встроенный в JBoss AS).
4. Рабочее место клиента (броузер).

Физически каждый слой может размещаться на отдельном компьютере и взаимодействовать с другими слоями по сети. В нашем случае все четыре слоя будут выполняться на одном компьютере.

Для решения поставленной задачи необходимо выполнить следующие шаги:

1. Создать новый проект для веб-сервиса.
2. Создать таблицу `weather` и заполнить ее данными о городах и погодных условиях.
3. Создать Java-класс WeatherInfo для представления и данных о погоде.
4. Разработать веб-сервис `WeatherWS`, содержащий метод `getWeatherInfo()`, принимающий в качестве параметра название города и возвращающий данные в виде объекта `WeatherInfo`. Также сервис должен содержать метод `getCities()` для получения списка городов.
5. Развернуть веб-сервис на сервере приложений.
6. Создать новый проект для веб-клиента.
7. Используя WSDL-описание развернутого веб-сервиса сгенерировать классы-заглушки для доступа к веб-сервису.

8. Разработать класс-фильтр, обеспечивающий получение от веб-сервиса списка городов и передачу его JSP-странице.
9. Разработать JSP страницу для ввода данных и отображения результатов.
10. Разработать сервлет, обеспечивающий вызов веб-сервиса для получения информации о погоде по выбранному городу.
11. Упаковать веб-приложение и развернуть на сервере.
12. Протестировать работу приложения в броузере

Подготовительный этап

Для реализации проекта необходимо установить и настроить Ecplise, JBoss AS, JBoss Tools, Apache Derby, Derby Plugins и SOAPUI Plugin (см. п. «Установка и настройка программного обеспечения»).

Создание нового проекта для веб-сервиса

1) Выберите пункт меню File/New/Project, в окне выбора типа проекта укажите Web/Dynamic Web Project и нажмите Next.
2) Укажите имя проекта Lab7_WebService. В полях Target Runtime и Configurations должна быть выбрана конфигурация сервера JBoss. Нажмите Finish.

Создание таблицы `weather` и вставка тестовых данных

1) Подключитесь к БД Derby и запустите сервер БД (см. п. «Установка и настройка программного обеспечения», пп. 8 «Запуск и остановка Apache Derby»).
2) Для хранения SQL-скриптов создадим новый файл weather.sql. В окне Project Explorer щелкните правой кнопкой мыши на значок проекта Lab7_WebService и выберите New/File, укажите имя файла weather.sql и нажмите Finish.
3) Созданный файл автоматически открывается для редактирования. Скопируйте в файл следующие команды:

```
-- подключение
connect
'jdbc:derby://localhost:1527/myDB;create=true;user=me;password=mine';

-- раскоментируйте следующую строку, если требуется пересоздать таблицу
drop table weather;

-- создание таблицы
create table weather(city varchar(20), temperature integer, cloudiness
varchar(20), precipitations varchar(20));

--вставка тестовых данных
```

```
insert into weather values ('St. Petersburg', -10, 'heavy', 'snow');
insert into weather values ('London', 0, 'average', 'rain');
insert into weather values ('Tokyo', 10, 'heavy', 'rain');
insert into weather values ('Paris', 5, 'light', 'no');

-- выбрать все из таблицы для проверки
select * from weather;

-- отключение и выход
disconnect;
exit;
```

4) Сохраните файл нажатием на Ctrl-S
5) Щелкните правой кнопкой мыши на файл weather.sql в окне Project Explorer и выберите Apache Derby/Run SQL Script using ‘ij’.
6) В случае успешного выполнения скрипта в консоли выводится следующее:

```
ij version 10.3
ij> -- подключение
connect
'jdbc:derby://localhost:1527/myDB;create=true;user=me;password=mine';
ij> -- раскомментируйте следующую строку, если требуется пересоздать
таблицу
drop table weather;
0 rows inserted/updated/deleted
ij> -- создание таблицы
create table weather(city varchar(20), temperature integer, cloudiness
varchar(20), precipitations varchar(20));
0 rows inserted/updated/deleted
ij> --вставка тестовых данных
insert into weather values ('St. Petersburg', -10, 'heavy', 'snow');
1 row inserted/updated/deleted
ij> insert into weather values ('London', 0, 'average', 'rain');
1 row inserted/updated/deleted
ij> insert into weather values ('Tokyo', 10, 'heavy', 'rain');
1 row inserted/updated/deleted
ij> insert into weather values ('Paris', 5, 'light', 'no');
1 row inserted/updated/deleted
ij> -- выбрать все из таблицы для проверки
select * from weather;
CITY                |TEMPERATURE|CLOUDINESS          |PRECIPITATIONS
--------------------------------------------------------------------------
--
St. Petersburg      |-10        |heavy               |snow
London              |0          |average             |rain
Tokyo               |10         |heavy               |rain
Paris               |5          |light               |no

4 rows selected
ij> -- отключение и выход
disconnect;
ij> exit;
```

Создание Java-класса `WeatherInfo` для представления и данных о погоде

1) Щелкните правой кнопкой мыши на значок проекта Lab7_WebService и выберите New/Class. В качестве имени класса (Name) укажите WeatherInfo, а в качестве имени пакета (Package) задайте `ru.tpu.JavaEELabs.Lab7ws`.
2) Класс `WeatherInfo` должен содержать четыре поля, соответствующих столбцам таблицы weather. Так как сложный объект WeatherInfo предназначен для трансляции в SOAP-сообщение для последующей передачи по сети, то он должен удовлетворять следующим основным требованиям:

   - содержать конструктор по умолчанию;
   - содержать поля простых типов, либо сложных типов, подчиняющихся данным правилам;
   - соответствовать требованиям спецификации JavaBeans, т. е. реализовывать методы get/set для доступа к полям;
   - не содержать статических полей и методов.

   Полный код класса приведен ниже:

```
package ru.tpu.JavaEELabs.Lab7ws;

public class WeatherInfo {

	private String city;
	private int temperature;
	private String cloudiness;
	private String precipitations;

	// Обязательный конструктор по умолчанию
	public WeatherInfo() {}

	public WeatherInfo(String city, int temperature,
	                   String cloudiness, String precipitations) {
		this.city = city;
		this.temperature = temperature;
		this.cloudiness = cloudiness;
		this.precipitations = precipitations;
	}

	public String getCity() {
		return city;
	}

	public void setCity(String city) {
		this.city = city;
	}
```

```
	public int getTemperature() {
		return temperature;
	}

	public void setTemperature(int temperature) {
		this.temperature = temperature;
	}

	public String getCloudiness() {
		return cloudiness;
	}

	public void setCloudiness(String cloudiness) {
		this.cloudiness = cloudiness;
	}

	public String getPrecipitations() {
		return precipitations;
	}

	public void setPrecipitations(String precipitations) {
		this.precipitations = precipitations;
	}
}
```

Разработка веб-сервиса WeatherWS

1) Щелкните правой кнопкой мыши на пакет `rpu.tpu.JavaEELabs.Lab7ws` и выберите New/Class. Создайте новый класс с именем WeatherWS.
2) Кодирование класса WeatherWS включает в себя следующие действия:

- добавление аннотации `@WebService` перед заголовком класса, указывающей на то, что созданный класс представляет собой веб-сервис;

- реализация метода `getWeatherInfo()`, принимающего в качестве параметра название города и возвращающего данные в виде объекта WeatherInfo;

- добавление аннотации `@WebMethod` перед заголовком метода getWeatherInfo, указывающей на то, что данный метод будет зарегистрирован в WSDL-файле и является методом веб-сервиса, доступным для вызова удаленными клиентами.

Полный код класса WeatherWS приведен ниже

```
package ru.tpu.JavaEELabs.Lab7ws;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.List;
```

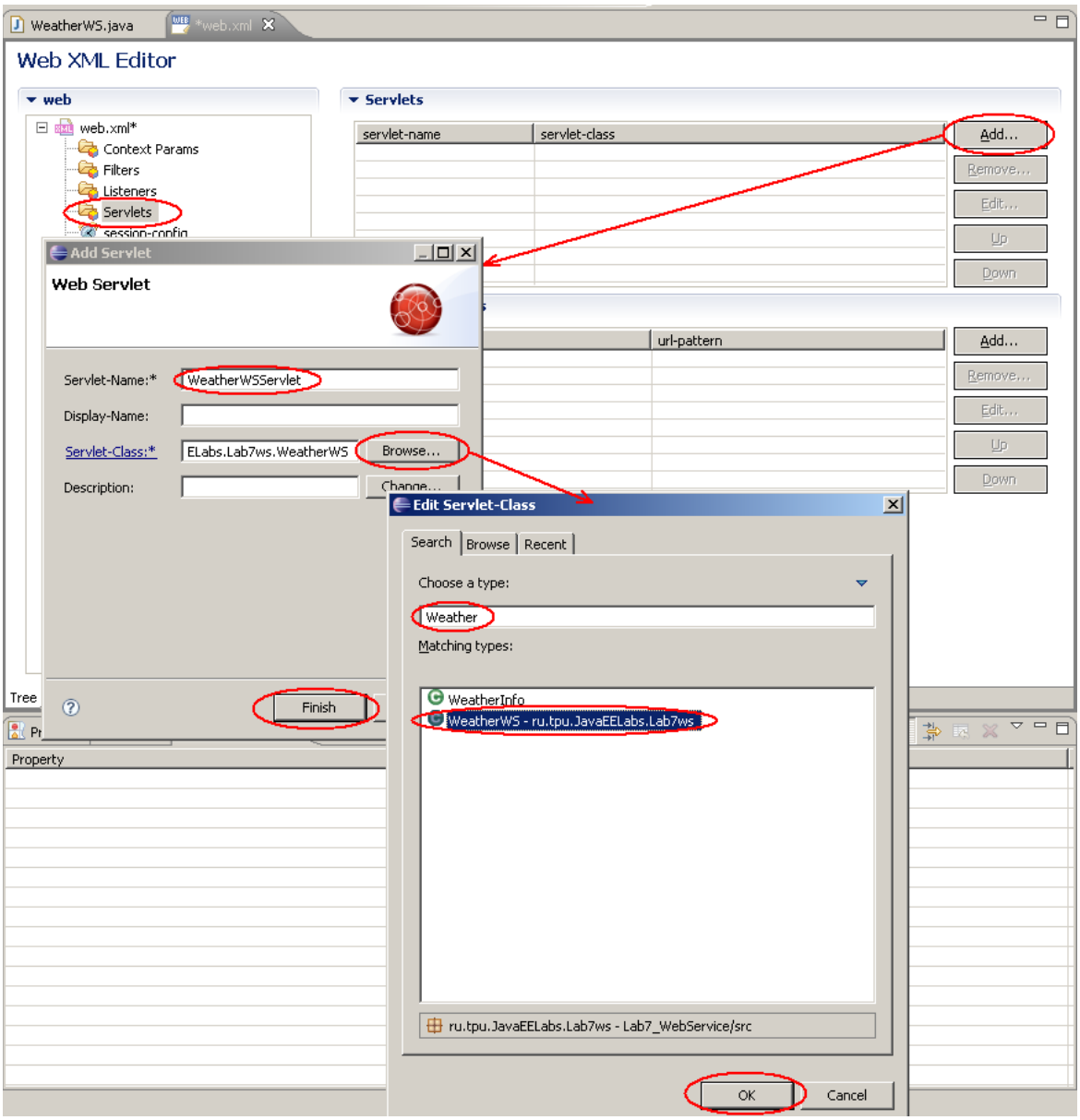
```
import javax.jws.WebMethod;
import javax.jws.WebService;

@WebService
public class WeatherWS {

	// Вспомогательный метод получения соединения
	private Connection getConnection() throws Exception {
		// Подгрузка драйвера БД Derby

	Class.forName("org.apache.derby.jdbc.ClientDriver").newInstance();
		// Получение соединения с БД
		return DriverManager.getConnection(
"jdbc:derby://localhost:1527/myDB;create=true;user=me;password=mine");
	}

	/**
	 * Возвращает список городов, по которым
	 * имеется информация о погоде
	 * @return список городов
	 */
	@WebMethod
	public List<String> getCities() {
		try {
			List<String> cities = new ArrayList<String>();
			// Получение соединения с БД
			Connection con = getConnection();

			// Выполнение SQL-запроса
			ResultSet rs = con.createStatement().executeQuery(
					"Select DISTINCT city From weather");
			while (rs.next()) {
				cities.add(rs.getString(1));
			}
			// Закрываем выборку и соединение с БД
			rs.close();
			con.close();
			return cities;
		} catch (Exception ex) {
			ex.printStackTrace();
			return null;
		}
	}

	/**
	 * Возвращает информацию о погоде в указанном городе
	 * @param city название города
	 * @return информация о погоде
	 */
	@WebMethod
	public WeatherInfo getWeatherInfo(String city) {
		try {
			WeatherInfo weatherInfo = null;

			// Получение соединения с БД
			Connection con = getConnection();
```

```
                // Выполнение SQL-запроса
                ResultSet rs = con.createStatement().executeQuery(
                      "Select city, temperature, " +
                      "cloudiness, precipitations " +
                      "from weather " +
                      "where city like '" + city + "'");
                if (rs.next()) {
                      // Формирование нового объекта класса WeatherInfo
                      // на основе данных выбранной записи
                      weatherInfo = new WeatherInfo(
                      rs.getString(1),
                      rs.getInt(2),
                      rs.getString(3),
                      rs.getString(4));
                }
                // Закрываем выборку и соединение с БД
                rs.close();
                con.close();

                return weatherInfo;
          } catch (Exception ex) {
                ex.printStackTrace();
                return null;
          }
    }
}
```

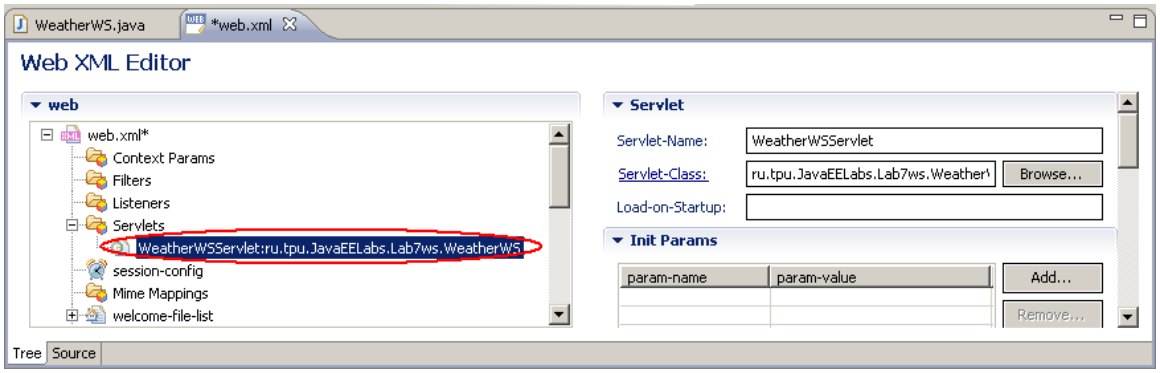
Подготовка веб-сервиса к упаковке и развертыванию

Веб-сервисы, реализованные в виде обычного java-класса (POJO – plain old java object), упаковываются в веб-приложение (war-архив) и развертываются в веб-контейнере сервера приложений. Веб-контейнер представляет веб-сервис в качестве сервлета, который обеспечивает транспортный (HTTP) уровень обмена сообщениями с удаленными клиентами и выполняет инициацию методов реализации веб-сервиса. Для того чтобы веб-сервис мог быть запущен на сервере приложений, необходимо описать параметры представляющего его сервлета в файле конфигурации web.xml.

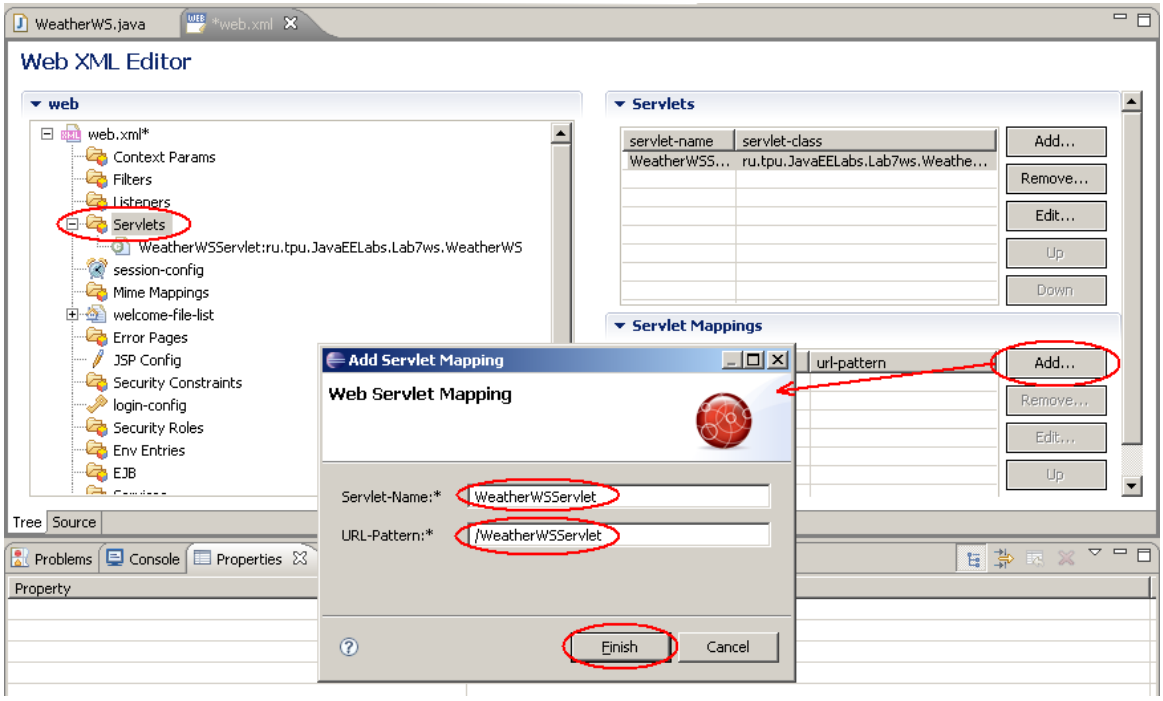
1) В представлении ProjectExplorer разверните каталог проекта Lab7_WebService и откройте элемент WebContent/WEB-INF/web.xml. В результате открывается редактор Web XML Editor в котором отображается структура документа web.xml.
2) Для того чтобы добавить описание нового сервлета, выделите элемент Servlet и нажмите на кнопку Add справа от таблицы Servlets.
3) В появившемся окне введите имя сервлета (Servlet-Name) WeatherWSServlet и с помощью кнопки Browse укажите в качестве класса реализации сервлета (Servlet-Class) класс веб-сервиса WeatherWS и нажмите Finish.



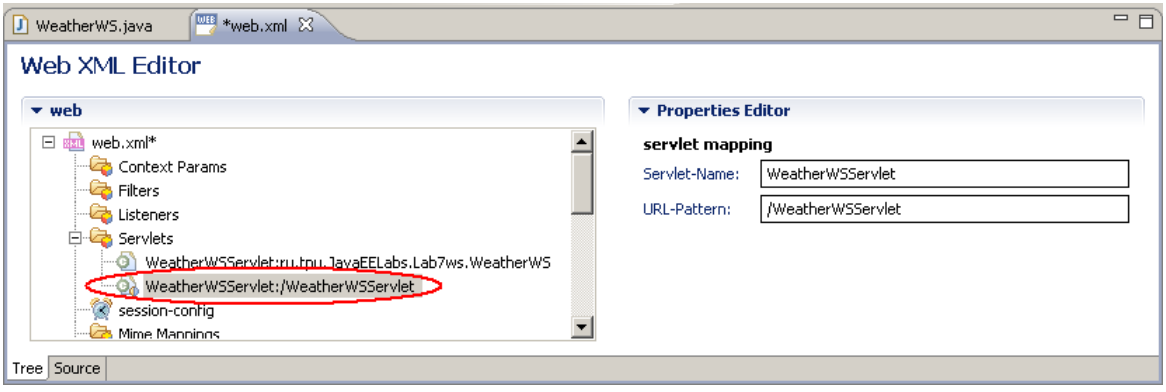
4) Созданное описание сервлета отображается в виде элемента узла Servlets.



5) Для того чтобы описать URL ссылку на сервлет, снова выделите элемент Servlets и нажмите кнопку Add справа от таблицы Servlet Mappings.
6) В появившемся окне укажите имя сервлета WeatherWSServlet, относительный URL адрес сервлета /WeatherWSServlet и нажмите Finish.

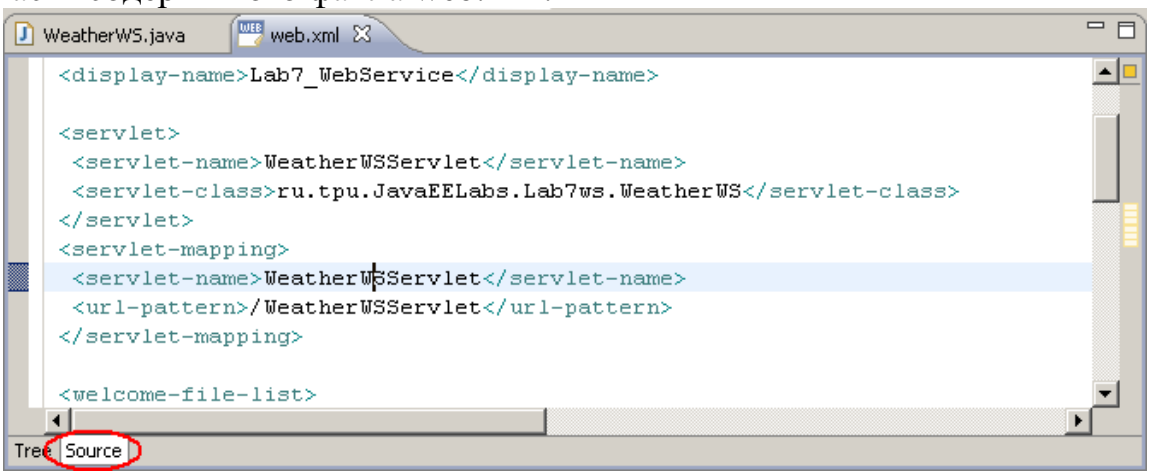


7) Созданное отображение сервлета (servlet mapping) отражается в виде узла элемента Servlets.



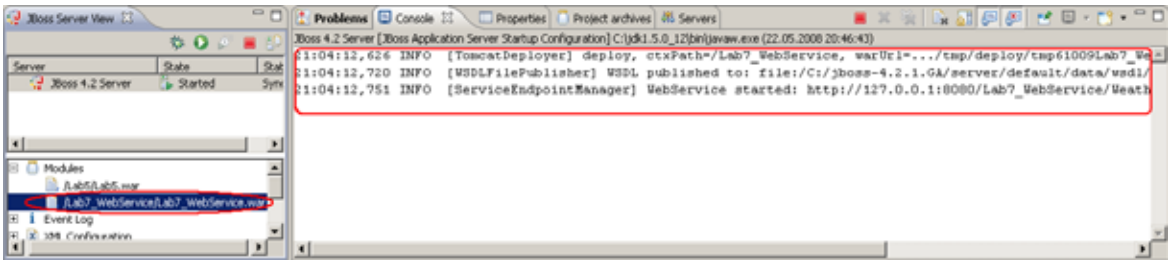
8) Нажмите Ctrl-S для сохранения файла web.xml.

Содержимое файла web.xml может быть отредактировано вручную. Для этого выберите закладку Source в нижней части редактора Web XML Editor. На следующем рисунке показано сгенерированная часть содержимого файла web.xml.



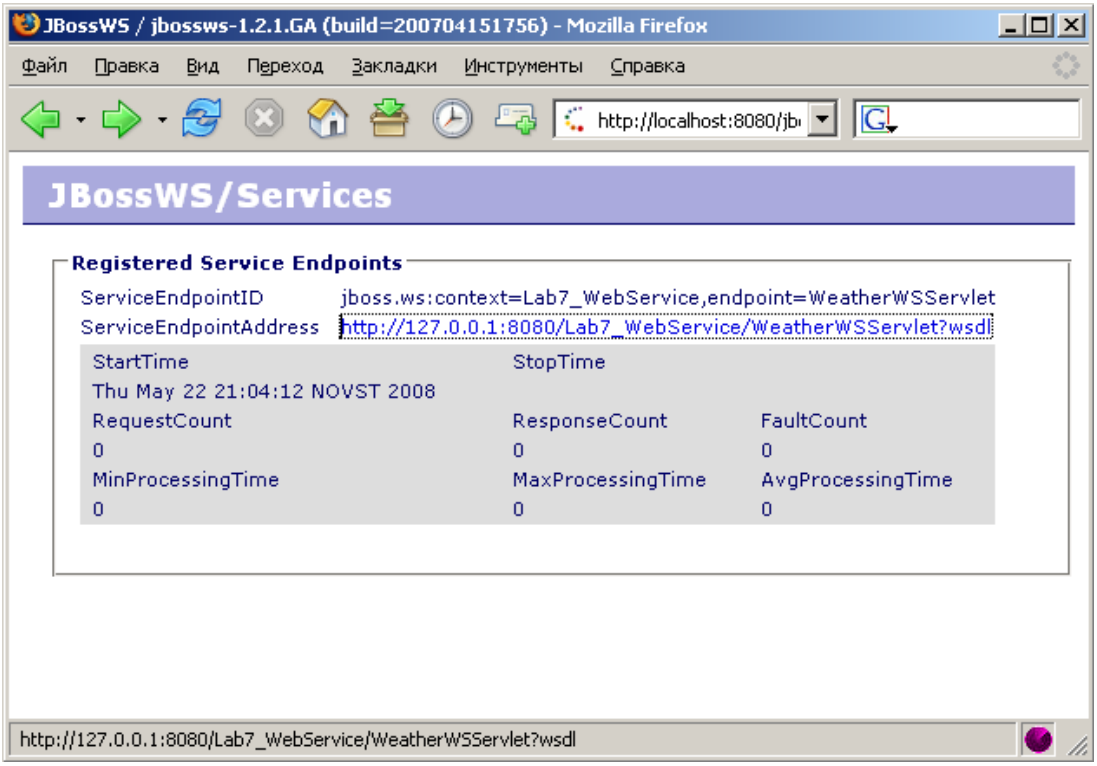
Упаковка и развертывание веб-сервиса

1) Для упаковки веб-сервиса в war-архив веб-приложения, нам понадобиться представление Project Archives. Оно может быть подключено с помощью команды основного меню Window/Show View/Other, где необходимо выбрать JBoss Tools/Project Archives.
2) В окне Package Explorer выделите заголовок проекта Lab7_WebService.
3) В представлении Project Archives щелкните на ссылку WAR.
4) Оставьте значения параметров архивации по умолчанию, нажмите Next и, затем, Finish. В результате созданный архив Lab7_WebService.war отображается в окне Package Explorer.
5) Запустите сервер приложений JBoss AS (должна быть активной перспектива JBoss AS).
6) Щелкните правой кнопкой мыши на значок проекта Lab7_WebService в окне Project Explorer и выберите Build Project. Если компиляция прошла успешно, выполняется создание/обновление архива Lab7_WebService.war.
7) Для развертывания приложения на сервере, щелкните правой кнопкой мыши на архив Lab7_WebService.war в окне Project Explorer и выберите команду Deploy To Server. В случае успешного выполнения в консоли сервера отображается информация о развертывании и запуске веб-сервиса:



Просмотр информации о запущенном веб-сервисе

1) Запустите броузер и перейдите по адресу `http://localhost:8080/jbossws`.
2) Щелкните по ссылке View a list of deployed services.
3) На открывшейся странице отображается список веб-сервисов запущенных на сервере приложений. Для каждого сервиса указана ссылка на сгенерированный автоматически WSDL-файл (ServiceEndpointAddress) и информация о запуске/остановке и вызовах веб-сервиса.



4) Для нашего сервиса WSDL-файл можно получить по адресу `http://127.0.0.1:8080/Lab7_WebService/WeatherWSServlet?wsdl`. Щелкните по ссылке и просмотрите содержимое файла.

Создание нового проекта для веб-клиента и генерация заглушек (stubs) для доступа к веб-сервису

1) Создайте новый проект типа Dynamic Web Project. Назовите проект Lab7_WebClient.

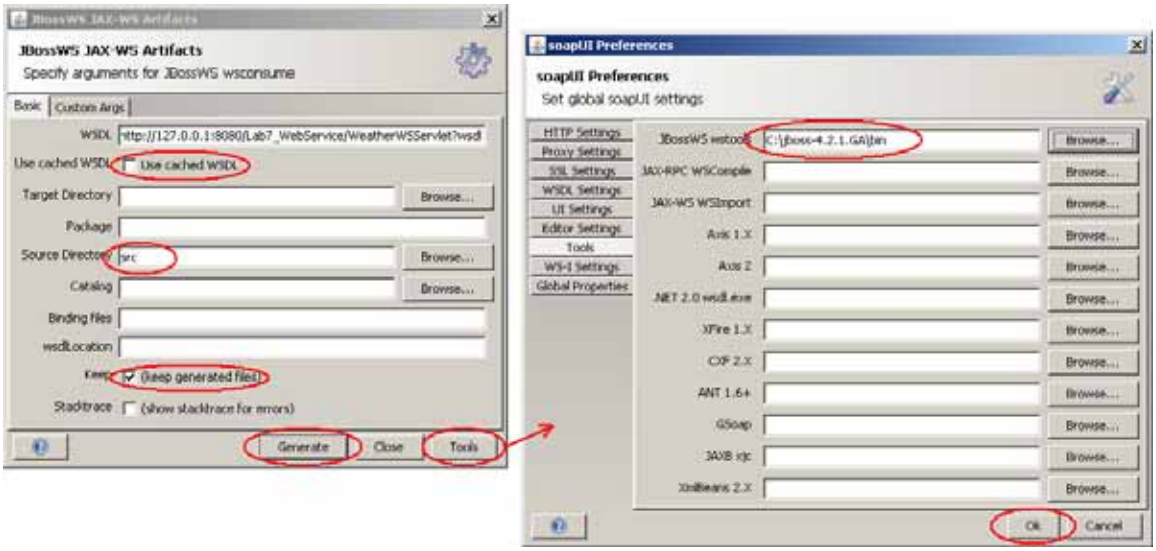
Классы-заглушки, необходимые для обращения к удаленному веб-сервису, могут быть сгенерированы при помощи утилиты wsconsume на основе WSDL-файла, предоставляемого поставщиком веб-сервиса. Мы будем использовать перспективу soapUI, предоставляющую в числе прочих возможностей удобный интерфейс для работы с утилитой wsconsume. Информация о подключении плагина soapui содержится в п. «Установка и настройка программного обеспечения», пп. 9 «Настройка утилиты wsconsume и плагина soapui для Eclipse».

2) С помощью команды меню Window/Open Perspective/Other или при помощи кнопки Open Perspective выберите soapUI и нажмите OK.
3) В представлении soapUI, расположенном по умолчанию в левой части перспективы soapUI, щелкните правой кнопкой мыши на элемент Projects и выберите New WSDL Project.
4) Задайте имя проекта (Project Name) равным WeatherService и включите флажок Creates a file for the project. Нажмите OK и сохраните файл проекта в каталог проекта веб-клиента Lab7_WebClient.

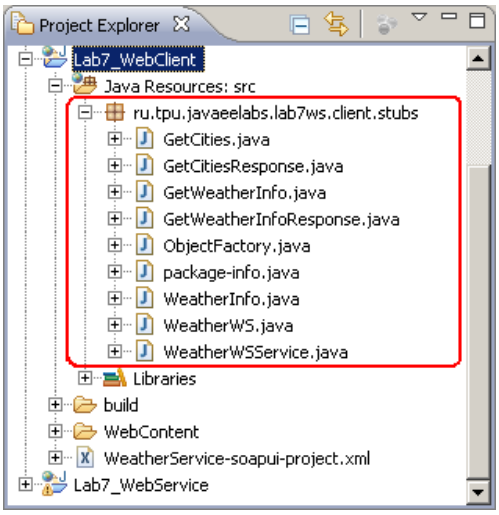
Примечание: созданный soapui-проект WeatherService значительно отличается от тех проектов, которые мы создавали ранее. Этот проект предназначен для просмотра, тестирования и выполнения прочих вспомогательных функций над удаленными веб-сервисами. В нашем случае этот проект будет содержать ссылку на WSDL-файл веб-сервиса, реализованного и запущенного на предыдущем этапе и поможет нам сгенерировать классы-заглушки для веб-клиента.

5) Щелкните правой кнопкой мыши на проект WeatherService и выберите Add WSDL from URL. Скопируйте из броузера ссылку на WSDL-файл `http://127.0.0.1:8080/Lab7_WebService/WeatherWSServlet?wsdl` и нажмите OK.
6) На вопрос, создавать ли запросы по умолчанию для всех операций (Create default requests for all operations), ответьте No.
7) Щелкните правой кнопкой мыши на созданную связку WeatherWSBinding и выберите Generate Code/JBossWS JAX-WS Artifacts.

8) Выключите флажок Use Cached WSDL.
9) В поле Package укажите ru.tpu.javaeelabs.lab7ws.client.stubs.
10) В поле Source Directory с помощью кнопки Browse укажите каталог src проекта веб-клиента (.../Lab7_WebClient/src). В этот каталог будут помещены сгенерированные классы-заглушки.
11) Включите флажок Keep (keep generated files).
12) Нажмите на кнопку Tools и в поле JBossWS Tools укажите каталог bin сервера JBoss AS, в котором размещается командный файл wsconsume.bat. Нажимте OK.



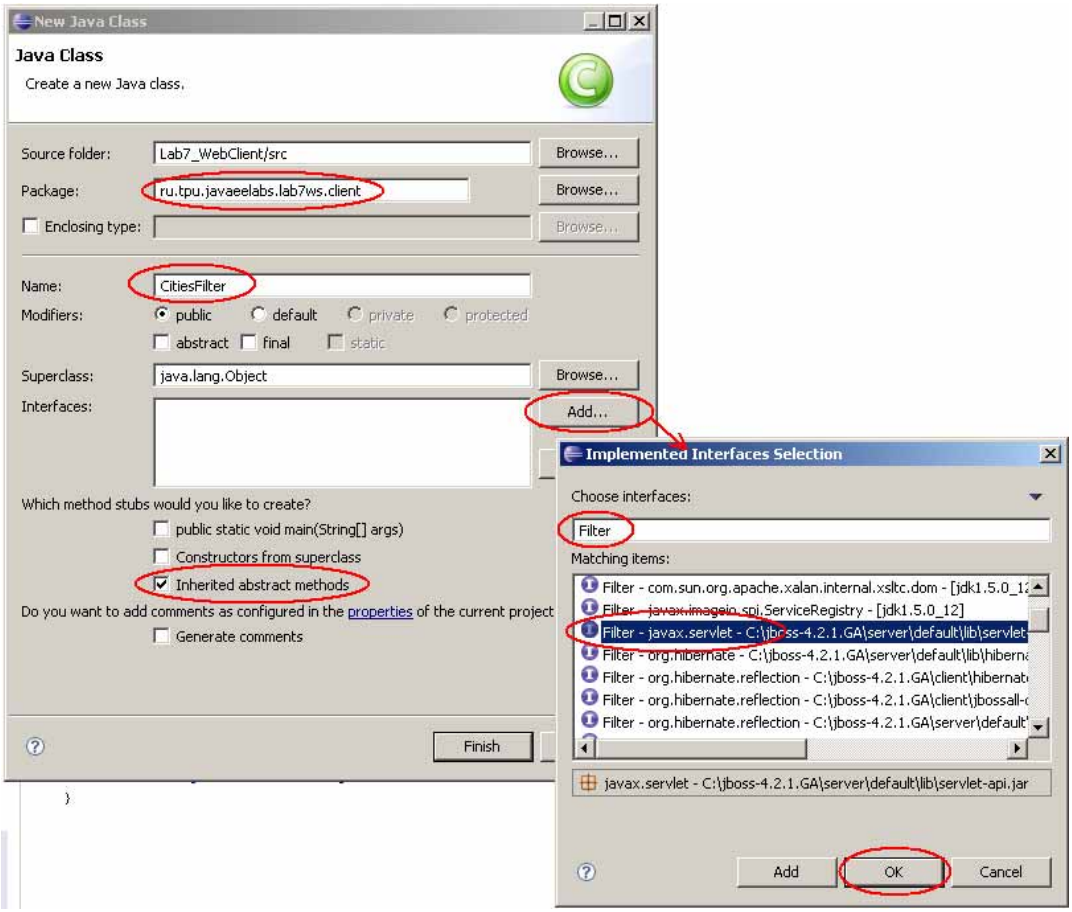
13) Нажмите на кнопку Generate. В случае успешной генерации классов отображается сообщение Execution finished successfully.
14) Закройте окно сообщения и окно настроек генерации классов.
15) Переключитесь на перспективу JBoss AS и выделите проект Lab7_WebClient в окне Project Explorer. Нажмите F5 для обновления состава проекта. Раскройте каталог src и просмотрите сгенерированные классы.



Разработка веб-фильтра `CityFilter` для передачи jsp-странице списка городов

Фильтры используются для перехвата запросов к странице и обработки параметров этого запроса. В нашем случае, в запрос к странице index.jsp будет добавляться список городов, полученный с помощью метода веб-сервиса `getCitites()`.

1) Создайте в пакете `ru.tpu.javaeelabs.lab7ws.client` новый класс CitiesFilter. При создании с помощью кнопки Add списка Interfaces укажите, что класс реализует интерфейс `javax.servlet.Filter`. Оставьте включенным флажок Inherited abstract methods раздела Which method stubs do you like to create?



2) В методе `doFilter()` необходимо установить связь с удаленным веб-сервисом, с помощью метода `getCities()` получить список городов и поместить этот список в параметры объекта-запроса (request) под именем cities.
   Код класса `CitiesFilter` приведен ниже:

```
package ru.tpu.javaeelabs.lab7ws.client;

import java.io.IOException;
import java.util.List;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

import ru.tpu.javaeelabs.lab7ws.client.stubs.WeatherWS;
import ru.tpu.javaeelabs.lab7ws.client.stubs.WeatherWSService;

public class CitiesFilter implements Filter {

      public void destroy() {}
```

```
    public void doFilter(ServletRequest request,
                ServletResponse response,
                FilterChain chain)
                      throws IOException, ServletException {

          // Создаем сервис-фабрику
          WeatherWSService service = new WeatherWSService();
          // Создаем прокси-объект для доступа к веб-сервису
          WeatherWS weatherWS = service.getWeatherWSPort();

          // Получаем список городов
          List cities = weatherWS.getCities();

          // Помещаем список городов в параметры запроса
          request.setAttribute("cities", cities);

          // Направляем запрос далее по цепочке
          chain.doFilter(request, response);
    }

    public void init(FilterConfig arg0) throws ServletException {}
}
```

Далее необходимо объявить созданный фильтр в файле web.xml.

3) Найдите в окне Project Explorer файл web.xml и двойным щелчком мыши откройте конфигуратор. В окне конфигуратора выберите режим Tree.
4) Для того чтобы объявить новый фильтр, выберите категорию Filters и нажмите на кнопку Add справа от списка Filters.
5) Введите имя фильтра (Filter-Name) – CitiesFilter и укажите класс фильтра `ru.tpu.javaeelabs.lab7ws.client.CitiesFilter`.
6) Нажмите Finish. Объявленный фильтр отображается в категории Filters.

Теперь необходимо связать фильтр со страницей index.jsp таким образом, чтобы он срабатывал при выполнении запроса к этой странице. Эти настройки также указываются в файле web.xml.

7) Снова выделите категорию Filters и нажмите кнопку Add справа от списка Filter Mappings.
8) Введите имя фильтра (Filter-Name) – CitiesFilter и укажите относительный путь к странице (URL-Pattern) – /index.jsp.
9) С помощью кнопки Change параметра Dispatchers выберите все четыре действия (FORWARD, INCLUDE, REQUEST и ERROR), при которых будет срабатывать фильтр.

10) Сохраните web.xml нажатием на Ctrl-S.

Создание сервлета WeatherServlet для получения информации о погоде

1) Создайте новый сервлет с именем WeatherServlet в пакете `ru.tpu.javaeelabs.lab7ws.client`.
2) Метод `doGet()` сервлета считывает параметр city, переданный пользователем, обращается к веб-сервису и с помощью метода `getWeatherInfo()` считывает данные о погоде в указанном городе. Результат – объект `WeatherInfo` – помещается в запрос в виде параметра с именем `weatherInfo` и запрос перенаправляется обратно странице index.jsp. Программный код сервлета приведен ниже:

```
package ru.tpu.javaeelabs.lab7ws.client;

import java.io.IOException;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import ru.tpu.javaeelabs.lab7ws.client.stubs.WeatherInfo;
import ru.tpu.javaeelabs.lab7ws.client.stubs.WeatherWS;
import ru.tpu.javaeelabs.lab7ws.client.stubs.WeatherWSService;

public class WeatherServlet extends javax.servlet.http.HttpServlet
                implements javax.servlet.Servlet {

      static final long serialVersionUID = 1L;

      public WeatherServlet() {
            super();
      }

      protected void doGet(HttpServletRequest request,
                  HttpServletResponse response)
                        throws ServletException, IOException {
            response.setContentType("text/html;charset=UTF-8");

            // Получаем из запроса значение параметра city
            String city = request.getParameter("city");

            // Создаем сервис-фабрику
            WeatherWSService service = new WeatherWSService();

            // Создаем прокси-объект для доступа к веб-сервису
            WeatherWS weatherWS = service.getWeatherWSPort();

            // Получаем список городов
            WeatherInfo weatherInfo = weatherWS.getWeatherInfo(city);
```

```
            // Помещаем список городов в параметры запроса
            request.setAttribute("weatherInfo", weatherInfo);

            // Перенаправляем вызов обратно странице index.jsp
            RequestDispatcher dispatcher = getServletContext()
                        .getRequestDispatcher("/index.jsp");
            dispatcher.forward(request, response);
      }
}
```

Разработка JSP-страницы index.jsp

Наш проект будет содержать одну страницу index.jsp, которая будет использоваться для выбора города в выпадающем списке и для отображения информации о погоде по этому городу.

1) Создайте новую JSP-страницу с именем index.jsp.
2) Реализация JSP-страницы включает в себя следующие основные шаги:

   - смена кодировки страницы на UTF-8 для корректного отображения символов кириллицы;

   - смена заголовка страницы на «Информация о погоде»;

   - создать форму, включающую в себя список (`<select>`) с именем city для отображения городов, полученных в параметре `cities`. При выборе элемента списка форма передает значение параметра `city` сервлету `WeatherServlet`;

   - считать параметр `weatherInfo`. Если он не пустой, отобразить на странице данные о погоде.

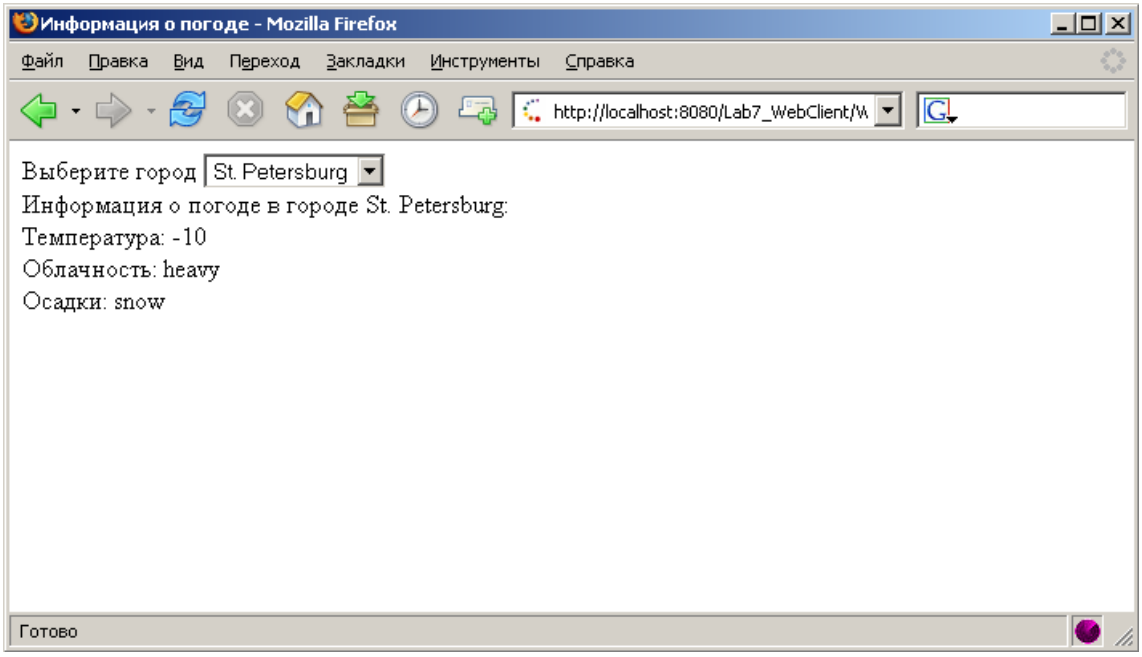
   Ниже приведен код JSP-страницы:

```
< %@ page language="java" contentType="text/html; charset=UTF-8"
      pageEncoding="UTF-8" %>
<!DOCTYPE   html   PUBLIC   "-//W3C//DTD   HTML   4.01   Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

< %@page import="java.util.List" %>
< %@page import="ru.tpu.javaeelabs.lab7ws.client.stubs.WeatherInfo" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Информация о погоде</title>
</head>
<body>
      <form action="WeatherServlet" method = "GET">
      Выберите город
      <select name="city" onChange="submit();">
      <!-- пустой элемент списка -->
      <option>
      < %
      // Считываем данные о погоде
      WeatherInfo weatherInfo =
```

```
            (WeatherInfo)request.getAttribute("weatherInfo");
      //Считываем список городов
      List cities  = (List)request.getAttribute("cities");
      //Отображение списка городов
      if (cities != null) {
            for (int i=0; i<cities.size(); i++) {
                  String sel = "";
                  // Если город уже указан, выбираем его в списке
                  if (weatherInfo!=null &&
                        cities.get(i).equals(weatherInfo.getCity()))
                        sel = " selected = \"selected\"";
                  out.print("<option" + sel + ">" +
                              cities.get(i) + "</option>");
            }
      }
       %>
      </select>
      </form>
      < %
      // Отображение информации о погоде
      if (weatherInfo != null) {
            out.print("Информация о погоде в городе " +
                              weatherInfo.getCity() + ":");
            out.print("<br>");
            out.print("Температура: " + weatherInfo.getTemperature());
            out.print("<br>");
            out.print("Облачность: " + weatherInfo.getCloudiness());
      out.print("<br>");
      out.print("Осадки: " + weatherInfo.getPrecipitations());
      }
       %>
</body>
</html>
```

Упаковка, развертывание и тестирование веб-клиента

1) Создайте WAR-архив приложения Lab7_WebClient с помощью представления Project Archives.
2) Выполните команду Build Project.
3) При необходимости запустите сервер приложений и СУБД Derby.
4) Разверните файл Lab7_WebClient.war на сервере.
5) Запустите броузер и перейдите по адресу `http://localhost:8080/Lab7_WebClient`.
6) Проверьте работу приложения – выберите город из списка и просмотрите данные о погоде.



Варианты заданий

1. Необходимо разработать веб-сервис, предоставляющий информацию о курсах валют в различных банках (например, соотношения RUB, USD и EUR). Информация о курсах валют хранится в БД или XML-файле. Пользователь с помощью веб-интерфейса выбирает два вида валюты и получает информацию о соотношении курса по всем банкам.

2. Необходимо разработать веб-сервис для банка XYZ, предоставляющий возможность перевода указанной суммы из одного вида валюты в другой (например, RUB<–>USD<–>EUR). Информация о курсах валют хранится в БД или в XML-файле. Пользователь с помощью веб-интерфейса указывает сумму, исходную и конечную валюту и получает информацию о сумме, полученной в результате конвертации из исходной валюты в конечную.

ПЕРЕЧЕНЬ РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Дубаков А.А. Проектирование информационных систем: Учебное пособие. – Томск: Изд-во ТПУ, 2001. – 149 с.
2. Фаулер М. UML. Основы. 3-е издание. Краткое руководство по стандартному языку объектного моделирования.: Пер. с англ. – СПб: Символ-Плюс, 2005.
3. Вендров А. М. Проектирование программного обеспечения экономических информационных систем. 2-е изд. – М.: Финансы и статистика, 2005.
4. Вендров А. М., Малышко В. В. Объектно-ориентированный анализ и проектирование с использованием языка UML.: Методическое пособие – М.: Издательский отдел факультета ВМиК МГУ, 2002.
5. Ken Schwaber, Agile Project Management with Scrum. Microsoft Press, 2004.
6. Якобсон А., Буч Г., Рамбо Дж. Унифицированный прцесс разработки программного обеспечения. СПб.: Питер, 2002. – 496 с.: с ил.
7. Bryan Basham, Kathy Sierra, and Bert Bates. Head First Servlets and JSP™, Second Edition, O'Reilly, 2008, ISBN: 0596005407, 883 p.
8. Jim Farely, William Crawford, David Flanagan. Java Enterprise in a Nutshell, 3rd edition, O'Reilly, 2005, ISBN: 0-596-10142-2, 892 p.
9. Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey, Donald F. Ferguson. Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More, Prentice Hall,.2005, ISBN: 0-13-148874-0, 456 p.
10. Dan Woods, Thomas Mattern,"Enterprise SOA: Designing IT for Business Innovation", O`Reilly, 2006, ISBN: 0-596-10238-0, 452 p.
11. David Chappell, Tyler Jewell, Java Web Services, O`Reilly, 2002, 0-596-00269-6, 276 p.
12. Neil Gray, Web Server Programming, John Wiley & Sons Ltd, 2003, ISBN 0-470-85097-3, 621 p.

13. Norbert Bieberstein; Robert G. Laird; Dr. Keith Jones; Tilak Mitra. Executing SOA: A Practical Guide for the Service-Oriented Architect, IBM Press, 2008, ISBN: 0-13-235374-1, 240 p.

14. H. M. Deitel, P. J. Deitel. Java How to Program, Sixth Edition, Prentice Hall, 2004, SBN-10 : 0-13-148398-6

15. Eric Pulier, Hugh Taylor. Understanding Enterprise SOA, Manning Publications Co., 2007, ISBN 1-932394-59-1, 242 p.

16. James McGovern, Sameer Tyagi, Michael Stevens and Sunil Matthew. Java Web Services Architecture, Morgan Kaufmann Publishers, 2003, ISBN:1558609008, 831 p

17. Пол Дж. Перроун, Венката Венката С. Р. "Кришна" Р. Чаганти Создание корпоративных систем на основе Java2 Enterprise Edition. Руководство разработчика. 2001 г. Издательство: "Вильямс" ISBN: 5845901685, 0672317958,

18. Марти Холл, Лэрри Браун. Программирование для Web. Библиотека профессионала, 2001 г. Издательство: "Вильямс"

19. Х. М. Дейтел, П. Дж. Дейтел, С. И. Сантри Технологии программирования на Java 2. Книга 1 Графика, JAVABEANS, Интерфейс пользователя Advanced Java 2 Platform. How to Program Издательство: Бином-Пресс, 2003. – 560 с.: ил.

20. Х. М. Дейтел, П. Дж. Дейтел, С. И. Сантри Технологии программирования на Java 2. Книга 2 Распределенные приложения Advanced Java 2 Platform. How to Program Издательство: Бином-Пресс, 2003. – 464 с.: ил.

21. Х. М. Дейтел, П. Дж. Дейтел, С. И. Сантри Технологии программирования на Java 2. Книга 3. Корпоративные системы, сервлеты, JSP, Web-сервисы Advanced Java 2 Platform. How to Program Издательство: Бином-Пресс, 2003. – 672 с.: ил.

22. Кришнамурти, Дж. Рексфорд Web-протоколы. Теория и практика. HTTP/1.1, взаимодействие протоколов, кэширование, измерение трафика Издательство: Бином, 2002.

23. Даконта М., Саганич А. XML и Java 2. Библиотека программиста, Издательство ПИТЕР, 2001 год, 384 с., ISBN 5-318-00187-4

24. Сью Шпильман JSTL. Практическое руководство для JSP-программистов Издательство: КУДИЦ-Образ, 2004 г.

25. Java портал Sun Microsystems – http://java.sun.com.

Оглавление

Учебное издание

ДУБАКОВ Анатолий Алексеевич
ПИНЖИН Алексей Евгеньевич

ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ И ТЕЛЕКОММУНИКАЦИОННЫХ СИСТЕМ

Учебное пособие

Научный редактор
доктор технических наук,
профессор — *Ю.П. Ехлаков*

Верстка — *О.Ю. Аршинова*

Дизайн обложки — *О.Ю. Аршинова*
О.А. Дмитриев

Подписано к печати 12.03.2009. Формат 60x84/16. Бумага «Снегурочка».
Печать XEROX. Усл. печ. л. 24,66. Уч.-изд. л. 22,30.
Заказ 191-09. Тираж 200 экз.

Томский политехнический университет
Система менеджмента качества
Томского политехнического университета сертифицирована
NATIONAL QUALITY ASSURANCE по стандарту ISO 9001:2000

ИЗДАТЕЛЬСТВО ТПУ. 634050, г. Томск, пр. Ленина, 30.