МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Государственное образовательное учреждение высшего профессионального образования
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

А. Ф. Тузовский

ПРОЕКТИРОВАНИЕ ИНТЕРНЕТ ПРИЛОЖЕНИЙ

*Рекомендовано в качестве учебно-методического пособия
Редакционно-издательским советом
Томского политехнического университета*

Издательство
Томского политехнического университета
2011

УДК 681.3.06
ББК 32.973.23-018.2
Т81

Тузовский А.Ф.

Т81 Проектирование Интернет приложений / А.Ф. Тузовский; Томский политехнический университет. – Томск: Изд-во Томского политехнического университета, 2010. – 200с.

В пособии рассматриваются современные методы проектирования и разработки web-приложений. Поясняются понятия и основные стандарты сети Интернет и реализованной на ее основе web-сети (World Wide Web), состав программного обеспечения данной сети, логика работы web-приложений, основные подходы и технологии их разработки. В качестве примера подхода к разработке web-приложений рассматривается технология ASP.Net Web Forms. В заключительной главе пособия рассмотрены современные методологии проектирования web-приложений и рекомендации по реализации различных их уровней.

Пособие подготовлено на кафедре «Оптимизации систем управления» института кибернетики ТПУ и предназначено для студентов, обучающихся по специальности 080801 «Прикладная информатика (в экономике)» изучающих дисциплину «Проектирование Интернет приложений» и ранее изучавших дисциплину «Высокоуровневые методы информатики и программирования».

УДК 681.3.06
ББК 32.973.23-018.2

Рецензенты

Доктор технических наук, профессор,
зав. кафедрой автоматизированных систем управления ТУСУРа
А.М. Кориков

Доктор физико-математических наук, профессор,
зав. кафедрой теоретической кибернетики ТГУ
Ю.Г. Дмитриев

Оглавление

1. Интернет и World Wide Web

Первоначально Web сеть (World Wide Web, WWW, Web) определялась, как информационная система коллективной работы пользователей с гипертекстовыми документами расположенная в сети Интернет. В настоящее время она уже является платформой, на основе которой могут разрабатываться различные приложения и целые информационные системы. Взаимосвязь web-приложений с технологиями сети Интернет показана на рис. 1.1.

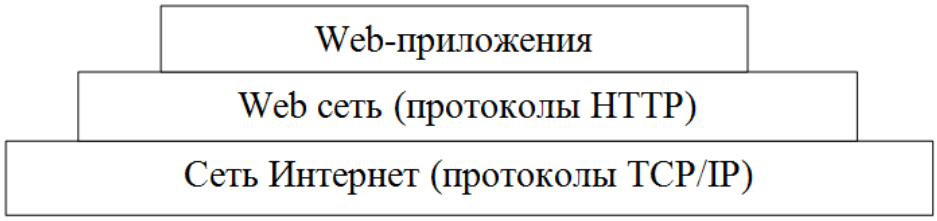


Рис. 1.1. Взаимосвязь web-приложений с другими технологиями

В связи с этим, прежде чем рассматривать разработку web-приложений следует рассмотреть основные понятия Интернет сети и Web сети.

1.1. Сеть Интернет

Интернет (Internet) это глобальная компьютерная сеть, позволяющая передавать данные между компьютерами расположенными по всему миру. Она появилась в 1969 году, когда впервые были связаны между собой большие компьютеры (mainframe) четырех университетов США.

Интернет включает инфраструктуру (скоростные каналы передачи данных, специальные компьютеры, выполняющие управление передачей пакетов данных (маршрутизаторы) и программное обеспечение) и большое количество подключенных к ней компьютеров (с серверным и клиентским программным обеспечением). Всем подключенным к сети Интернет компьютерам задаются уникальные цифровые адреса, называемые адресами Интернет протокола (IP адресами), которые являются целыми 4-х байтовыми (беззнаковыми) числами. Для удобства они записываются в виде последовательности значений каждого байта, разделенных символом точка ‘.’. Например: 109.123.152.2.

Все программное обеспечение сети Интернет работает по технологии «клиент-сервер», т.е. все используемые программы делятся на два типа:

- Серверы – пассивные программы, которые ожидают запросы от клиентов, обрабатывают их как можно быстрее, отправляют запрашиваемую информацию и ожидают следующих запросов.
- Клиенты – активные программы, с которыми обычно работает пользователь сети на своих компьютерах, отправляют запросы серверам для выполнения некоторой работы (обычно получение некоторой информации).

Для взаимодействия клиентов и серверов, они должны соблюдать одинаковые правила описания запросов и ответов на них. Такие наборы правил называются ***протоколами передачи данных***. Базовый уровень сети Интернет работает с использованием двух основных протоколов Internet Protocol (IP) и Transmission Control Protocol (TCP), на которые часто ссылаются совместно, как TCP/IP. Они позволяют передавать данные в виде специально оформленных пакетов ограниченного размера. Работа с протоколом IP и TCP обычно поддерживается на уровне операционной системы.

Интернет сокеты

Реализация передачи данных между компьютерами обычно выполняется с использованием ***Интернет сокетов*** – специальным программных объектов, которые позволяют организовать передачу данных между выполняемыми процессами, с использованием протокола TCP/IP.

Сокеты это некоторые абстракции (объекты), с помощью которых приложение может посылать и получать данные, во многом аналогично тому, как с помощью указателя на открытый файл приложение может читать и писать данные на внешнее устройство хранения. Сокет позволяет приложению подключиться к сети и выполнять взаимодействие (обмен данными) с другими приложениями, которые с помощью своих сокетов подключаются к той же самой сети. Данные переданные сокету на одном компьютере, могут читаться другим приложением, использующим сокеты, на другом компьютере.

Сокеты обычно реализуются с помощью API библиотек, как например библиотеки «Berkeley sockets», первоначально созданной в 1983. Большинство реализаций библиотек сокетов создано на основе данной библиотеки, например, библиотека Winsock, разработанная в 1991 году. Разработка прикладных программ, использующих такие API библиотеки, называется сетевым программированием.

Программные процессы, которые предоставляют прикладные сервисы, называются ***серверами***, они в начале работы создают сокеты, находящиеся в состоянии прослушивания портов. Такие сокеты ожидают действия от ***клиентских*** программ. Каждый сокет имеет свой адрес, который является комбинацией IP адреса компьютера и номера порта, который связан с процессом прикладной программы. Порт это некоторый номер (от 0 до 65535), указывающий, какому приложению предназначаются поступающие по сети пакеты данных, имеющие такой адрес порта. Операционная система компьютера, получившего пакет, передает его приложению, выполняющему работу с данным портом (рис. 1.2).

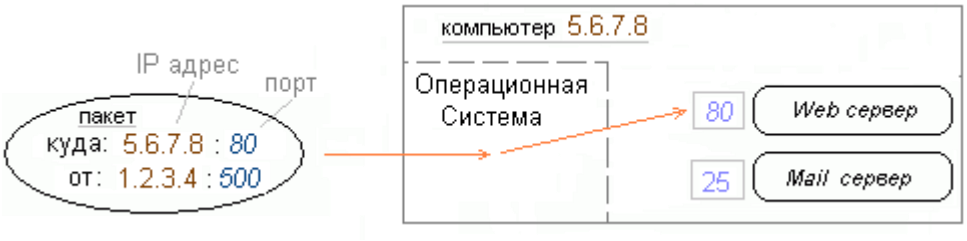


Рис. 1.2. Связь IP адреса и номера порта с приложением

Использование портов позволяет независимо использовать на одном и том же компьютере TCP протокол сразу многими приложениями. Сервер при запуске сообщает операционной системе, номер порта (или несколько портов), который он хотел бы «занять». После этого все пакеты, приходящие на компьютер к этому порту, ОС будет передавать данному серверу. Говорится, что сервер «прослушивает» указанный порт.

С помощью сокетов, работающих в режиме прослушивания, разрабатываются приложения-серверы. А с помощью сокетов, работающих в режиме клиентов – приложения-клиенты. Приложения с помощью сокеты могут обмениваться между собой сообщениями разных форматов.

Сеть Интернет является основой для реализации большого количества различных информационных систем, предоставляющих пользователям различные информационные услуги.

Система доменных имен

Примером одной из информационных систем Интернет является «Система доменных имен» (Domain Name System, DNS). Данная система является частью инфраструктуры сети, которая позволяет хранить и быстро находить специальные записи, связывающие IP адреса с символьными именами (доменными именами).

Доменное имя, это символьное имя, служащее для обозначения иерархической структуры подобластей сети Интернет. Каждая из таких подобластей называется доменом. Доменные имена могут соответство-

вать компьютерам (серверам, предоставляющим услуги в сети, хосты), ***web-сайтам***, почтовым (e-mail) серверам. Доменные имена компьютеров и web-сайтов, хранимые в DNS используются в URL адресах ресурсов web-сети. Полное доменное имя состоит из имени ближайщего (самого низкого уровня) домена и далее имён всех доменов более высокого уровня, в которые он входит, разделённых точками. Например, полное имя ru.wikipedia.org обозначает домен третьего уровня ru, который входит в домен второго уровня wikipedia, который входит в домен верхнего уровня org, который входит в безымянный корневой домен. Обычно под доменным именем понимают полное доменное имя.

Система DNS поддерживается с помощью иерархически организованных DNS-серверов, взаимодействующих по определённому протоколу, которые предоставляют доступ к иерархически распределенной базе данных. Для повышения устойчивости системы используется множество серверов, содержащих идентичную информацию, а в протоколе работы данной системы имеются средства, позволяющие выполнять синхронизацию информации, расположенной на разных серверах. Существует 13 корневых серверов, их адреса практически не меняются.

Доменное имя и IP-адрес не являются тождественными – один IP-адрес может иметь много доменных имён, что позволяет поддерживать на одном компьютере набор web-сайтов (так называемый виртуальный хостинг). Справедливо также и обратное – одному доменному имени может быть сопоставлено множество IP-адресов, что позволяет поддерживать работу одного web-сайта несколькими серверами (выполнять балансировку нагрузки на web-сайт).

Интернет сервисы

Интернет сервисы, это информационные системы, разработанные для работы в сети Интернет для предоставления некоторых информационных услуг. Они включают наборы программ и протоколов прикладного уровня, обеспечивающих пользователей сети возможностью выполнять работу с распределенными информационными ресурсами. Примерами таких сервисов являются следующие:

- система обмен сообщениями (первоначально, включающими, только текст, а затем и документы в других форматах) между пользователями – электронная почта (протоколы SMTP, POP3);
- система передача файлов между компьютерами сети (протокол FTP);
- система проведение коллективных обсуждений по разным темам (протокол NNTP).

Каждый сервис разрабатывается с использованием специального протокола, который используют соответствующие клиенты и серверы. Примерами таких протоколов являются следующие:

- FTP (File Transfer Protocol) – протокол для просмотра содержания каталогов и передачи файлов между компьютерами;
- Telnet (TELecommunication NETwork) – протокол для поддержки функциональности Network Virtual Terminal, позволяющей подключиться к удаленному компьютеру по сети Интернет и удаленно работать с его операционной системой;
- SSH (Secure Shell) – протокол, позволяющий производить удаленное управление операционной системой и передачу файлов. В отличие от Telnet шифрует весь трафик;
- Протоколы работы с электронной почтой E-mail: SMTP (Simple Mail Transfer Protocol) для отправки электронных почтовых сообщений; POP (Post Office Protocol) –для получения сообщений электронной почты, хранящихся на почтовом сервере (последняя версия POP3); IMAP (Internet Message Access Protocol) – более совершенный протокол доступа к электронной почте в Интернет. Последняя версия IMAP4.
- Протоколы работы в форумах: NNTP (Network News Transfer Protocol) – для отправки чтения сообщений из новостных групп, поддерживаемых NNTP сервером;
- Протоколы ведения разговоров (чатов) и передачи сообщений: XMPP (Jabber) – основанный на XML расширяемый протокол для мгновенного обмена сообщениями в почти реальном времени; talk – для обмена сообщениями;
- LDAP – протокол для доступа к службе каталогов X.500, является широко используемым стандартом доступа к службам каталогов;
- SNMP – базовый протокол управления Интернет сетью.
- Gopher – протокол поиска и передачи документов.

Наиболее известным сервисом сети Интернет является сервис предоставления доступа к гипертекстовой информации, который называется web-сетью (World Wide Web, WWW), либо Всемирной паутиной. В связи с популярностью данного сервиса часто под Интернет сетью (физической сетью) понимают web-сеть (виртуальную сеть).

1.2. Web сеть

В 1989 году исследователь, работающий в Европейской организации ядерных исследований CERN, Tim Berners-Lee [6] разработал пер-

вую гипертекстовую систему коллективной работы с документами, которая позволяла выполнять переходы между ними. Он также разработал основные стандарты, на которых основывается данная система.

Web-сеть не является физической сетью. Это сервис предоставляющий пользователям доступ к связанному с помощью ссылок набору гипертекстовых документов и web-приложений. Web сеть состоит из набора программного обеспечения (клиентов и серверов) и огромного количества информационных ресурсов, которые работают совместно ***в соответствии с набором заранее согласованных стандартов***. Основными стандартами Web сети являются:

1. стандарт задания адресов ресурсов сети – URL;
2. протокол взаимодействия между клиентами и серверами – HTTP;
3. язык описания информационных ресурсов (гипертекста) – HTML;
4. язык форматирования информационных ресурсов – CSS;
5. язык выполнения программ на стороне клиента – JavaScript.

Web-сеть может рассматриваться с разных точек зрения (способа представления Web сети), как:

- набор стандартов, в соответствии с которыми web-сеть работает (основными являются: URL, HTML, HTTP);
- набор ресурсов, составляющие web-сеть (основным документами являются web-страницы (или HTML страницы, или гипертекстовые страницы), связанные между собой гиперссылками);
- набор программных средств позволяющих работать такой информационной системе. Основными программными средствами являются: web-серверы (HTTP серверы) и браузеры.

Web-сайты

Основными структурными элементами web-сети являются ***web-сайты*** (website, site – «место») или просто сайты. Web-сайт это объединённый под одним адресом (доменным именем или IP-адресом) набор логически связанных ресурсов. Все ресурсы web-сайтов можно разделить на следующие два типа:

- статические ресурсы – HTML документы; изображения; мультимедиа файлы; любые файлы данных, к которым есть доступ;
- динамические ресурсы – web-приложения: программные модули (exe, dll); шаблоны web-страниц; скрипты; программные объекты и т.п., которые, как правило, по запросу формируют HTML документы.

Web-сайт имеет некоторую начальную страницу (home page), используя которую можно перейти к другим страницам сайта. Страницы

web-сайта обычно разделены по папкам, которые создают иерархическую структуру сайта. Если при обращении к web-сайту начальная страница в URL адресе не указывается, то web-сервер использует страницу со стандартным именем (обычно это страницы с именами default или index).

Web-сайты идентифицируются в глобальной сети под доменными именами, зарегистрированными в системе DNS. В большинстве случаев одному web-сайту соответствует одно доменное имя. Однако, на одном компьютере (под управлением одного web-сервера) могут находиться несколько web-сайтов.

Адресация ресурсов web-сети

Для адресации всех ресурсов, которые доступны пользователям web-сети, используются единообразные указатели ресурсов (Uniform Resource Locator, URL) – URL адреса, которые помимо идентификации ресурсов, предоставляет еще и информацию об их местонахождении.

URL адреса ресурсов имеют следующую структуру:

<схема>://хост[:порт] /путь/. . . /имя-ресурса [?строка-запроса] [#ссылка]

где:

- схема – схема обращения к ресурсу; обычно это название используемого протокола (например, ftp или http);
- хост – доменное имя web-сайта в системе DNS или IP-адрес компьютера (хоста), на котором расположен web- сайт;
- порт – порт хоста для подключения (номер, используемый для разделения сообщений по обрабатывающим серверам);
- /путь/. . . / (URL-путь) – уточняющая информация о месте нахождения ресурса в файловой системе сервера. Это может быть или путь к конкретному файлу или к каталогу (если заканчивается символом '/'). Если задан каталог, то сервер либо вернет список, содержащихся в нем файлов, либо файл, имя которого задано в конфигурации сервера по умолчанию (обычно, index.html или default.html);
- имя-ресурса – собственное имя ресурса в файловой системе сервера, или условное имя, по которому web-сервер будет определять требуемый физический ресурс (в процессе разрешения адреса);
- строка-запроса – набор пар «параметр=значение», разделенных символом ‘&’, обычно передаются в результате обработки формы web-страницы;
- ссылка (anchor) – указатель на некоторый раздел web-страницы.

1.3. Протокол HTTP

Взаимодействие всех участников web-сети основывается на использовании протокола передачи гипертекстов – HTTP (HyperText Transfer Protocol). HTTP является протоколом прикладного уровня, в стеке TCP/IP протоколов, использующим TCP в качестве базового протокола транспортного уровня для передачи сообщений. Текущей версией данного протокола является HTTP/1.1. По умолчанию данный протокол использует порт 80.

С помощью протокола HTTP web-серверы и браузеры обмениваются информацией, поэтому web-серверы также часто называются HTTP серверами, а web-браузеры – HTTP клиентами. Однако HTTP клиентами могут быть не только браузеры, но и любые программы, которые могут использовать HTTP протокол (прокси серверы, поисковые агенты и т.п.). Пример взаимодействия web-браузера и web-сервера с использованием протокола HTTP показан на рис. 1.3.

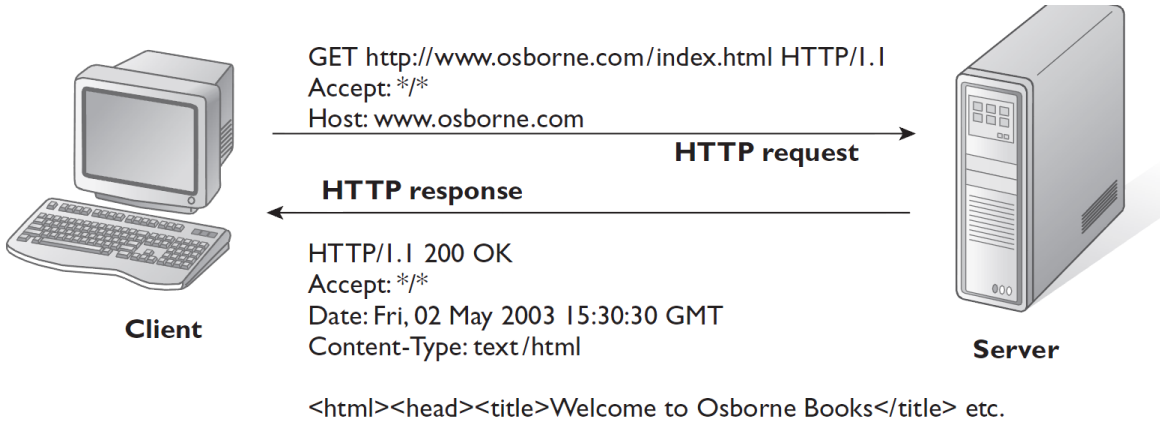


Рис. 1.3. Взаимодействие браузера и сервера по протоколу HTTP

Протокол HTTP использует принцип «запрос-ответ», означающий, что программа HTTP клиент посылает HTTP серверу сообщение (команду) вида «HTTP запрос» (request), а сервер возвращает сообщение вида «HTTP ответ» (response). Структура сообщений запросов и ответов сходна с сообщениями электронной почты (e-mail), в том, что они состоят из группы строк, содержащих заголовки сообщения (headers), после которых следует пустая строка (символы ‘\r\n’, коды 13 и 10), а затем следует тело сообщения.

HTTP является протоколом ***без поддержки состояния*** (***stateless***), который не полагается на удержание соединения между сокетами в логике обмена сообщениями. Единичная HTTP транзакция состоит из одного запроса от клиента серверу, за которым следует ответ сервера данному клиенту.

Если протокол поддерживает состояние (stateful), как протоколы FTP, SMTP и POP, то последовательность связанных команд (сообщений запросов) обрабатывается, как одно взаимодействие. В этом случае, сервер должен поддерживать “состояние” данного взаимодействия с клиентом в течение всего времени передачи сообщений (сеанса связи), до прекращения взаимодействия (завершения сеанса связи). Последовательность передаваемых и обрабатываемых сообщений называется ***сеансом*** (session) работы клиента.

В отличие от таких протоколов, каждый обмен HTTP сообщениями состоит из одного запроса и одного ответа. Таким образом, от HTTP клиентов и серверов не требуется поддерживать состояние между передаваемыми сообщениями. Однако, для работы web-приложений требуется поддерживать состояния сеансов.

Рассмотрим пример загрузки web-страницы, хранящейся на web-сервере. Пользователь может вручную напечатать URL адрес в браузере, или щелкнуть кнопкой мыши по гиперссылке в текущей странице, показываемой браузером, или выбрать закладку на ранее полученную страницу. Во всех этих случаях, запрос на посещение конкретного URL будет преобразован браузером в HTTP запрос, который имеет следующую структуру:

```
МЕТОД   /путь-к-ресурсу  HTTP/номер-версии
Имя-заголовка-1: значение
Имя-заголовка-2: значение

[тело запроса, которое может отсутствовать]
```

Первая строка запроса состоит из трех полей:

- первое поле МЕТОД соответствует одному из нескольких поддерживаемых методам запроса, главными из которых являются GET и POST;
- второе поле /путь-к-ресурсу соответствует части URL адреса, которая задает на web-сервере путь к запрашиваемому ресурсу;
- третье поле номер-версии соответствует версии протокола HTTP, который использует клиент (1.0 или 1.1).

После первой строки записывается список HTTP заголовков (headers), за которыми следует пустая строка, часто называемая <CR><LF> (символы “возврат каретки и завершение строки”). Данная пустая строка отделяет заголовки запроса от тела запроса. После пустой строки может быть (хотя и не обязательно) тело запроса, после которого следует другая пустая строка, указывающая на конец запроса.

Если, например, запрашиваемым URL адресом является http://www.mywebsite.ru/~ivanov/defaut.html, то далее показана упрощенная версия HTTP запроса, который будет передан web серверу с именем www.mywebsite.com:

```
GET  /~ivanov/defaut.html  HTTP/1.1
Host: www.mywebsite.ru
```

Отметим, что HTTP запрос заканчивается пустой строкой. При использовании GET запроса, в сообщении нет тела, поэтому запрос просто завершается пустой строкой. Также отметим, наличие заголовка Host, который указывает доменное имя используемого web-сайта.

Сервер, при получении такого запроса будет формировать HTTP ответ, который имеют следующую структуру:

```
HTTP/номер-версии  код-состояния  текстовое-пояснение
Имя-заголовка-1: значение
Имя-заголовка-2: значение

[тело ответа]
```

Первая строка (***строка состояния***) содержит версию HTTP протокола, за которой следует трех цифровой код состояния и краткое текстовое пояснение данного кода состояния. Ниже приведена упрощенная версия HTTP ответа, который может сформировать и отправить сервер, если запрашиваемый файл существует и тот, кто его запрашивает, имеет право доступа к нему:

```
HTTP/1.1  200  OK
Content-Type: text/html
Content-Length: 9934

<HTML>
   <HEAD>
      <TITLE>Домашняя страница</TITLE>
   </HEAD>
   <BODY>
      <H2 align="center">Добро пожаловать на сайт Иванова И.П.</H2>
      . . .
   </BODY>
</HTML>
```

Так как переданный в данном примере HTTP запрос был успешно обработан, то строка состояния содержит код 200 и краткое его пояснение (OK). Отметим, что в HTTP ответ включены строки заголовков, по-

сле которых идет пустая строка, за которой следует содержание запрашиваемого файла (тело ответа).

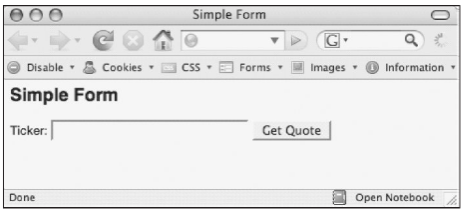
Процесс передачи запросов и ответов между браузерами и серверами не всегда является таким простым. Например, HTML страница может содержать ссылки на другие доступные ресурсы, которые связаны с ней (изображения, скрипты, стили и т.п.). Клиенты, которые выполняют показ изображений, выполнение скриптов, используют стили для представления страниц, должны выполнить разбор полученной HTML страницы, чтобы определить, какие дополнительные ресурсы требуются для правильного отображения данной страницы, а затем формируют новые HTTP запросы на получение этих дополнительных ресурсов.

Методы запроса

В HTTP протоколе определены следующие основные методы запроса GET, HEAD и POST. Кроме них имеются и другие, менее часто используемые, методы, как например PUT, DELETE, TRACE, OPTIONS и CONNECT. HTTP серверы обязаны поддерживать только методы GET и HEAD. Все другие методы являются не обязательными, хотя большинство серверов их поддерживают.

Метод GET

Метод GET является самым простым методом запроса. При вводе URL адреса в браузере, переходе по гиперссылке или выборе закладки на ранее посещаемые страницы, браузер использует метод GET при формировании запроса к web серверу.



(a)

```
<HTML>
<HEAD><TITLE>Simple Form</TITLE></HEAD>
<BODY>
<H2>Simple Form</H2>
<FORM action="http://finance.yahoo.com/q" method="get">
Ticker: <INPUT size="25" name="s">
<INPUT type="submit" value="Get Quote">
</FORM>
<BODY>
</HTML>
```

(b)

Рис. 1.4. Отображение HTML формы в браузере (a) и ее описание в странице (b)

При заполнении в web-странице HTML формы, у которой задан атрибут «metod="GET"» (рис. 1.4), браузер, поддерживающий протокол HTTP/1.1, будет формировать URL адрес, включающий строку запроса (query string), содержащую названия всех полей ввода формы и заданных пользователем значений:

```
GET   /q?s=YHOO  HTTP/1.1
Host: finance.yahoo.com
User-Agent: Mozilla/5.0 (Windows; U; Windows XP; en-US; rv:1.8.0.11)
```

В данном случае в форме есть поле s, в котором было задано значение YHOO (?s=YHOO). При успешной обработке этого запроса HTTP ответ от сервера может выглядеть следующим образом:

```
HTTP/1.1  200  OK
Date: Tue, 08 Apr 2010 15:16:46 GMT
Connection: close
Content-Type: text/html

<HTML>
<HEAD>
<TITLE>YHOO: Summary for YAHOO INC – Yahoo! Finance</TITLE>
</HEAD>
<BODY>
…
</BODY>
</HTML>
```

Метод POST

Основное отличие между методами GET и POST состоит в том, что POST запросы имеют тело: некоторое содержание, которое следует за блоком заголовков, и отделенное от заголовков пустой строкой.

Если в форме на рис. 1.4 изменить используемый метод на POST, т.е. «metod="POST"», то браузер сформирует следующее HTTP сообщение, в котором строка запроса (данные формы) будут помещены в тело запроса:

```
POST   /q   HTTP/1.1
Host: finance.yahoo.com
User-Agent: Mozilla/4.75 [en] (WinNT; U)
Content-Type: application/x-www-form-urlencoded
Content-Length: 6

s=YHOO
```

Ответ от сервера будет таким же, что и на предыдущий запрос, использующий метод GET.

Метод HEAD

Запросы, которые используют метод HEAD обрабатываются также, как и запросы, использующие метод GET, за исключением того, что

сервер отправляет в ответах только заголовки сообщения, без тела, содержащего запрашиваемый ресурс. В некоторых случаях клиенту это достаточно, чтобы принять решение о выполнении дальнейшей работы. На основании содержания заголовков клиент может определить, стоит ли запрашивать полное содержание документа.

Исторически, запросы с методом HEAD наиболее часто использовались для поддержки кэширования – временного хранения полученных браузером ресурсов (страниц, изображений и т.п.). Если копия требуемого ресурса была ранее сохранена в кэше (временном хранилище) браузера, то браузер может использовать данную копию, а не просить сервер переслать исходный ресурс. Это будет выполняться, если дата последнего изменения данного ресурса была ранее даты сохранения копии ресурса в кэше, что можно определить путем использования данных заголовка HTTP ответа. Если после сохранения копии ресурса в кэше он был изменен, то браузер должен запросить «свежую» копию содержания данного ресурса.

Предположим, что требуется снова посмотреть в браузере страницу, которую была посещена ранее (например, домашняя страница Иванова И.П.). Так как данная страница ранее загружалась браузером, то ее копия хранится в кэше. Браузер может определить, нужно ли получить эту страницу заново отправив вначале запрос с методом HEAD:

```
HEAD  http://www.mysite.ru/~ivanov/  HTTP/1.1
Host: www.mysite.ru
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.0.11)
```

Ответ на данный запрос будет содержать только набор заголовков, включающий и дату последнего изменения содержания запрашиваемой страницы:

```
HTTP/1.1  200  OK
Date: Tue, 08 Apr 2008 15:55:04 GMT
Server: Apache/2.2.4 (Unix)
Last-Modified: Tue, 29 Oct 2002 04:22:52 GMT
Content-Length: 2111
Content-Type: text/html
```

Из присланного сообщения видно, что это очень старая страница. Браузер (или некоторый другой HTTP клиент) может сравнить дату изменения страницы (заголовок Last-Modified) с датой создания ее копии в кэше и использовать либо имеющуюся копию (если ресурс не изменился) или отправить такой же запрос, но с методом GET (если ресурс был изменен).

Однако в настоящее время в HTTP/1.1 появилась более эффективная возможность поддержки кэширования.

Коды состояния

Первой строкой ответа является строка состояния (status line), которая включает имя протокола и его версии, после которых записывается трех цифровой код состояния (результата обработки) и его краткое текстовое пояснение. Код состояния сообщает HTTP клиенту (браузеру или прокси-серверу) был ли запрос успешно обработан или требуется выполнить некоторые дополнительные действия, которые могут быть уточнены с помощью значений включенных в сообщение заголовков. Пояснительная часть первой строки предназначена для использования человеком: ее изменение или отсутствие не вызовет изменения в действиях правильно разработанного HTTP клиента.

В протоколе HTTP/1.1 определены пять категорий сообщений ответов, на основе первой цифры кода состояния:

- сообщения с кодом состояния, который начинается с цифры 1, классифицируются, как “информационные”;
- сообщения с кодом состояния, который начинается с цифры 2, классифицируются, как “успешные”;
- сообщения с кодом состояния, который начинается с цифры 3, говорят клиенту о необходимости выполнить дополнительные действия (например, выполнить запрос по другому адресу);
- сообщения с кодом состояния, который начинается с цифры 4, говорят о наличии ошибок в запросах клиентов или особых условиях;
- сообщения с кодом состояния, который начинается с цифры 5, являются сообщениями об ошибках сервера.

Информационные коды состояний (1xx)

Эти коды состояний используются только в информационных целях. Они не сообщают об успешном или не успешном завершении обработки запроса сервером, а содержат информацию об их последующих обработках. Серверы используют код 100 для оповещения клиентов о том, что они могут продолжать с работу с частично переданным запросом (например, предоставить тело сообщения после начального предоставления заголовков для метода POST). Клиенты могут сообщить о своем намерении частично передать запрос заголовок «Expect: 100-continue». Сервер может проанализировать такие запросы и послать соответствующий ответ. Если данный сервер может обработать данный запрос, то он отправляет сообщение с кодом состояния 100: «HTTP/1.1 100

Continue». А если сервер не может обработать такой запрос, то он отправляет ответ с кодом состояния, указывающим ошибку в запросе клиента (код состояния “4”).

Коды состояния успешных ответов (2xx)

Наиболее часто используемым успешными кодами состояния является:

- **200 OK** – обработка запроса была успешно выполнена и запрашиваемый ресурс был отправлен клиенту;
- **201 Created** – обработка запроса была успешно выполнена и в результате этого на сервере был создан новый ресурс (для запроса с методом PUT);
- **204 No Content** – обработка запроса была успешно выполнена, но никакого содержания передавать не требуется.

Коды состояния для ответов перенаправления (3xx)

Коды состояния, начинающиеся с цифры “3” указывают на то, что клиенту нужно выполнить дополнительные действия для выполнения исходного запроса. Обычно это предполагает повторение запроса подругому URL. Коды 301 и 307 сообщают HTTP клиента послать исходный запрос на адрес, заданный в заголовке Location ответного сообщения.

- **301 Moved Permanently** – запрашиваемый ресурс был перемещен постоянно на новое место;
- **302 Found** – аналогично коду 303;
- **303 See Other** – запрошенный ресурс может быть найден с помощью другого URI, используя метод GET;
- **304 Not Modified** – ресурс не включен в ответ, так как не был изменен после даты заданной в заголовке If-Modified-Since.
- **307 Temporary Redirect –** запрашиваемый ресурс был перемещен временно на новый адрес.

В любом случае клиент при получении ответа с кодами 301 и 302 должен сформировать и передать новый запрос «переправленный» по новому адресу.

Коды состояния об ошибках в запросах клиента (4xx)

Коды состояния, начинающиеся с цифры “4”, указывают на то, что запрос клиента не может быть выполнен по каким-то причинам:

- **400 Bad Request** – запрос не правильно сформирован;

- **401 Not Authorized** – проблема с правом доступа к ресурсу, клиент должен передать заголовок WWW-Authenticate с данными аутентификации пользователя;
- **403 Forbidden** – сервер отказывается выполнять запрос;
- **404 Not Found** – сервер не может найти запрошенный ресурс;
- **412 Precondition Failed** – сервер не может выполнить одно из условий запроса (например, если запрошенный ресурс с заголовком If-Unmodified-Since был изменен после указанной даты);
- **417 Expectation Failed** – сервер не может выполнить частично переданный запрос с заголовком Expect: 100-continue.

Коды состояния с ошибками сервера (5xx)

Коды состояния, начинающиеся с цифры "5" указывают на проблемы сервера, которые не позволяют успешно выполнит переданный запрос:

- **500 Internal Server Error** – внутренняя ошибка сервера;
- **501 Not Implemented** – сервер не может выполнять запрашиваемый метод;
- **505 HTTP Version Not Supported** – используемая браузером версия протокола сервером не поддерживается.

Заголовки сообщений

HTTP заголовки являются метаданными HTTP сообщений. Правильное использование заголовков позволяет разработать сложные web-приложения, которые устанавливают и поддерживают сеансы работы, задают политику кэширования данных, управляют аутентификацией и авторизацией, и реализуют бизнес логику. Спецификация HTTP протокола разделяет заголовки на следующие группы: общие заголовки, заголовки запросов, заголовки ответов и заголовки содержания.

К ***общим заголовкам***, которые могут задаваться, как в HTTP запросах, так и в HTTP ответах, относятся следующие заголовки:

- Date – задает время и дату создания данного сообщения, например «Date: Tue, 29 Apr 2008 22:28:31 GMT».
- Connection – указывает, будет ли клиент или сервер, сформировавший данное сообщение сохранять соединение открытым. Возможными значениями являются: "keep-alive" – сохранить соединение; "close" – закрыть соединение. Данное значение используется протоколом HTTP/1.1 по умолчанию (в отличие от протокола HTTP/1.0, который по умолчанию использует значение "close"). Например: «Connection: close».

- Warning – текст для использования человеком, полезен при выполнении отладки. Например: «Предупреждение: может быть нарушение безопасности!».

Заголовки запросов позволяют клиентам передавать дополнительную информацию о себе и запросе, например:

```
Host: www.neurozen.com
User-Agent: Mozilla/5.0 (Windows; U; Windows XP; en-US; rv: 1.8.0.11)
Referer: http: //www.mysite.ru/~ivanov/defaul.html
Authorization: Basic [encoded-credentials]
```

К ***заголовкам запросов*** относятся следующие заголовки:

- User-Agent – описывает программу, отправившую данный запрос (например, web браузер).
- Host – указывает серверу, какой web-сайт должен использоваться при обработке запроса (при использовании виртуального хостинга). Позволяет с помощью одного IP адреса обрабатывать много web-сайтов.
- Referer – указывает информацию о том, откуда данный запрос поступил. Если запрос был сформирован путем щелчка по гиперссылке на странице, то указывается URL адрес данной страницы.
- Authorization – передается с запросом к ресурсу, использование которого разрешено только авторизованным пользователям. Браузеры включают такой заголовок после того, получения ответа на предыдущий запрос с кодом 401 и получения запрашиваемых данных от пользователя (т.е., имени и пароля). Если сервер решит, что эти данные правильные (что будет показано успешным кодом состояния), то браузер будет продолжать включать их в последующие запросы доступа к ресурсам, находящиеся в той же области поддержки безопасности. Следует отметить, что разные браузеры могут по-разному определять время окончания передачи этих данных.

Заголовки ответов помогают серверу передать дополнительную информацию об ответе, которая не может быть определена только на основе анализа кода состояния:

```
Server: Apache/2.2.4
Location: http://www.mywebsite.com/relocatedPage.html
WWW-Authenticate:Basic realm="KremlinFiles"
```

К заголовкам ответов относятся:

- Location – задает URL адрес на который клиент должен перенаправить свой запрос. Используется с кодами состояния 301, 302, 303 и 307.
- WWW-Authenticate – задается вместе с кодом состояния 401, который указывает на проблему прав доступа к требуемому ресурсу. Значение данного заголовка указывает защищенную область, для получения ресурсов из которой должны быть предоставлены правильные данные авторизации. При получении в ответе кода состояния 401 и заголовка WWW-Authenticate, web браузером должен запросить у пользователей данные авторизации (т.е., имя и пароль).
- Server – содержится информация о web сервере, сформировавшим ответ (не обязательный заголовок).

Заголовки содержания (entity headers) описывают или содержание тела сообщения или (в запросах, не имеющих тела) целевой ресурс. Общими заголовками содержания являются следующие:

```
Content-Type: mime-type/mime-subtype
Content-Length: xxxx
Last-Modified: Tue, 29 Apr 2008 22:28:31 GMT
```

- Content-Type – описывает MIME тип содержания тела документа;
- Content-Length – задает длину тела сообщения, в байтах. Данное значение может использоваться для отображения процесса передачи сообщения;
- Last-Modified – если включено в ответ, то указывает дату последнего изменения содержания, передаваемого в теле сообщения.

Поддержка различных типов содержания

Браузер, получив в теле сообщения некоторое содержание (контент), должен определить, что с ним можно сделать. Он может:

- показать самостоятельно полученное содержание в своем окне в виде текста или HTML страницы;
- показать содержание в своем окне с помощью подключаемых компонентов (plug-in);
- запустить на выполнение вспомогательное приложение, которое может показать пользователю содержание, не являющееся HTML страницей;
- сообщить о невозможности показать переданные данные.

Для описания типа содержания HTTP использует систему кодирования, использованную в электронной почте. Система кодирования типов содержания MIME (Multipurpose Internet Mail Extensions) это стандарт, разработанный для того, чтобы помочь почтовым клиентам показать не текстовое содержание, прикрепленных к электронным письмам данных.

Типы данных, содержащихся в теле HTTP сообщений, описываются двухуровневой упорядоченной моделью кодирования, с помощью заголовков:

- Content-Encoding – описывает способ кодирования (сжатия) содержания тела сообщения;
- Content-Type – описывает MIME тип содержания тела сообщения.

Для использования содержания тела сообщения оно вначале должно быть декодировано, в соответствии с методом кодирования заданным в заголовке Content-Encoding, а затем показано в соответствии с типом, заданным в заголовке Content-Type. В HTTP/1.1 заголовки Content-Encoding могут использовать следующие значения:

- gzip – содержание закодировано с помощью программы сжатия данных GNU zip (по умолчанию);
- compress – содержание закодировано с помощью программы сжатия данных compress, используемой в ОС Unix;
- deflate – содержание закодировано в формате zlib, описанном в Интернет стандартах RFC 1950 и 1951.

Очевидно, что если web серверы кодируют (сжимают) содержание с помощью таких методов кодирования, то web браузеры (и другие клиенты) должны уметь выполнять декодирование тела сообщения до того, как выполнять его визуализацию и обработку.

Например, когда web-сервер включает в ответ заголовок Content-Encoding: gzip, то браузер знает, как его декодировать, перед тем, как показать пользователю. Например, браузер является достаточно способным, чтобы распаковать сжатый файл с документом Microsoft Word (например, test.doc.gz), а затем автоматически вызвать программу Microsoft Word, чтобы показать пользователю исходный файл test.doc.

Заголовку Content-Type задается код типа содержания (media type), которое определяется в виде комбинации типа, подтипа и пар «атрибут-значения»:

тип "/" подтип [";" строка-параметров]

Код типа задает основной вид содержания, а код подтипа задает уточняющий вид содержания. Например, наиболее часто используемым типом содержания является следующий:

```
Content-Type: text/html
```

В данном заголовке типом является значение text, а подтипом – html, что сообщает браузеру визуализировать тело сообщения в виде HTML страницы. Другим примером является:

```
Content-Type: text/plain; charset='us-ascii'
```

Здесь подтипом задано значение plain. Строка параметров передается клиентской программе, которая будет окончательно обрабатывать тело сообщения, тип содержания которого задан, как text/plain. Данный параметр может влиять на то, как клиентская программа будет обрабатывать переданное содержание. Если данный параметр не известен используемой программе, то она его просто проигнорирует. Описание основных MIME типов и подтипов приведено в табл. 1.1.

Таблица 1.1

Описание типов данных MIME

Тип/подтип	**Расширение файла**	**Описание**
application/pdf	.pdf	Документ, предназначенный для обработки Acrobat Reader
application/msexcel	.xls	Документ в формате Microsoft Excel
application/msword	.doc	Документ в формате Microsoft Word
application/rtf	.rtf	Документ в формате RTF, отображаемый с помощью Microsoft Word
image/gif	.gif	Изображение в формате GIF
image/jpeg	.jpeg, .jpg,	Изображение в формате JPEG
image/tiff	.tiff, .tif	Изображение в формате TIFF
text/plain	.txt	ASCII-текст
text/html	.html , .htm	Документ в формате HTML
audio/midi	.midi, .mid	Аудиофайл в формате MIDI
audio/x-wav	.wav	Аудиофайл в формате WAV
message/rfc822		Почтовое сообщение
message/news		Сообщение в группы новостей
video /mpeg	.mpeg, .mpg, .mpe	Видеофрагмент в формате MPEG
video/avi	.avi	Видеофрагмент в формате AVI

Поддержка сеансов работы пользователей

Как было отмечено ранее, HTTP является протоколом без поддержки состояния сеанса работы. Однако часто требуется выполнять поддержку сеанса работы, т.е. сохранять на сервере данные переданные в одном запросе, чтобы их можно было использовать при обработке последующих запросов в данном сеансе работы. Например, в web-приложении, позволяющем выполнять выбор и покупку товаров, вначале выполняется просмотр страниц каталога товаров, затем постепенно выбираемые товары добавляются в «корзину» (ссылки на них должны сохраняться где-то на сервере). Когда пользователь закончит выбирать товары и перейдет к оформлению заказа и оплате, то web-приложение должно помнить, какие товары были выбраны (помещены в «корзину»).

Для поддержки состояния между HTTP запросами необходимо иметь некоторый способ для участников взаимодействия передавать в HTTP запросах ссылки на информацию о состоянии сеанса работы. Протокол HTTP/1.1 позволяет это делать с помощью ***куки*** (cookie), работа с которыми выполняется с помощью заголовков Set-Cookie и Cookie:

- Заголовок ответа Set-Cookie отправляется сервером браузеру и в нем содержится информация о состоянии или идентификатор сеанса, который ссылается на состояние, хранимое на стороне сервера.
- Заголовок ответа Cookie передается браузером в последующих запросах тому же самому (или связанному с ним) серверу. В нем содержатся или элементы самой информации состояния или ***ссылки на идентификатор сеанса***, который помогает связать запросы с состоянием текущего сеанса.

Серверные web-приложения могут использовать заголовок Set-Cookie следующим образом:

```
Set-Cookie: <имя>=<значение>
      [; Max- Age=<значение >] [; Expires=<дата>]
      [; Path=<путь>] [; Domain=<имя домена>]
      [; Secure] [; Version=<версия>]
```

Пара атрибут-значение, <имя>=<значение>, отправляется браузером назад приславшему их серверу в последующих запросах. Атрибут Max-Age задает максимальное время, в течение которого данный куки (cookie) может использоваться (в секундах). Атрибут Expires представляет собой устаревший (уже не рекомендуемый) способ определения времени жизни данного куки, путем задания даты окончания срока его использования. Атрибуты Path и Domain задают границы применения куки, т.е. для каких серверных доменов и URL путей должен использо-

ваться данный куки. Атрибут Secure сообщает браузеру выполнять передачу последующих заголовков Cookie по шифрованному соединению. И, наконец, атрибут Version указывает на версию спецификации управления состоянием.

Серверы могут также использовать более новый заголовок Set-Cookie2, который является немного измененной версией заголовка Set-Cookie. Рассмотрим пример, web-приложения, работающего на сервере с доменным именем cs.rutgers.edu, которое формирует заголовок Set-Cookie следующего вида:

```
HTTP/1.1  200  OK
Set-Cookie: Name="Leon"; Path="/test/"; Domain=".rutgers.edu"; Version="1"
```

Здесь атрибут Domain имеет значение .rutgers.edu, а атрибут Path – /test/. Это указывает браузеру включать заголовок Cookie со значением Name=Leon каждый раз, при выполнении запроса к серверу Rutgers для ресурса, который URL путь к которому начинается с /test/. Отсутствие даты окончания срока использования означает, что данный куки должен поддерживаться только в течение текущего сеанса работы браузера.

Виртуальный хостинг

Одной из проблем, которую позволил решить протокол HTTP/1.1, является поддержка виртуального хостинга, т.е. возможности связать множество имен web-сайтов (host names) с одним IP адресом. Например, один компьютер-сервер может содержать (выполнять хостинг) несколько web сайтов, имеющих разные доменные имена. Для этого web-серверу требуется некоторый способ определения запрашиваемого web-сайта. С этой целью в протокол HTTP/1.1 был включен заголовок Host, который должен задаваться в каждом запросе. Например:

```
GET   /q?s=YHOO    HTTP/1.1
Host: finance.yahoo.com
```

HTTP протокол с шифрованием

Даже если предположить, что сервер является безопасным, то этого все равно не достаточно для выполнения очень важных web приложений (например, выполняющих покупки по банковским картам). Это связано с тем, что HTTP сообщения, содержащие важную информацию, все равно являются уязвимыми (они могут перехватываться, изменяться, использоваться с нежелательными целями). Наиболее очевидным реше-

нием для управления такой информацией, конечно, является ее шифрование.

HTTPS (Hypertext Transfer Protocol Secure) является безопасной версией HTTP протокола. HTTPS и HTTP используют одинаковые сообщения, но в соответствии с HTTPS сообщения передаются с использования соединения по безопасному уровню сокетов (Secure Sockets Layer, SSL), перед передачей по сети они шифруются клиентом, а после получения они дешифруются, получившим их сервером. В отличие от HTTP, для HTTPS по умолчанию используется порт 443. В настоящее время HTTPS поддерживается наиболее популярными браузерами.

Для подготовки web-сервера для обработки https-соединений, администратор должен получить и установить сертификат для этого web-сервера. Сертификат состоит из двух частей (двух ключей) – public и private. Public-часть сертификата используется для зашифровывания передаваемых данных (трафика) от клиента к серверу в защищённом соединении, private-часть – для расшифровывания полученного от клиента зашифрованного трафика на сервере. Сертификат можно получить в – сертифицирующей компании (например, VeriSign) – это платная услуга. Сертификат должен быть подписан уполномоченной стороной (сертифицирующей компанией), которая будет гарантировать клиентам, что держатель сертификата является тем, за кого себя выдаёт.

Некоторые сайты используют собственные сертификаты. Существует возможность создать такой сертификат, не обращаясь в сертифицирующую компаниями. Такие сертификаты менее надёжны, чем сертификаты, подписанные сертифицирующими компаниями. Такое использование защищает от пассивного прослушивания, но без проверки сертификата каким-то другим способом (например, звонок владельцу и проверка контрольной суммы сертификата) этот метод не будет являться вполне безопасным.

Такая система также может использоваться для аутентификации клиента, чтобы обеспечить доступ к серверу только авторизованным пользователям. Для этого администратор обычно создаёт сертификаты для каждого пользователя и загружает их в браузер каждого пользователя. Также будут приниматься все сертификаты, подписанные организациями, которым доверяет сервер. Такой сертификат обычно содержит имя и адрес электронной почты авторизованного пользователя, которые проверяются при каждом соединении, чтобы проверить личность пользователя без ввода пароля.

1.4. Web страницы

Основными ресурсами web-сети являются ***web-страницы*** (page) и ***web-приложения***. Под web-страницей понимаются файл, который содержит информацию, описанную на языке гипертекстовой разметки HTML, и который может включать гиперссылки на другие web страницы. Иногда web-страницы также называются HTML документами. Web-страницы могут быть статическими или формироваться динамическими с помощью web-приложений.

Современные web страницы являются достаточно сложными информационными объектами, которые могут включать содержание, представление и поведение, описываемые с помощью разных стандартов Web сети (рис.1.5).

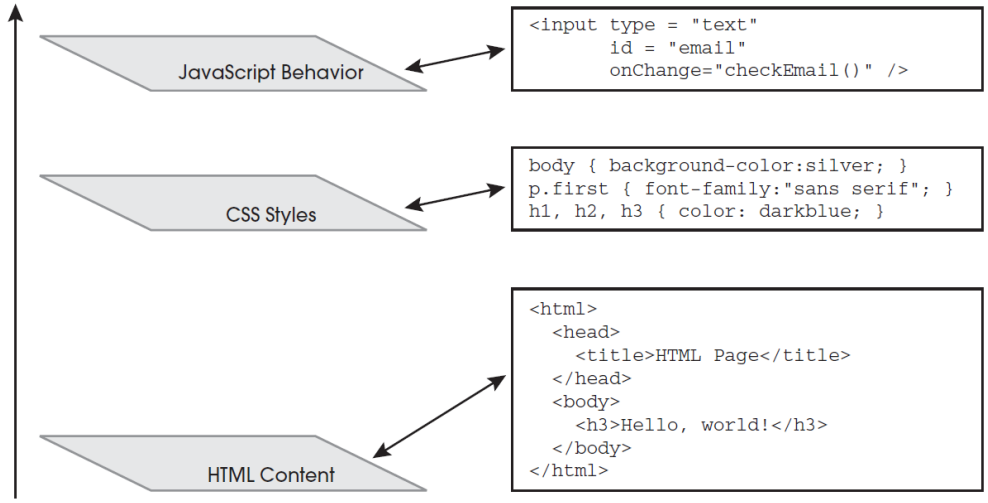


Рис. 1.5. Уровни описания web страниц

Содержание web-страниц задается с помощью языка гипертекстовой разметки HTML. Хорошим стилем разработки HTML кода является только описание и структурирование содержания, без использования имеющихся в языке HTML средств описания представления. Описание представления выполняется с помощью специального языка каскадных таблиц стилей, позволяющих задать способ представления описанного содержания.

Помимо содержания и описания способа его представления в web-странице может быть описан программный код, выполняемый браузером и задающий взаимодействие пользователя с данной страницей. Программный код может быть достаточно сложным и самостоятельно взаимодействовать с web-сервисами.

Язык гипертекстовой разметки HTML

HTML (**H**yper**T**ext **M**arkup **L**anguage) это язык разметки документов, размещаемых в web-сети (web-страниц). Первая версия данного языка была разработана Тимом Бернерс-Ли (изобретателем сети WWW) в 1991 году. HTML постоянно развивается. В 1999 году была опубликована версия HTML 4.01, которая используется до настоящего времени. После этого была предпринята попытка сделать HTML документы правильными XML-документами и была разработана версия языка XHTML 1.0. Длительное время велась разработка следующей версии языка XHTML 2.0, которая была прекращена без результата в 2009 году. В настоящее время активно ведется разработка первой версии языка HTML 5. Далее в пособии рассматривается версия языка разметки XHTML 1.0.

Структура документа HTML

Документ на языке HTML состоит из следующих обязательных частей:

- описания используемой версии языка HTML;
- корневой элемент html, включающий элементы head и body;
- заголовок (элемент head),
- тело, содержащее собственно сам документ (элементом body).

HTML документ начинается с описания *типа* документа (doctype), который имеет следующий вид:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Данный элемент описывает тип HTML документа (версию и DTD правила его проверки), чтобы браузер мог правильно обработать документ, и определить, соответствует ли он указанным правилам.

Все содержание web-страницы описывается в элементе html, начало которого выделено тэгом <html>, а окончание – тэгом </html>. Внутри элемента html должен быть задан элемент head. Он содержит информацию о документе (метаданные). В элемент head могут быть включены: элемент title, который определяет заголовок страницы; метаданные и ссылки на CSS и JavaScript.

После элемента head следует элемент body, в котором описывается реальное содержимое страницы. В примере, приведенном ниже, в элемент body включен только элемент заголовка первого уровня (h1), который содержит текст «Привет Мир!». Элементы часто содержат другие элементы. Тело документа всегда будет содержать множество вложенных друг в друга элементов.

Пример простого HTML документа пока ниже:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
            "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
   <head>
      <title> Простая страница </title>
   </head>
   <body>
      <h1>Привет Мир!</h1>
   </body>
</html>
```

Пример более сложной HTML страницы и ее отображение в web-браузере показано на рис. 1.6.

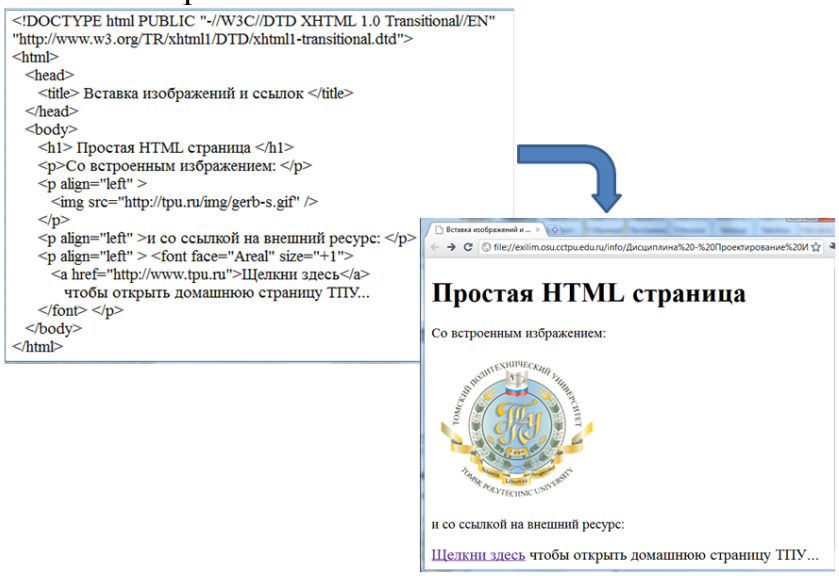


Рис. 1.6. Отображение простой HTML страницы в web-браузере

Синтаксис языка HTML

Документ на языке XHTML является правильно сформированным XML документом. Содержание HTML документа состоит из набора элементов, которые выделяются открывающими и закрывающими ***тэгами***. С помощью тэгов HTML документ делится на иерархически вложенные элементы. Описание элемента имеет следующий формат:

```
<имя_тэга имя_атрибута1=значение имя_атрибута2=значение …>
…
</имя_тэга >
```

Названия всех тэгов рассматриваемой версии языка HTML являются чувствительным к регистру (т.е., маленькие и большие буквы считаются разными) и должны записываться маленькими буквами.

Элементы могут включать текст и подэлементы (как например, элемент html содержит подэлементы head и body в приведенном выше

примере). Элементы должны быть правильно вложены друг в друга. Подэлементы включенные в элемент должны заканчиваться до закрывающего тэга элемента. Например, следующая запись является ошибочной:

```
<b><i>Это полужирный и наклонный текст</b></i>
```

Правильно данный фрагмент должен иметь следующий вид:

```
<b><i>Это полужирный и наклонный текст</i></b>
```

Если элемент не включает текст и подэлементы (пустой элемент), он все равно должен иметь открывающийся и закрывающийся тэги. Например, элемент br не имеет содежания и вызывает разрыв строки (продолжение текста начинается с новой строки). Однако в соответствии с правилами он должен записываться следующим образом:
</br> или сокращенно
. Точно также должен записываться и элемент <hr /> – вывод горизонтальной линии.

Открывающие тэги элементов могут иметь *атрибуты*, которые уточняют поведение элемента и задают дополнительные значения. Практически у каждого тэга имеется большое число необязательных параметров. Далее будут рассматриваться только основные атрибуты тэгов.

Многие атрибуты в HTML являются общими для всех элементов, но некоторые являются специфическими для данного элемента или элементов. Все они имеют форму:

```
имя_атрибута="значение"
```

Например:

```
<div id="mySection">
   <h1>Основы Hypertext Markup Language"</h1>
</div>
```

В данном примере элемент div (раздел HTML страницы, позволяющий разделять документы на логические блоки) имеет атрибут id, для которого задано значение mySection. Элемент div содержит элемент h1 (заголовок первого, или самого важного уровня), который в свою очередь содержит некоторый текст. Значения атрибутов должны быть помещены в одиночные или двойные кавычки.

Имена тэгов и атрибутов и их возможные значения определяются спецификацией языка HTML, и свои собственные тэги или атрибуты создавать нельзя.

У всех элементов можно задавать атрибуты id и class, значения которых используются для идентификации конкретных элементов (id) или

группы элементов (class). Эти атрибуты активно используются в скриптах документа и описаниях CSS стилей документа.

Основные категории элементов

Имеется две основные категории элементов в HTML: *блочные и строковые*.

- *Блочные элементы* обычно информируют о структуре документа. Такие элементы начинаются с новой строки, отрываясь от того, что было перед этим. Примерами блочных элементов являются *параграфы, пункты списка, заголовки и таблицы, контейнеры div*.
- *Строковые элементы* содержатся внутри структурных элементов блочного уровня и охватывают только части текста документа, а не целые области. Строковый элемент не приводит к появлению в документе новой строки, т.к. они являются элементами, которые появляются в параграфе текста. Примерами строковых элементов являются *ссылки, выделенные слова или фразы, контейнеры span*.

Заголовок HTML документа

Заголовок HTML документа используется для размещения информации, относящейся ко всему документу в целом. Он не является обязательным элементом разметки. Элемент разметки title служит для именования документа в web-сети. Например:

```
<head>
   <title>Моя страница</title>
   ...
</head>
```

Основными элементами, включаемыми в заголовок head, являются следующие: title – название документа; meta – метаинформация, связанная с документом; link – общие ссылки для данного документа; style – описатели стилей или ссылки на внешние ресурсы со стилями; script – скрипты или ссылки на внешние ресурсы со скриптами. Чаще всего применяются элементы title, script и style.

Элемент разметки meta

Элемент meta содержит управляющую информацию, которую браузер использует для правильного отображения и обработки содержания тела документа, например с помощью атрибута Content-type можно задать перекодировку документа на стороне клиента. С помощью тэга meta так же можно указать время последнего изменения документа (Last-Modified) или дату истечения срока его актуальности (Expire). Тэг meta также может использоваться для описания поискового образа документа. Например, для описания документа используется два meta-тэга.

Один определяет список ключевых слов, а второй – краткое содержание документа.

```
<meta name="description" http-equiv="description"
        content="Учебный курс проектирования web-приложений.">
<meta name="keywords" http-equiv="keywords" content="учебный курс; web-технологии; XHTML 1.0; заголовок; HTML; документ; элемент; разметка">
```

Элемент разметки link

Элемент link позволяет связать документ с другими документами. Он имеет следующий вид:

```
<link [href=url] [rel=тип_отношения] [type=тип_содержания]>
```

В частности элемент LINK позволил загружать внешние описатели стилей:

```
<link href="../css/style.css" rel="stylesheet" type="text/css">
```

В данном случае атрибут rel определяет тип гипертекстовой связи, атрибут href указывает адрес документа, идентифицирующего связь, а атрибут type определяет MIME тип содержания данного документа.

Элемент разметки style

Элемент разметки style предназначен для размещения описателей стилей для всего документа. В настоящее время данный элемент используется только с одним атрибутом type, который задает тип описателя стиля. Это может быть либо text/css, либо text/javascript. В общем виде запись элемента style выглядит следующим образом:

```
<style type=тип_описания_стилей>
    описание стиля/стилей
</style>
```

Элемент разметки script

Элемент разметки script служит для размещения кода JavaScript, VBScript или JScript (по умолчанию подразумевается JavaScript). Данный элемент можно использовать не только в заголовке документа, но и в теле. В отличие от элемента style, ему не требуется дополнительный контейнер link для загрузки внешних файлов кодов. Это можно сделать непосредственно в самом элементе script:

```
<script  type="text/javascript"   src="script.code">
```

В общем виде запись данного элемента выглядит следующим образом:

```
<script [type=тип_языка_сценариев]>
    [JavaScript / VBScript – код]
```

</script>

или

<script [type=тип_языка_сценариев] [src=url]> </script>

Элементы тела документа

Элементы тела документа предназначены для управления отображением информации в программе интерфейса пользователя. Они описывают гипертекстовую структуру базы данных при помощи встроенных в текст контекстных гипертекстовых ссылок. Тело документа состоит из следующих элементов:

- иерархических контейнеров и заставок;
- заголовков (от h1 до h6);
- блоков (параграфы, списки, формы, таблицы, картинки и т.п.);
- текста, разбитого на области действия стилей (подчеркивание, выделение, курсив);
- графики и гипертекстовых ссылок.

Описание элементов тела документа следует начинать с тэга body. В отличие от тэга head, тэг body имеет атрибуты. Атрибут background определяет фон, на котором отображается текст документа. Если источником для фона HTML- документа является графический файл, например, image.gif, то в открывающем тэге тела body появляется соответствующий атрибут:

<body background="image.gif">

Следует отметить, что имеются различные дополнительные атрибуты для тэга body. Значения атрибутов bgcolor, text, link, alink и vlink задаются либо RGB-значением в шестнадцатеричной системе, либо одним из 16 базовых цветов.

Таблица 1.2

Основные атрибуты HTML

Атрибут	Описание
marginheight	определяет ширину (в пикселях) верхнего и нижнего полей документа
topmargin	определяет ширину (в пикселях) верхнего и нижнего полей документа
marginwidth	определяет ширину (в пикселях) левого и правого полей документа
leftmargin	определяет ширину (в пикселях) левого и правого полей документа

Окончание табл. 1.2

Атрибут	Описание
background	определяет изображение для «заливки» фона; значение задается в виде полного URL или имени файла с картинкой в формате GIF или JPG
bgcolor	определяет цвет фона документа
text	определяет цвет текста в документе
link	определяет цвет гиперссылок в документе
alink	определяет цвет подсветки гиперссылок в момент нажатия
vlink	определяет цвет гиперссылок на документы, которые были ранее просмотрены

Например:

```
<html>
<head> <title> Пример атрибутов body </title> </head>
<body background="image.jpg" bgcolor="black" text="#FFFFFF" link="#FF0000"
      vlink="#656565" marginheight="25" TOPMARGIN="25" leftmargin="45"
      marginwidth="45">
   [Текст документа.]
</body>
</html>
```

Таблица 1.3

Тэги управления разметкой

Тэги	Описание тэгов	Атрибуты	Описание атрибутов
от <h1> до <h6>	Заголовки задают начало раздела документа. В стандарте определено 6 уровней заголовков: от h1 до h6. Текст, окруженный тэгами , получается большим – это основной заголовок. Если текст окружен тэгами , то он выглядит несколько меньше (подзаголовок); текст внутри еще меньше и так далее до .		
<p>	Тэг <p> применяется для разделения текста на параграфы. В нем используются те же атрибуты, что и в заголовках. Позволяет выровнять текст по левому или правому краю, по центру или ширине. По умолчанию текст выравнивается по левому краю. Данный атрибут применим также к графике и таблицам.	align	Возможные значения атрибута: justify, left, right,center.

Окончание табл. 1.3

Тэги	Описание тэгов	Атрибуты	Описание атрибутов
 	Принудительный перевод строки. Используется для остановки в указанной точке обтекание объекта текстом и затем продолжения текста за объектом.	clear	Возможные значения атрибута: left, right, all.

Элементы – контейнеры

Элементы-контейнеры никакого действия браузера не вызывают, а используются только для логического деления HTML документа на части. В контейнеры (между начальным и закрывающим тэгами) размещают другие элементы, которыми надо управлять совместно.

Такими элементы создаются с помощью тэгов div и span. С помощью тэга div создается блочный контейнер, который начинается с новой строки, и следующий за ним элемент также будет начинаться с новой строки. А с помощью тэга создается строчный элемент, который размещается в той же самой строке, что и окружающий его текст. Основными атрибутами контейнеров являются id и class. Следует помнить, что строчные контейнеры могут включать только другие строчные контейнеры, а блочные контейнеры могут включать как строчные, так и блочные контейнеры.

Элементы-контейнеры позволяют применять правила каскадных таблиц стилей (CSS) к содержимому HTML документа. Пример HTML страницы, в которой используются контейнеры и CSS стили (язык CSS описан в следующем разделе пособия) показан ниже:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
            "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
   <head>
      <style style = "text/css">
         body { font-family: Verdana; font-size: 9pt; text-align: right; }
         div { font-family: Georgia; }
         .important { background-color: #ffffde; border: thin black ridge; }
      </style> </head>
   <body> Здесь начинается текст тела страницы.
     <div>А здесь идет текст div контейнера. Этот текст пишется другим
     шрифтом.</div> Текст тела страницы продолжается
     <span> в span контейнере </span> и здесь заканчивается.
     <div class="important">Это очень важная информация в другом div контейнере
     !</div> А здесь текст тела страницы заканчивается.
   </body>
</html>
```

Браузер отображают данную страницу так, как показано на рис. 1.7.

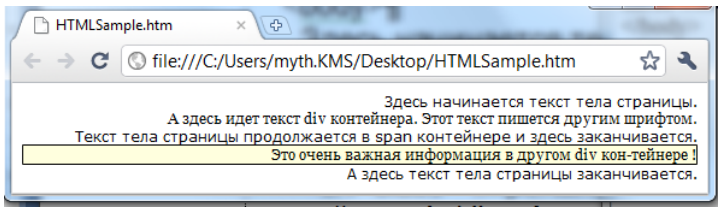


Рис. 1.7. Отображение HTML страницы с использованием описанной в ней таблицы стилей

Списки в HTML

Одним из часто используемых способов структурирования текстов является использование списков. В HTML имеются следующие виды списков: ненумерованный список (тэг <ul>) и нумерованный список (тэг <ol>);

Ненумерованный список записывается в виде последовательности:

```
<ul>
    <li>первый элемент списка</li>
    <li>второй элемент списка</li>
</ul>
```

Можно задавать любой тип маркера в произвольном месте списка. Ниже перечислены тэги с атрибутами стандартных маркеров:

- <ul type="disc"> – стандартные маркеры списков первого уровня (по умолчанию);
- <ul type="circle"> – маркеры в виде кругов;
- <ul type="square"> – квадратные маркеры.

Тэг <ol> вместе с атрибутом type позволяет создавать *нумерованные* списки, используя в качестве номеров не только обычные числа, но и строчные и прописные буквы, а также строчные и прописные римские цифры:

- <ol type="1"> – создает список с нумерацией в формате 1., 2., 3...;
- <ol type="A"> – создает список с нумерацией в формате A., B., C...;
- <ol type="I"> – создает список с нумерацией в формате I., II., III....

Комментарии

Комментарии используются для пояснения документа и не отображаются браузерами. В HTML они начинаются с символа "<!--" и заканчиваются символом "-->".

Гипертекстовые ссылки

Для записи гипертекстовой ссылки используется тэг <a>, который называют «якорем». Якорь имеет несколько атрибутов, главным из которых является href. Простую ссылку можно записать в виде

```
<a href="http://www.example.com"> Пример гипертекстовой ссылки </a>
```

Значением атрибута href является URL адрес документа, на который браузер будет выполнять переход.

Вставка изображений в HTML

Для вставки изображения используется тэг <img> с обязательным атрибутом src, значением которого является имя выводимого графического файла. Замыкающего тэга не требуется. Пример вставки изображения:

```
<img src="image.gif" alt="изображение" />
```

Изображения на web-странице могут использоваться в качестве гипертекстовых ссылок, как и обычный текст. Пользователь браузера может щелкнуть по изображению и перейти к другой странице. Для обозначения изображения как гипертекстовой метки используется тот же тэг <a>, что и для текста, но между <a> и вставляется тэг изображения <img>. Тэг <img> также имеет несколько необязательных атрибутов: alt, align, usemap, hspace, vspace, border, width, height. Основные атрибуты тэга <img> описаны в табл. 1.4.

Таблица 1.4

Основные атрибуты тэга <img>

Атрибут	**Описание**
src	Указывает файл изображения и путь к нему.
alt	Позволяет указать текст, который будет выводиться вместо изображения браузерами, неспособными представлять графику.
align	Определяет положение изображения относительно окружающего его текста.
usemap	Если щелкнуть кнопкой мыши на активной области изображения, для которого определен атрибут usemap, произойдет переход по ссылке на документ, заданный для этой области.
border	Определяет толщину рамки вокруг изображения. Если значение равно нулю, рамка отсутствует.
hspace	Задает горизонтальное расстояние между вертикальной границей страницы и изображением, а также между изображением и огибающим его текстом.

Окончание табл. 1.4

vspace	Задает вертикальное расстояние между строками текста и изображением.
width , height	Задают значения размеров изображения по горизонтали и по вертикали соответственно.

Описание таблиц в HTML

Для описания таблиц используется тэг <table>. Строки таблицы задается с помощью тэгов <tr>…</tr>. Внутри строки таблицы обычно размещаются элементы (ячейки) с данными. Каждая ячейка, содержащая текст или изображение, должна быть окружена тэгами <td>…</td>. Заголовки для столбцов и строк таблицы задаются с помощью тэга заголовка <th>…</th>. Эти тэги подобны <td>…</td>, однако текст, заключенный между тэгами <th></th>, автоматически выделяется жирным шрифтом и по умолчанию располагается посередине ячейки. Тэг <caption> позволяет создавать заголовки таблицы. Основные атрибуты тэга <table> описаны в табл. 1.5.

Таблица 1.5

Основные атрибуты тэга <table>

Атрибут	**Описание**
nowrap	При использовании атрибута nowrap с тэгами <th> или <td> длина ячейки расширяется настолько, чтобы заключенный в ней текст поместился в одну строку
colspan	Если требуется сделать какую-нибудь ячейку шире, чем верхняя или нижняя, то можно воспользоваться атрибутом colspan, чтобы расширить ее на любое количество обычных ячеек.
rowspan	Подобен атрибуту colspan, только он задает число строк, на которые расширяется ячейка
width	Можно поместить его в тэг <table>, чтобы задать ширину всей таблицы, или можно использовать в тэгах <td> или <th>, чтобы задать ширину ячейки или группы ячеек. Ширину можно указывать в пикселах или в процентах.
cellpadding	Определяет ширину пустого пространства между содержимым ячейки и ее границами
align, valign	Тэги <tr>, <td> и <th> можно модифицировать с помощью атрибутов align и valign. Атрибут align определяет выравнивание текста и графики по горизонтали. valign осуществляет выравнивание текста и графики внутри ячейки по вертикали.
border	В тэге <table> определяет, как будут выглядеть рамки, то есть линии, окружающие ячейки таблицы и саму таблицу

Окончание табл. 1.5

cellspacing	Определяет ширину промежутков между ячейками в пикселах. Если этот атрибут не указан, по умолчанию задается величина, равная двум пикселам
bgcolor	Позволяет установить цвет фона. В зависимости от того, с каким тэгом он применяется, цвет фона может быть установлен для всей таблицы (table), для строки (tr) или для отдельной ячейки (td). Значением данного атрибута является RGB-код или стандартное название цвета.
background	Задает фоновое изображение для таблиц. Применим к тэгам table и td. Его значением является URL файла с фоновым изображением.

Таблицы рекомендуется использовать только для отображения табличных данных, а не для задания структуры web-документа. Для задания структуры документа рекомендуется использовать CSS стили, как будет описано далее.

Формы HTML

Для ввода пользователем исходных данных и их передачи браузером на сервер для обработки используются HTML формы. Такие формы задаются с помощью тэга <form>. В данном тэге с помощью атрибутов method и action описывается способ обработки данных, вводимых пользователем в форму (см. рис. 1.4):

- Атрибут method задает HTTP метод отправки данных формы на сервер. Такими методами являются Get или Post.
- Атрибут action задает адрес ресурса сервера, которому будут передаваться данные. Это может быть серверный скрипт или шаблон web-страницы. Если атрибут action не задан, то данные будут передаваться на тот же адрес, по которому была получена страница с данной формой.

В элемент формы вставляются тэги <input> для задания элементов управления, которые формируют поля для ввода информации пользователями. Это могут быть текстовые поля, флажки, кнопки и т.п. Вид поля ввода определяется значением атрибута type. Описание типов элементов, включаемых в форму, приведено в табл. 1.6.

Таблица 1.6

Описание типов элементов, включаемых в HTML форму

Тип элемента	**Описание**
text	Используется, если необходимо ввести небольшое количество текста (одну или несколько строк). Кроме того, задается атрибут name для определения наименования переменной поля. Имеет еще три дополнительных атрибута: • maxlength – ограничивает число символов, вводимых пользователем в текущее поле. По умолчанию данное число не ограничено. • size – определяет размер видимой на экране области, занимаемой текущим полем. Значение по умолчанию определяется типом браузера. • maxlength – если его значение больше, чем size, браузер будет прокручивать данные в окне. • value – обеспечивает начальное значение поля ввода
checkbox	Применяется для создания флажков, у которых пользователь может задать состояние «выбран/невыбран». Должны быть заданы атрибуты name, и value. В некоторых случаях необходимо инициализировать данный флажок, как уже выбранный. В этих случаях тэг <input> должен содержать атрибут checked.
radio	Применяется в случае, когда требуется организовать выбор одного из нескольких возможных значений (альтернатив). Должны быть также указаны атрибуты name и value.
image	Когда пользователь щелкает мышью по изображению, браузер сохраняет координаты соответствующей точки экрана. Далее он «обрабатывает» введенную в форму информацию. Должны быть указаны также атрибуты name и src. Атрибут src содержит URI файла – источника изображения. Атрибут align является дополнительным и используется аналогично тому же атрибуту тэга <img> .
password	Используя данный тип, можно организовать ввод пароля без вывода на экран составляющих его символов. При этом следует помнить, что введенные данные передаются по незащищенным каналам связи и могут быть перехвачены.
button	Используется для размещения кнопок на web-странице. Имеет атрибуты name и value.
file	Элемент управления для запуска диалогового окна выбора файла. Для задания имени файла по умолчанию можно использовать атрибут value.

Окончание табл. 1.6

Тип элемента	Описание
submit	Используется как элемент интерфейса для завершения ввода данных. Может содержать два дополнительных атрибута: name и value.
reset	Используется как элемент интерфейса для сброса всех введеных в поля формы данных. Дополнительно содержать атрибут value. Данный атрибут определяет надпись на изображении кнопки.
hidden	Позволяет включить в отправляемую форму значения атрибутов name и value, недоступные для изменения пользователем.

Тэг <textarea> используется для создания текстового поля из нескольких строк. Данный тэг использует три атрибута: cols (число колонок, содержащихся в текстовой области), name и rows (количество видимых строк текстовой области).

Тэг <select> используют для определения списка пунктов, задаваемых тэгами <option>. Тэг <select> поддерживает три необязательных атрибута: multiple, name и size: multiple – позволяет выбрать более чем одно значение; name – определяет наименование объекта; size – определяет число видимых пользователю пунктов списка.

В форме может использоваться тэг <option> только внутри тэга <select>. Эти тэги поддерживают два дополнительных атрибута: selected и value. Атрибут selected используется для первоначального выбора значения элемента по умолчанию. Атрибут value указывает на значение, возвращаемое формой после выбора пользователем данного пункта. Например:

```
<br>Ваш выбор:
<select name="choice">
    <option value="Вариант 1">Вариант 1</option>
    <option value="Вариант 2">Вариант 2</option>
    <option value="Вариант 3">Вариант 3</option>
    <option selected>Вариант 4</option>
</select>
```

Язык каскадных таблицей стилей CSS

В то время как язык HTML описывает содержание web-документа, язык CSS дает браузеру инструкции о том, как показать различные элементы web-документа. Есть разные версии языка CSS. Текущим стандартом языка является CSS2 (вторая версия) утвержденная организацией W3C в 1998 г.

Язык CSS позволяет описать внешний вид документа, составленного с использованием языка разметки, и позволяет решать две основные задачи:

- определить внешний вид элементов (цвет, шрифт, размер и т.п.);
- задать позиционирование блоков документов (слева, справа, снизу, в заданном месте и т.п.).

Использование CSS дает следующие преимущества:

- Возможность создавать несколько способов оформления одной и той же страницы для разных устройств просмотра. Например, на экране дизайн будет рассчитан на большую ширину, во время печати меню не будет выводиться, а на КПК и сотовом телефоне меню будет следовать за содержимым.
- Уменьшение времени загрузки страниц сайта за счет переноса правил представления данных в отдельный CSS-файл. В этом случае браузер загружает только структуру документа и данные, хранимые на странице, а представление этих данных загружается браузером только один раз и кэшируется.
- Простота последующего изменения оформления страниц сайта. Не нужно описывать оформление каждой страницы, а достаточно лишь указать CSS-файл, в котором описано требуемое оформление.
- Дополнительные возможности оформления. Например, с помощью CSS можно сделать блок текста, который остальной текст будет обтекать (например, для меню) или сделать так, чтобы меню было всегда видно при прокрутке страницы (фиксированное местоположение).

Однако CSS имеет и некоторые недостатки:

- Различное отображение верстки в различных браузерах (особенно устаревших), которые по-разному интерпретируют одни и те же данные CSS.
- На практике часто возникает необходимость исправлять не только CSS-файл, но и HTML документ, тэги которого сложным способом связаны с селекторами CSS, что усложняет применение единых файлов стилей и значительно удлиняет время редактирования и тестирования.

Описание правил стилей

Отображение элементов HTML документа в браузере описывается с помощью набора правил, которые определяют, каким образом должны быть оформлены и позиционированы разные HTML элемент. В каждом правиле задается набор свойств (например, цвет, размер, шрифт, и т.д.)

этих элементов HTML и их значений. Правило CSS имеет следующий общий вид:

```
селектор {
    свойство1: значение;
    свойство2: значение;
    свойство3: значение;
}
```

Селектор определяет способ отбора HTML элементов, к которым будет применяться данное правило. Например, он может соответствовать реальным названиям элементов (например, body) или задавать элементы, имеющие некоторое значение атрибутов class либо id.

В фигурных скобках { } записываются пары «свойство: значение», которые указывают, какие значения будут задаваться свойствам отобранных элементов. *Свойства* определяют, что требуется сделать с выделенными элементами. Они могут задавать, например, цвет элемента, цвет фона, позицию, поля, заполнение, тип шрифта, и многое другое.

Значения являются конкретными данными, которые требуется задать каждому *свойству* выделенных элементов. Каждое свойство имеет свой набор присваиваемых значений.

Все множество подобных правил совместно формируют таблицу стилей.

Описание значений свойств

Свойства, которые влияют на положение, поля, ширину, высоту и т.д. могут измеряться в *пикселях (px), в высоте шрифта элемента (em), процентах (%), миллиметрах (mm)* и т.п..

Например, следующее правило:

```
p { margin: 10px; font-family: Times New Roman; color: green; }
```

указывает, что вокруг параграфов документа (селектор <p>) создаются поля (margin) размером в 10 пикселей, для вывода текста используется шрифт (font-family) Times New Roman, а цвет текста параграфа (color) является зеленым (green).

Цвета в CSS могут задавать разными способами: в виде названия цвета (всего имеется 16 таких цветов, например, gray, green, black); в виде rgb кода (например, RGB(255,255,0) – желтый цвет) или в виде шестнадцатеричного кода (например, #a52a2a – коричневый цвет).

В таблицы CSS стилей можно включать **комментарии**, помещая их между символами /* и */. Комментарии могут содержать несколько строк текста, которые будут игнорироваться браузером.

Объединение селекторов в группу

Различные селекторы также могут объединяться в группу. Например, если требуется применить одинаковое оформление к h2 и p, тогда можно написать следующий CSS:

```
h2 {color: red}
p  {color: red}
```

Однако, код CSS можно сократить, группируя селекторы вместе с помощью запятой – правила в скобках применяются к обоим селекторам: h2, p {color: red}

Основные типы селекторов правил CSS

Имеются следующие основные типы селекторов:

- ***символ звездочка*** (*) – правило применяется ко всем элементам документа. Например:

```
* { font-family: sans-serif } /* Применяется ко всем элементам документа */
```

- ***селектор типа*** – совпадает с именем элемента в документе и указывает, что его субъектами являются все элементы документа с данным именем. Например:

```
H1 { font-family: sans-serif } /* Применяется ко всем элементам H1 */
```

- ***селекторы классов*** .class применяется ко всем элементом с заданным значением атрибута class. Например, правило

```
.warning { font-style: italic }
```

задает стиль (курсив шрифта) для всех элементов, имеющих атрибут class="warning" (например, для элемента).

- ***селектор идентификаторов*** #id применяется ко всем элементом с заданным значением атрибута id. Например, правило

```
#sample { letter-spacing: 0.3em }
```

будет применяться к элементу Разреженный текст

- имеется несколько селекторов атрибутов, например:
  - [attr] – применяется ко всем элементам, имеющим атрибут attr, независимо от его значения.
  - [attr=value] – применяется ко всем элементам, у которых атрибут attr имеет значение value.

Использование CSS к HTML странице

Существует три способа применения CSS к документу HTML: строковые, вложенные и внешние таблицы стилей.

Строковые стили позволяют задать стили у элемента, используя атрибут style следующим образом:

```
<p style="background:blue; color:white; padding:5px;">Hello</p>
```

Внутри этого атрибута перечисляются все свойства CSS и их значения. Преимущество строковых стилей состоит в том, что браузер будет вынужден использовать эти настройки. Все другие стили, определенные в других таблицах стилей, или даже вложенные в заголовок документа, будут переопределены этими стилями.

Вложенные стили записываются в заголовке документа внутри элемента style, как в следующем примере страницы:

```
<style type="text/css" media="screen">
   p { color:white; background:blue; padding:5px; }
</style>
```

Преимущество вложенных таблиц стилей состоит в том, что не нужно добавлять атрибут style в каждый параграф – можно оформить их все с помощью одного единственного определения. Это означает также, что если потребуется изменить внешний вид всех параграфов, это можно будет сделать в одном единственном месте, однако все это, тем не менее, ограничено одним документом.

Внешние таблицы стилей позволяют разместить все определения CSS в отдельном файле с расширением CSS, и затем применение его к документам HTML, используя элемент link в заголовке документа. Например:

```
<link href="styles.css" rel="stylesheet" type="text/css">
```

Внешние таблицы стилей позволяют хранить все определения оформления в одном единственном файле. Это означает, что можно вносить изменения на всем сайте, изменяя только один файл. Web-браузер при этом может загрузить его один раз и затем кэшировать для всех других документов, которые на него ссылаются, что уменьшает объем загружаемой информации.

Позиционирование элементов страницы

Когда браузер получает данные от сервера, он выводит их на страницу в том порядке, в котором они записаны – этот порядок называется нормальным потоком. То есть браузер получает какой-то элемент, вычисляет его размеры и располагает следом за предыдущим элементом. При обработке данных из нормального потока браузер выводит данные строку за строкой, добавляя перевод строки при появлении структурного блока. В связи с этим, в частности, с помощью стандарт-

ного HTML кода нельзя организовать вывод, например, нескольких колонок текста. В связи с этим, первоначально для задания взаимного расположения частей web-страницы на экране использовались таблицы (элементов table), имеющих нулевую толщину границы (т.е. таблицы на экране не показывается). В ячейках таблица размещаются элементы HTML страницы.

Однако с помощью каскадных таблиц стилей CSS стало возможным задавать не только оформление элементов HTML документа, но и взаимное расположение (позиционирование) отдельных его частей (блоков). Позиционирование задается с помощью свойства position. Есть два способа позиционирования блока в нормальном потоке: relative и static. Способ static используется по умолчанию и выполняет стандартное форматирование, оставляя блок в нормальном потоке. Способ relative позволяет сдвинуть положение блока относительно того положения, которое он занимал бы в нормальном потоке. Например, если задано

```
position: relative; left: 20px; color: green;
```

то положение левого края блока сдвигается на 20 пикселей относительно предыдущего элемента в нормальном потоке, в результате чего сам блок смещается вправо. Если указать отрицательное смещение, то блок сдвинется влево. Аналогично можно сдвигать блок по вертикали – для этого используется свойство top.

Кроме этого, имеется другой способ позиционирования – «абсолютное позиционирование», позволяющее точно указать расположение блока на странице. В этом случае блок изымается из нормального потока размещения и помещается браузером в заданное место экрана. Использовать абсолютное позиционирование достаточно удобно в строгих сайтах с фиксированным размером страницы. Наиболее типичной является ситуация, когда страница разбивается на несколько абсолютно позиционированных блоков, внутри которых используется относительное позиционирование (а так как вложенные блоки позиционируются относительно родителя, то структура страницы сохраняется).

Еще одним способом позиционирования является «плавающее позиционирование»: float. Плавающие блоки ведут себя очень похоже на картинки с указанным тегом align: они прижимаются к указанному краю контейнера, заставляя "нормальный поток" их обтекать. Такие блоки довольно удобно использовать для организации так называемой «резиновой верстки», когда заданы несколько плавающих блоков, которые располагаются один за другим по горизонтали (если их суммарная ширина не превышает ширины страницы). Для того, чтобы заставить ото-

бражать блок ниже плавающего блока нужно использовать атрибут clear (значение both).

При использовании плавающих блоков надо обязательно задавать их ширину (иначе они будут растягиваться до ширины контейнера), и кроме того, желательно определить поля для элемента body, иначе в некоторых браузерах (например, в версиях Internet Explorer) появятся большие поля вокруг плавающего блока.

Блочная верстка HTML документов

В настоящее время задание структуры сайтов (верстка) выполняется не с помощью таблиц, а путем использования так называемой «блочной верстки» (или div верстки). Основная идея данного способа состоит в том, что в самих web-страницах расположение элементов не задается. Просто их содержимое разделено на логические блоки, описываемые с помощью div контейнеров, имеющих разные значения атрибута id. При отображении страницы в браузере такие блоки позиционируются на экране требуемым образом с помощью стилей позиционирования. Такое размещение блоков создает каркас, который уже затем наполняют включенным в них содержимым. Достоинством данного способа структурирования HTML страницы заключается в том, что структуру web-страниц можно очень просто изменить путем изменения используемых каскадных таблиц.

На рис. 1.8 показана стандартная структура web-страницы, заголовок, три колонки и подвал.

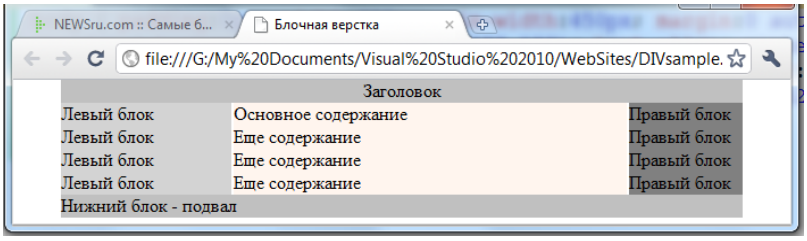


Рис. 1.8. Стандартный шаблон web-страницы

Для создания web-страницы с такой структурой используется блочная верстка HTML документа (рис. 1.9), содержащего четыре div блока (с id атрибутами header, left, content и footer). В связанной с данным документом таблице стилей DivStyle.css (рис. 1.10) для всех div контейнеров заданы свои стили. Для контейнера left задан размер (width:150px;) и плавающее позиционирование (float:left – блок размещается слева и последующий блок «обтекает» его). Для контейнера content задано смещение слева и справа (margin-left:152px; margin-right:102px;) для расположения его между левой и правой колонками. Для контейнера right задан

размер (width:110px;) и плавающее позиционирование (float:right – размещается справа и последующий блок «обтекает» его). И, наконец, для контейнера footer задано, что он размещается под всеми плавающими блоками (clear:both;). Кроме этого задан стиль для тэга body, в котором указывается ширина (width:600px;) области отображения документа, расположение ее по центру экрана (margin:0 auto;) и указывается прекращение ее сжатия при достижении минимального размера (min-width:450px).

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
            "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
    <title>Блочная верстка</title>
    <link href="DivStyle.css" rel="stylesheet" type="text/css" />
</head>
<body>
    <div id="header"> Заголовок </div>
    <div id="left"> Левый блок <br/>Левый блок <br/>Левый блок <br/>Левый блок</div>
    <div id="right"> Правый блок <br/> Правый блок <br/> Правый блок <br/> Правый блок</div>
    <div id="content"> Основное содержание <br/> Еще содержание <br/> Еще содержание <br/> Еще содержание </div>
    <div id="footer"> Нижний блок - подвал </div>
</body>
</html>
```

Рис. 1.9. HTML страница, использующая ссылку на внешнюю таблицу стилей DivStyle.css и разделеная на div блоки

```
body { width:600px; min-width:450px; margin:0 auto; padding:0px; }
#header { text-align:center; background-color:rgb(192,192,192); }
#left { float:left; width:150px; background-color:rgb(211,211,211); }
#content { margin-left:152px; margin-right:102px; background-color:rgb(255,245,238);}
#right { float:right; width: 100px; background-color:Gray; }
#footer { clear:both; background-color: Silver; }
```

Рис. 1.10. Таблица стилей DivStyle.css

Язык сценариев JavaScript

Языки сценариев позволяют включать в web-страницы программный код, который задает поведение web страницы в браузере при работе с ними пользователей. Существуют разные языки сценариев, но наиболее распространенным является язык JavaScript. JavaScript это интерпретируемый язык программирования, стандартизированный международной организацией ECMA в спецификации ECMA-262. Языки JavaScript, Jscript и ActionScript являются расширением стандарта ECMA-262.

Возможны 3 варианта встраивания кода JavaScript в web-страницы:

1. ***Расположение внутри страницы***. Для добавления JavaScript-кода на страницу, можно использовать тэги <script></script> . Например:

```
<script type="text/javascript"> alert('Hello, World!'); </script>
```

2. ***Расположение внутри тэга***. Спецификация HTML описывает набор атрибутов, используемых для задания обработчиков событий. Пример использования:

```
<a href="delete.php" onclick="return confirm('Вы уверены?');">Удалить</a>
```

3. ***Вынесение в отдельный файл***. Можно сохранить сценарий в отдельном файле, а потом подключить его с помощью конструкции

```
<script type="text/javascript" src="URL_файла_со_сценарием"></script>
```

Основы языка JavaScript

С помощью JavaScript описываются сценарии или скрипты, которые могут выполняться при помощи специальных модулей – ***интерпретаторов скриптов***, которые включены во все современные web-браузеры.

Код на языке JavaScript пишется в текстовом формате, и состоит из *инструкций*, *блоков*, включающих наборы инструкций и *комментариев*. В инструкциях могут использоваться *переменные* и *данные*, такие как *строки, числа* и *выражения*. Для указания *конца инструкции* используется символ ‘;’ (точка с запятой). *Блоком* является набор инструкций JavaScript, заключенных в фигурные скобки { }.

Комментарием в JavaScript является текст, расположенный после // до конца строки. *Многострочный комментарий* начинается с /*, и заканчивается */.

Язык JavaScript не поддерживает строгий контроль типов, поэтому не требуется явно объявлять тип переменных. Во многих случаях JavaScript выполняет преобразования типов данных автоматически, когда они необходимы. Например, при сложении строки и числа, число будет преобразовываться в строку.

Для объявления переменных может использоваться инструкция var. Инструкция var является обязательной при объявлении локальных переменных внутри функции. Разрешается выполнять объявление переменной неявно – без инструкции var. Однако, не разрешается применять необъявленные переменные в выражениях. В языке JavaScript различаются большие и прописные символы в именах переменных. Т.е., имена Name и name считаются различными.

Типы данных. Переменные в JavaScript не имеют строго фиксированного типа. *Переменные* имеют *тип*, эквивалентный типу значения, которое они содержат. Однако, в некоторых случаях, необходимо принудительное преобразование переменной в определенный тип. *Числа* могут быть объявлены как *строки*, а *строки* необходимо преобразовать в *числовой* тип. Для этого применяют функции parseInt() и parseFloat().

В языке JavaScript используется шесть типов *данных – числа, строки, объекты, логический*, null и undefined:

- JavaScript поддерживает *числа*, как *целые*, так и с *плавающей запятой*. Также существуют специальные представления чисел, например NaN (не число). Примеры чисел:

```
3.14 // Вещественное число
15 // Целое число
0177 // Восьмеричное число 177
0XA8 // Шестнадцатеричное число A8
```

- *Строки* объявляются при помощи *двойных кавычек* или *апострофов*. Строка может состоять из нескольких символов в формате Unicode.
- *Логический* тип допускает значения – true и false. Любое выражение, равное 0, считается эквивалентным false, а любое выражение, равное числу, отличному от 0 будет эквивалентным true.
- Переменная типа null – не имеет никакого определенного значения.
- Undefined – означает, что тип не определен. Значение undefined имеет переменная после ее объявления и до присвоения ей какого-либо определенного значения.

Выражения. В языке JavaScript выделяют *логические* или *числовые* выражения. Любая допустимая комбинация констант и переменных, объединенных знаками операций, является *выражением*.

Знак равенства (=) используется как *присваивание*. Например, выражение Pi = 3.14; подразумевает «Присвоить значение 3.14 переменной Pi». А для задания операции сравнения двух значений на равенство применяется двойной знак равенства (==).

Символ «+» в выражении означает «сложение» или «объединение строк».

Операторы. Язык поддерживает *условные операторы* if и if…else. Для объединения нескольких условий можно использовать логические операции || (ИЛИ) или && (И).

В JavaScript существует несколько видов циклов: for, for…in, while, do…while и switch. Также имеется оператор остановки выполнения цикла. break. Оператор continue используется для немедленного перехода к вы-

полнению следующей итерации, пропуская остальную часть выполнения кода текущей итерации.

Функции. В языке JavaScript функция выступает в качестве одного из основных типов данных. Имеется два вида *функций*: встроенные и определяемые. Программист имеет возможность создавать собственные функции. Определение функции состоит из *объявления параметров* и *блока инструкций* JavaScript. Перед тем как воспользоваться функцией, ее необходимо предварительно определить. Описание функции имеет следующий вид:

```
function имя (аргументы)
{
    операторы
}
```

Здесь *имя* – идентификатор, задающий имя функции, аргументы – необязательный список идентификаторов, разделенных запятыми, который содержит имена формальных аргументов функции, а **операторы** – любой набор операторов, который называется *телом* функции и исполняется при ее вызове. Например:

```
function area(radius)
{
    return  pi* radius * radius;
}
```

Данная функция имеет имя area и имеет один формальный аргумент radius. При вызове этой функции вместо формального аргумента подставляется его фактическое значение, функция вычисляет площадь круга на основе этого значения и возвращает полученное число с помощью оператора return.

Переменные, декларированные в теле функции, являются *локальными*, т. е. они доступны только в ее теле.

При вызове функции в JavaScript действуют следующие правила передачи аргументов:

- Аргументы примитивных типов передаются функции *по значению*.
- Объекты (и встроенные, и определенные пользователем) передаются *по ссылке*. Это означает, что все изменения свойств объекта в теле функции производятся непосредственно в самом объекте, а не в его локальной копии и, следовательно, сохраняются после возврата из функции.

Объекты – это главный тип данных в JavaScript. Объекты в JavaScript, являются наборами *методов* и *свойств*. Переменная любого

(отличного от объекта) типа данных, прежде чем к ней можно будет получить доступ, конвертируется в объект, и только после этого над ее значением выполняются действия. Тип данных Object сам может использоваться для объявления переменных.

В сценариях JavaScript могут использовать следующие виды объектов:

- *Объекты элементов документа*, входящие в модель DOM, т.е. соответствующие содержанию и поведению web-страницы, обрабатываемой браузером. Они создаются браузером при разборе (парсинге) HTML-страницы. Примеры: window, document, location, navigator и т.п.
- *Встроенные объекты*, представляющие различные типы данных, свойства, методы, присущие самому языку JavaScript, независимо от содержимого HTML-страницы. Например: Array, String, Date, Number, Function, Boolean, Math.
- *Пользовательские объекты*, которые создаются программистом в процессе создания сценария с использованием конструкторов типа объектов (класса).

Использование массивов

В скрипте JavaScript можно определять массивы (которые аналогичны коллекциям). Для работы с элементами массивов имеются различные методы: join(), reverse(), sort() и другие. Свойство массива length позволяет определить количество элементов в данном массиве.

Для объявления массивов (коллекции) существует специальный конструктор Array. Если ему передается единственный аргумент – целое неотрицательное число, то создается незаполненный массив соответствующей длины. Если же передается один аргумент, не являющийся числом, либо более одного аргумента, то создается массив, заполненный этими элементами:

```
a = new Array(); // пустой массив
b = new Array(17); // массив длины 17
c = new Array(10, '12345'); // массив из двух элементов: числа и строки
```

Нумерация элементов массивов начинается с нуля. В языке JavaScript массив может состоять из разнородных элементов. Элементами массива могут быть также и массивы.

Для перебора элементов массива используется оператор for (*переменная* in *объект*). Например:

```
for (d in document)
  document.write("document."+d+" = <b>"+ document[d]+"</b><br>");
```

Обработка *объектов* и *массивов* идентична. Можно обратиться к любой части объекта (его свойствам и методам) либо по *имени*, либо по *индексу*.

Объектная модель документа

Для работы с элементами web-страницы на клиентской стороне используется объектная модель документа (Document Object Model, DOM). Объектную модель позволяет связать между собой web-страницу и обрабатывающий ее браузер.

Смысл DOM модели состоит в том, что для каждого HTML-элемента создается соответствующий ему объект, который обладает своими свойствами, методами и событиями. Основным назначением DOM модели является возможность изменения HTML страницы (и ее отображения на экране) в сценариях JavaScript.

Связи между объектами различных уровней, представленные на рис. 1.11, означают, что объект верхнего уровня содержит ссылку на объект нижнего уровня. Так, например, между объектами Window и Document есть связь. Это означает, что объект Window имеет свойство с именем document, содержащее ссылку на объект типа Document.

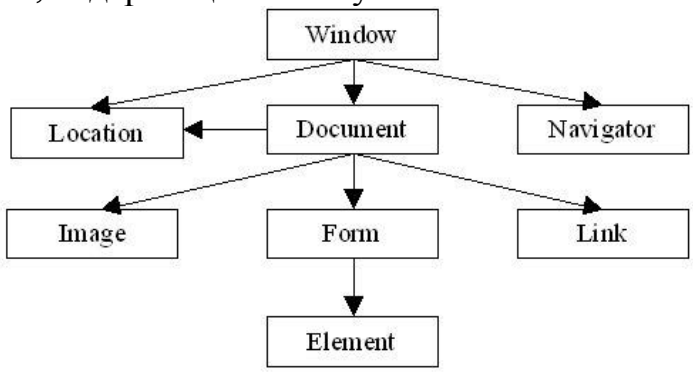


Рис. 1.11. Иерархия объектов в окне браузера

Для обращения к свойству или методу объекта, требуется использовать выражение следующего вида:

объект.свойство или объект.метод()

Основным объектом, который необходим для написания простейшего сценария является переменная document, ссылающаяся на объект Document, описывающий HTML документ, отображаемый в окне браузера.

Для того чтобы хранить произвольное количество ссылок на объекты, расположенные на web-странице, используются массивы. Поэтому свойства объекта Document, предназначенные для связи с компонентами web-страницы, выглядят следующим образом:

- links[ ] – массив ссылок на гипертекстовые связи, сформированные в HTML-документе с помощью элементов <a>;
- images[ ] – массив ссылок на изображения, включенные в состав web-страницы с помощью элемента <img>;
- forms[ ] – массив ссылок на формы, созданные посредством элементов <form>.

Ссылки на объекты Link, Image и Form содержатся в массивах links[], images[] и forms[] в том же порядке, в котором они встречаются в исходном тексте HTML-документа. Так, images[0] ссылается на первое изображение в составе web-страницы, images[1] – на второе изображение и т.д.

Объект Form, в свою очередь, ссылается на интерактивные элементы, принадлежащие форме. Несмотря на то, что каждый из элементов является объектом определенного типа (например, Text, Password, Radio, Checkbox и т.д.), для их описания существует также обобщенный тип Element. Соответственно ссылки на интерактивные элементы, принадлежащие форме, содержатся в свойстве elements объекта Form.

Например, чтобы получить строку текста, которую пользователь ввел в поле формы, расположенной на web-странице, можно использовать следующее выражение:

```
line = document.forms[0].elements[2].value;
```

В результате выполнения данной команды, значение третьего интерактивного элемента, принадлежащего первой форме, входящей в состав текущего документа, будет скопировано в переменную line. Данная команда выполнится корректно лишь в том случае, если будет правильно указан индекс массива elements[]. Он должен соответствовать порядковому номеру элемента в составе формы.

Динамическое изменение HTML документов

Для работы с объектной моделью документа DOM чаще всего используются методы группы getElements, которые предназначены для быстрого поиска элементов HTML:

- Метод document.getElementById(id), используется для поиска элементов по значению атрибута id в элементах документа.
- Метод document.getElementsByTagName(tag) возвращает коллекцию всех элементов с определенным тэгом, и среди них можно искать нужный. Например, можно получить второй элемент (нумерация в массиве идет с нуля) с тэгом <p>: document.getElementsByTagName('p')[1]
- Метод document.getElementsByName(name) возвращает все элементы, у которых имена (атрибут name) равны заданному значению. Он

работает только с теми элементами, для которых в спецификации явно предусмотрен атрибут name, например form, input, a, select, textarea и ряд других, более редких. Данный метод не будет работать с остальными элементами типа div, p и т.п.

- Метод getElementsByClassName() для поиска элементов по классу, но он работает не во всех браузерах, поэтому его редко используют.

После того, как ссылка на элемент будет найдена, можно менять его свойства и атрибуты. Например, следующий код изменит цвет текста на голубой (blue) в div контейнере с идентификатором dataKeeper:

```
document.getElementById('dataKeeper').style.color = 'blue'
```

Для изменения в скрипте содержания элементов HTML документа без перерисовки всей страницы нужно использовать свойство innerHTML объектов документа. Обычно данное свойство используется вместе с функцией getElementById() для получения ссылки на требуемый элемент документа.

```
document.getElementById('ID элемента').innerHTML = 'содержание';
```

Например, вывода сообщений в тэге <p> можно выполнить следующим образом:

```
<script type="text/javascript">
   function Msg1(){
      document.getElementById('myText').innerHTML = 'Спасибо!';
   }
   function Msg2(){
      document.getElementById('myText').innerHTML =
         'Попробуй получить сообщение 1 еще раз...';
   }
</script>
<input type="button" onclick="Msg1()" value="Показать сообщение 1" />
<input type="button" onclick="Msg2()" value="Показать сообщение 2" />
<p id="myText"></p>
```

Введенный пользователем текст можно вывести на экран в тэге с id="userMsg":

```
<script type="text/javascript">
        function showMsg(){
        var userInput = document.getElementById('userInput').value;
        document.getElementById('userMsg').innerHTML = userInput;
        }
</script>
<input type="input" maxlength="40" id="userInput" onkeyup="showMsg()"
        value="Введите текст здесь ..." />
```

```
<p id="userMsg"></p>
```

Обработка событий в JavaScript

Скрипты, составленные на языке JavaScript, могут вызываться в ответ на возникновение различных событий. Каждое такое событие связано с тем или иным объектом web-страницы: формой, гипертекстовой ссылкой или даже с окном, содержащим текущий документ. Примерами внешних событий, для которых могут вызываться скрипты JavaScript, являются следующие:

- Окончание загрузки документа в браузере. Это событие связано с объектом window.
- Щелчок кнопкой мыши на объекте. Это событие может быть связано с интерактивным элементом формы или с гипертекстовой ссылкой.
- Получение объектом фокуса ввода. Это событие может быть связано с объектами типа Text, Password и с другими интерактивными элементами.
- Передача на сервер данных, введенных пользователем с помощью интерактивных элементов.

Обработка события производится с помощью специально предназначенной для этого функцией, называемой *обработчиком события*. Для каждого события JavaScript может быть задан свой обработчик.

Для связывания события с обработчиком используются атрибуты тэга, описывающего объект, с которым связано событие. Задание обработчика для события выполняется следующим образом:

```
on[имя_события]="команды_обработчика"
```

Например, для того чтобы сценарий реагировал на щелчок мышью, используется обработчик с именем onClick, для обработки события, заключающегося в получении фокуса ввода, – обработчик onFocus.

Если необходимо обработать событие, заключающееся в получении фокуса полем ввода, то тэг, описывающий этот элемент управления, должен иметь примерно следующий вид:

```
<input type="text" name="Inform" onFocus="handleFocus();">
```

В данном случае обработка события производится с помощью функции handleFocus(). В принципе, обработчиком может быть не только функция, но и любая последовательность команд JavaScript в виде составного оператора. Следующий пример демонстрирует обработку события, связанного с наведением курсора мыши на гиперссылку:

```
<a href = "http://www.myhp.edu" onMouseOver="alert('Событие onMouseOver');
```

```
        return false">
    <img src="my.gif" width="50" height="50" />
</a>
```

Обработчик события можно назначить для некоторого элемента страницы и в коде скрипта. Например:

```
<div id='test123'>Нажми меня!</div>

<script><!--
document.getElementById('test123').onclick = function()
{
   alert(this.id);
};--></script>
```

2. Программное обеспечение Web сети

Работа web-сети, также, как и работа других Интернет сервисов, реализуется по технологии «клиент-сервер». В качестве клиентов в web-сети используются программы – ***web-браузеры*** (browser, “бродилка”, web-обозреватель, навигатор). А в качестве серверов – ***web-серверы*** (HTTP-серверы), которые принимает от браузеров HTTP-запросы, выполняют их обработку и отправляют HTTP-ответы. Клиент, которым обычно является web-браузер, передаёт web-серверу запросы по протоколу HTTP (HTTP-запрос) на получение ресурсов, заданных URL-адресами. Ресурсами могут быть HTML-страницы, графические файлы (изображения), медиа-потоки или другие данные, которые необходимы клиенту. В ответ web-сервер передаёт клиенту HTTP-ответ, включающий запрошенные данные.

2.1. Web-браузеры

Web-браузеры это клиентское программное обеспечение (с которым работают пользователи), которое позволяет пользователям выполнять запросы к web-серверам по протоколу HTTP, получать от них ответы, содержащие обычно HTML документы, анализировать их и показывать пользователям в удобной для них форме. Браузер позволяет выполнять следующие основные операции:

- поддержка взаимодействия с использованием разных протоколов (HTTP, FTP);
- отображение содержания на экране разных типов ресурсов (HTML документов; изображений; мультимедиа документов и т.п.);

- выполнение переходов по ссылкам, которые содержатся в web-страницах путем формирования HTTP запросов к web-серверам;
- выполнение скриптов, содержащихся в web-страницах (обычно JavaScript) при возникновении требуемых событий;
- ввод данных пользователя с помощью форм (Form) и их передачу web-серверу с помощью методов GET или POST.

Практически все популярные браузеры распространяются бесплатно. По данным компании «Net Applications» в августе 2010г. наиболее популярными браузерами являлись следующие: Microsoft Internet Explorer – 60.40%, Firefox – 22.93%, Chrome – 7.52%, Safari – 5.16%, Opera – 2.37% (процентами показана доля всех компьютеров в мире, на которых установлен данный тип браузера).

Функции браузеров

Основными функциями браузеров являются:

- Формирование и передача запросов web серверам от имени пользователей, в результате: перехода по гиперссылкам; явного ввода URL адреса; отправки данных формы (нажатие клавиши типа submit); анализа HTML страниц, которые требуют дополнительных ресурсов (например, изображений, аудио-файлов и т.п.).
- Получение ответов от web серверов и их интерпретация для создания визуального представления для пользователя. В самом простом случае это требует проверки некоторых заголовков ответа, таких, как Content-Type, для определения того, какие действия требуется выполнить и какой способ визуализации (рэндеринга) требуется.
- Визуализация полученных данных в окне браузера или с помощью программ, в зависимости от типа контента в HTTP ответе.

Кроме этих функций браузер выполняет и много других функций, в зависимости от значений кода состояния и заголовков ответов:

- ***Кэширование:*** Каждый браузер выполняет временное хранение копии ресурсов, получаемых от web-серверов (поддерживает локальный кэш).
- ***Аутентификация и авторизация:*** web сервер может затребовать авторизацию для получения ресурса, если для него были заданы соответствующие параметры безопасности. В этом случае браузер будет запрашивать данные аутентификации (имя и пароль) у пользователя или использовать уже ранее введенные данные и отправлять их web серверу.
- ***Поддержка состояния:*** Для записи и поддержки состояния между запросами и ответами web сервер может присылать браузеру куки в заголовках ответов. Браузер должен сохранить информацию, со-

держащуюся в куки и возвращать ее серверу в заголовках последующих запросов.

- ***Запрашивание поддерживающих элементов данных:*** Обычная web страница содержит ссылки на изображения, Java апплеты, мультимедиа файлы и другие вспомогательные ресурсы. В этом случае, для правильного отображения такой страницы браузер должен вначале получить их от сервера (т.е. сделать соответствующие запросы). Это выполняется без всякого участия пользователя.
- ***Выполнение действий в ответ на другие заголовки и коды состояния:*** HTTP заголовки и коды состояний могут предоставлять дополнительные инструкции обработки. Такие инструкции могут сообщать о проблеме доступа к ресурсу или могут указывать браузеру перенаправить запрос на другой адрес. Они также могут информировать браузер не прерывать текущее соединение (оно должно оставаться открытым), чтобы последующие запросы отправлялись с использованием того же самого соединения.
- ***Визуализация сложных объектов***: Большинство браузеров поддерживают работу с такими типами содержания, как text/html, text/plain, image/gif и image/jpeg. Это означает, что браузер включает функциональность для отображения такого содержания в своем окне, без необходимости устанавливать дополнительное программное обеспечение. Для отображения или проигрывания других, более сложных объектов (таких, как аудио, видео и мультимедиа), браузер должен предоставлять поддержку для их типов содержания. Должны иметься возможности для вызова внешних вспомогательных приложений или встроенных плагинов (plug-ins), которые требуются для отображения и проигрывания таких объектов.
- ***Обработка ошибочных состояний***: Браузеры должны быть иметь средства для обработки ошибок соединения, не правильных ответов серверов и других аналогичных ситуаций.

Архитектура браузеров

В общем виде браузеры можно представить, как набор взаимодействующих между собой модулей, совместная работа которых позволяет им выполнять все требуемые функции.

- **Модуль пользовательского интерфейса**: Данный модуль является ответственным за предоставление интерфейса, с помощью которого пользователи могут взаимодействовать с web приложениями. Он включает средства преставления и отображения (рендеринга) конечного результата обработки браузером переданных web-серверами ответов.

- **Модуль формирования HTTP запросов**: Данный модуль отвечает за решение задачи формирования HTTP запросов и их отправку HTTP серверам. При получении запроса от модуля пользовательского интерфейса или модуля интерпретации содержания на формирование запроса на основе относительных ссылок, данный модуль вначале должен выполнить их преобразование в абсолютные URL адреса (выполнить разрешение адресов).
- **Модуль обработки HTTP ответов**: Данный модуль отвечает за разбор HTTP ответов, их интерпретацию и передает полученное в ответе содержание модулю пользовательского интерфейса.
- **Модуль поддержки работы в сети**: Данный модуль является ответственным за передачу данных по сети. Он принимает сформированные запросы от модуля формирования HTTP запросов и выполняет их передачу по сети соответствующему web или прокси-серверу. Также данный модуль принимает запросы, поступающие по сети, и передает их модулю обработки HTTP ответов. В процессе выполнения этих работ он отвечает за установление сетевых связей между сокетами и работу с прокси-серверами, которые могут быть заданы в параметрах конфигурации сети пользователя.
- **Модуль интерпретация контента**: При получении ответа, модулю обработки HTTP ответов требуется помощь в грамматическом разборе и декодировании полученного содержания. Начальные запросы часто имеют в качестве типа содержания text/html, но в данное содержание могут быть включены ссылки на изображения, мультимедиа объекты, JavaScript код и информацию таблиц стилей. Модуль интерпретация контента выполняет дополнительную обработку необходимую для того, чтобы браузер мог понять и обработать такие ссылки, запрашивая модуль формирования запросов составлять дополнительные запросы для поиска вспомогательных объектов.
- **Модуль кэширования**: Web браузеры должны стремиться уменьшить количество запрашиваемых ресурсов. Для этого, они поддерживают в дисковой памяти временное хранилище (кэш) для сохранения копий ранее полученных ресурсов. Как раз данный модуль и отвечает за сохранение копий ресурсов в кэше и их последующее использование, а также за управление памятью (как оперативной, так и файловой) выделенной в соответствии с параметрами конфигурации браузера.
- **Модуль поддержки состояния**: Так как HTTP является протоколом без поддержки состояния, то требуются некоторые дополнительные возможности для выполнения такой поддержи, между выполнениями взаимосвязанных запросов и ответов. Одним из средств реше-

ния данной задачи на строне браузера являются куки. Данный модуль отвечает за сохранение присланной в куки информации и включения ее во вновь формируемые запросы.

- **Модуль аутентификация/авторизация**: Данный модуль отвечает за предоставление информации аутентификации пользователя, если ее запрашивает web-сервер. Он должен обрабатывать заголовки ответов, в которых запрашиваются данные аутентификации пользователя путем запрашивания требуемой информации у пользователя (обычно с помощью специального диалогового окна). Он также должен сохранять эти данные на тот случай, если они потребуются для доступа к другому защищенному ресурсу в той же самой «области» безопасности. Это освобождает пользователя от необходимости многократного ввода таких данных для каждого запрашиваемого ресурса.
- **Модуль конфигурирование**: Для работы браузера требуются большое количество параметров конфигурации, которые должно поддерживаться. Некоторые такие параметры являются фиксированными, а другие могут задаваться пользователями. Данный модуль поддерживает фиксированные и изменяемые параметры конфигурации и предоставляет пользовательский интерфейс для их изменения.

2.2. Web-серверы

Web-сервер (или HTTP-сервер) это серверная программа, работающая в фоновом режиме, ожидающая запросы пользователей и выполняющая их обработку. Web-сервер принимает HTTP-запросы от клиентов, обычно web-браузеров, и возвращает им HTTP-ответы, обычно вместе с web-страницами (HTML-документами). Web-серверы составляют основу Web сети. Наиболее используемые в 2010 году web-серверы описаны в табл. 2.1.

Таблица 2.1

Список наиболее известных web-серверов

Производитель	**Система**	**Кол-во поддерживаемых web сайтов**	**Процент**
Apache	Apache	119,664,128	56,06%
Microsoft	Internet Information Services (IIS)	53,434,586	25.03%
Google	Google Web Server (GWS)	15,526,781	7.27%
Игорь Сысоев	engine x (nginx)	11,713,607	5.49%
Lighttpd	lighttpd	1,821,824	0.85%

Описание работы web-сервера

Web серверы и браузеры обмениваются между собой HTTP сообщениями. Серверы получают и обрабатывают HTTP запросы, определяют местоположение и выполняют доступ к запрашиваемым ресурсам, и формируют ответы, которые они отправляют назад браузерам, сделавшим эти запросы. На рис. 2.1 показано, как web-сервер обрабатывает поступающие запросы, формирует на них ответы и передает их запрашивающему браузеру.

Модуль поддержки работы с сетью отвечает за получение запросов и передачу сформированных ответов по сети. При получении очередного HTTP запроса сервер, прежде всего, передает его **модулю разрешения запроса**, который отвечает за анализ и предварительную обработку поступившего запроса. Предварительная обработка запроса включает:

1. **Виртуальный хостинг:** если web сервер предоставляет доступ к нескольким web сайтам, имеющим разные доменные имена, то для поступившего запроса требуется определить целевой web сайт и использовать его для выбора параметров конфигурации.

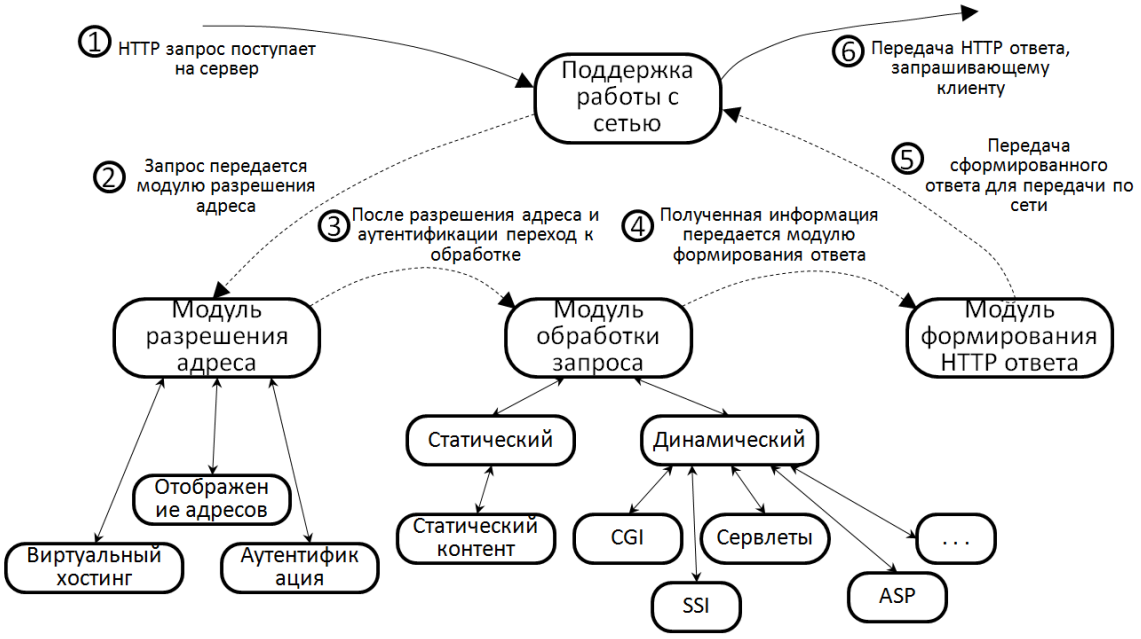


Рис. 2.1. Обработка сервером HTTP запросов

2. **Разрешение адреса:** определение типа запрашиваемого контента: статический или динамический; на основе заданного URL пути и выбранных параметров конфигурации сервера, выполняется разрешение URL адреса (т.е. его преобразование) в реальный адрес в файловой системе сервера.

3. **Аутентификация:** если запрашиваемый ресурс является защищенным, то требуется проверить данные авторизации (имя и пароль), чтобы определить имеет право пользователь использовать данный ресурс.

После завершения предварительной обработки запрос передается модулю обработки запроса, который вызывает подмодули для выполнения соответствующей обработки статического и динамического контента. Если запрос обращается к динамическому контенту, то сервер передает данные на выполнение некоторый среде: контейнеру сервлетов или серверу приложений, которые выполняются в других процессах.

Данная среда управляет выполнением web-приложений:

- организует взаимодействием web сервера с приложением (передает данные сообщения и параметры сервера приложению и возвращает сформированный код серверу);
- запускает требуемое приложение на выполнение;
- решает дополнительные задачи (управление состоянием сеанса работы, поддержка очереди сообщений, управление кэшем и т.п.).

Когда выбранный подмодуль или сервер приложений закончит обработку запроса, он передаст результаты выполнения модулю формирования HTTP ответа, который формирует заголовки ответа, объединяет их с полученным результатом обработки и передает модулю поддержки работы с сетью для передачи сформированного ответа тому клиенту, который прислал данный запрос. Следует отметить, что в связи с тем, что HTTP не поддерживает состояние сеанса работы, то единственная информация, которая доступна серверу о поступившем запросе, содержится в самом запросе (заголовках и теле).

Состояние сеанса работы в форме информации, связанной с сеансом, может поддерживаться web-приложениями или средами, в которых они выполняются (например, контейнерами сервлетов), но ссылка на эту информацию должна содержаться в самом запросе (например, с помощью куки или специального URL адреса).

Кроме этого, web-серверы могут выполнять различные дополнительные функции, например:

- ведение журнала обращений пользователей к ресурсам;
- выполнение аутентификации и авторизации пользователей;
- поддержка HTTPS для защищённых соединений с клиентами.

Web-серверы могут предоставлять, как статический, так и динамический контент. Под статическим контентом понимаются файлы, такие, как HTML-страницы, XML-документы, простой текст, изображе-

ния, аудио-видео файлы, которые передаются без изменения в теле HTTP ответов сервера. Для динамического контента, сервер должен запустить на выполнение некоторую программу, формирующую данные, которые будут помещаться в тело HTTP ответа.

Web серверы используют комбинацию расширений запрашиваемых файлов, URL префиксов и файлы конфигурации для определения того, какой способ обработки должен быть использован, чтобы сформировать ответ. По умолчанию предполагается, что URL должен обрабатываться, как запрос к статической web странице.

При установке серверу выделяется каталог (папка), с которой он может работать. Например, для сервера IIS (Inernet Information Services операционной системы Windows) по умолчанию создается папка с именем inetpub, в которой хранятся ресурсы, предоставляемые web-сервером. Кроме физических папок, расположенных в данном месте, сервер может использовать так называемые виртуальные папки. Виртуальные папки описаны в файлах конфигурации web сервера, а располагаться могут в разных местах внешней памяти компьютера, на котором установлен сервер, или в локальной сети сервера (с заданием прав доступа к этим папкам).

Конфигурирование сервера

Поведение web сервера во много задается его установочными данными (конфигурацией). В то время, как детали конфигурирования web сервера в значительной степени отличаются между разными реализациями web-серверов, имеются некоторые общие понятия, которые используются везде. Например, HTTP сервер должен связывать (map) расширения файлов с MIME типами и любой сервер должен выполнять разрешение (связывание) URL адресов с адресами в локальной файловой системе сервера.

Настройка работы сервера выполняется с использованием файлов конфигурации. Обычно это XML файлы, в котором задаются параметры работы сервера:

- используемые расширения сервера;
- поддерживаемые сервером web-сайты;
- виртуальные папки сервера;
- описание web-приложения;
- и т.д.

Например, описание параметров сервера IIS содержится в XML-файлах, в которых указаны настройки web-сервера, web-сайтов и web-приложений. Основным файлом конфигурации IIS 7 является Applica-

tionHost.config (в папке %windir%\system32\inetsrv\config). Для web-приложений и каталогов в службах IIS 7 также используются файлы Web.config.

Web сервер работает в фоновом режиме (без использования графического интерфейса), а для управления его работой и настройки параметров используется консоль управления «Диспетчер служб IIS», который можно вызвать с помощью диалоговых окон «Панель управления=>Администрирование =>Диспетчер служб IIS».

2.3. Web-приложения

Под Интернет приложениями[1] обычно понимаются web-приложения. ***Web-приложение*** – это приложение разработанное по архитектуре «клиент-сервер», использующее в качестве клиента web-браузер и работающее с использованием протокола HTTP на стороне web-сервера (рис. 2.2).

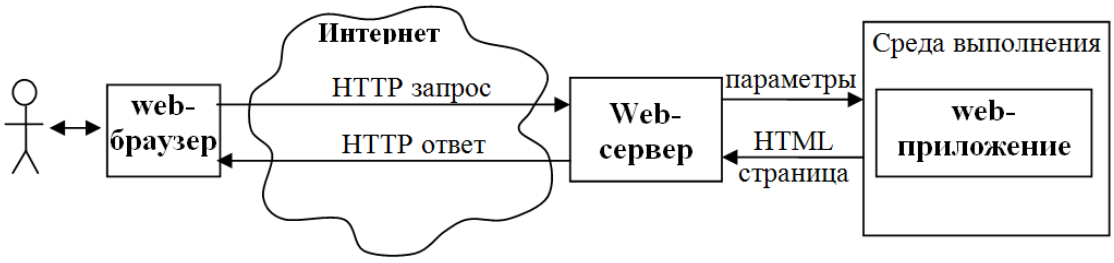


Рис. 2.2. Общая схема взаимодействия пользователя с web-приложением

Пользователь с помощью браузер отправляет HTTP-запрос по определенному URL адресу, который сопоставляются с ресурсами на web-сервере. Если данный URL адрес указывает на динамический ресурс (web-приложение), то сервер запускает некоторую внешнюю программу (web приложение), формирующую HTML-страницу, которая может быть показана браузером, и возвращает ее клиенту. Основной частью web-приложения является его логика на стороне web-сервера.

Чаще всего, web приложение не является самостоятельной программой (кроме технологии CGI), а выполняется под управлением некоторой внешней программы – ***среды выполнения***. В качестве такой программы могут выступать: сервер приложений или контейнер приложений. Среда выполнения делает следующие действия:

[1] А не просто программы, которые используют в своей работе сеть Интернет (обмен сообщениями, получение обновлений и т.п.).

- принимает от web-сервера всю информацию, связанную с HTTP запросом;
- определяет, какое приложение, созданное для данной среды, должно быть выполнено;
- создает все вспомогательные объекты, которые могут потребоваться для работы данного приложения (такие, как например, Application, Session, Request, Response, User и т.п.);
- загружает запрашиваемое приложений и передает ему управление;
- web приложение выполняется и создает HTML страницу, которую записывает в специально созданный для этого объект (обычно, объект Response) и после этого возвращает управление web-серверу;
- web-сервер формирует HTTP ответ, включает в него сформированную web-приложением HTML-страницу и отправляет его клиенту, который прислал запрос к данному web-приложению.

Обычно HTML страницы, созданные web-приложением, включают HTML формы, с которыми работает пользователь. При нажатии пользователями на кнопки типа submit браузер отправляет новый HTTP запрос к тому же самому web-приложению. Последовательность вызовов web-приложения пользователем составляют ***сеанс*** его работы. Данный сеанса завершается либо в результате выбора пользователем команды об окончании его работы с web-приложением, либо в результате длительного промежутка времени между отправками последующих HTTP запросов (обычно, более чем 20 минут).

Отметим, что web-приложение предназначено для решения некоторых задач пользователей и обязательно имеет пользовательский графический интерфейс.

На основе анализа современного содержания web сети можно выделить следующие примеры типичных web-приложений:

- Поисковые системы (например, google.ru, rambler.ru, yahoo.ru)
- Коллективно заполняемые документы: открытые энциклопедии (например, wikipedia.com/wikipedia.ru), словари (http://en.wiktionary.org).
- Видео-коллекции (например, youtube.com).
- Новостные системы (например, lenta.ru, newsru.ru).
- Масс-медиа: сайты радиостанций (например, echo.msk.ru); телеканалов; газет (например, kp.ru) и журналов.
- Бизнес: магазины (amazon.com, ozon.ru); банки; заказы услуг.
- Социальные сети (например: facebook.com; odnoklassniki.ru http://vkontakte.ru/).

- Прогноз погоды (например: www.gismeteo.ru, pogoda.vtomske.ru).
- Блоги (web дневники) (например, http://www.livejournal.ru/, jj.ru).
- Хостинги файлов (позволяют хранить файлы пользователей) (например, Disk.yes.ru, Pipebytes.com).
- Обработка изображений (например, Pixlr.com (онлайн-фотошоп), Picmeleo.com).
- Работа с музыкой (например, Mp3skip.com (поиск и прослушивание MP3), Mp3cut.ru (быстрая нарезка аудиофайлов)).
- Ведение бухгалтерии (например, Moedelo.org (онлайн-бухгалтерия для ООО и ИП), Zenmoney.ru (управление личными финансами)).
- Планировщики работ (например, Miniplan.ru (онлайн-органайзер), Getfriday.ru (личный онлайн-помощник), Reminday.com (сервис напоминаний о днях рождения).
- Покупка билетовок на транспорт (Skyscanner.ru (поиск недорогих авиабилетов), rzd.ru (покупка железнодорожных билетов), Seatguru.com (информация о хороших местах в самолете)).

Достоинства и недостатки web приложений

Web-приложения в сравнении с локальными приложениями имеют следующие преимущества:

- Простота доступа к приложению. Любой человек, имеющий компьютер, подключенный к сети Интернет, может использовать web приложение.
- Простота развертывания (установки). В отличие от локальных приложений, web приложения после завершения разработки не требуется устанавливать на компьютерах пользователей. Достаточно только сообщить им URL адрес приложения. При изменении приложения все пользователи сразу начинают работать с измененной версией.
- Наличие большого количества обученных пользователей.
- Высокий уровень развития и надежности сетевых соединений и web технологий.

Однако web-приложения имеют и свои недостатки:

- ***«Слабо-связанная» архитектура web-сети***: отсутствие поддержки состояния сеанса работы и задержка при перезагрузке каждой страницы. Каждая перегрузка или обновление страницы вызывает заметную задержку, вызванную необходимостью установить HTTP со-

единение, обработать запрос на сервере, передать по сети ответное HTTP сообщение и перегрузить страницу браузером. Это создает скачки и прерывистый режим работы пользователя.

- ***Ограниченный набор элементов управления для проектирования форм приложения***. Текущая версия языка HTML поддерживает только ограниченный набор элементов управления (текстовые элементы, радио кнопки, флажки, выпадающие списки и командные кнопки). Язык не предлагает поддержку сложных взаимодействий, которые часто используются в настольных приложениях, такие, как календари, мастера, закладки, инструментальные полосы, контекстные меню и т.п., которые доступны в даже самых простых локальных приложениях.

Хотя такие ЭУ могут быть разработаны с помощью JavaScript и CSS, отсутствие встроенной поддержки браузеров ведет разнообразию реализаций с несогласованными представлениями и способами работы с ними.

Как базовая технологическая архитектура Web сети, так и ограниченный набор доступных элементов управления делает поддержку взаимодействия с web приложениями более сложной, в сравнении с локальными приложениями.

Кроме этого, так как большинство web приложений делаются независимыми от конкретного браузера, то взаимодействия и внешнее представление web приложения может быть не одинаковым в разных операционных системах.

- ***Несогласованные подходы к оформлению и способу взаимодействия пользователями***. Хотя Web сеть предоставляет разработчикам и дизайнерам значительную гибкость и возможность проявить свои творческие способности, получающаяся в результате этого разнообразие и несогласованность пользовательских интерфейсов и способов взаимодействия с web приложениями часто создает проблемы для пользователей. Это вызвано тем, что пользователи сталкиваются с большим разнообразием стилей визуальных интерфейсов и способов взаимодействия, каждый из которых предлагает свой словарь объектов, действий и визуальных представлений, смешанных в одном и том же приложении.

Для решения таких проблем проектирования (разработки) и связанных с ними проблем использования, многие компании (организации) разрабатывают руководства по проектированию пользовательского интерфейса и стиля для упорядочения оформления и работы web приложений.

- ***Открытость доступа к приложениям***. В связи с открытым доступом к web-приложениям практически для всех web приложений

требуется поддерживать безопасность их использования пользователями.

Структура web-приложения

Приложения обычно делятся на логические части, называемые «слоями», при этом, каждому слою назначается своя роль. Локальные приложения могут состоять только из одного слоя, который размещается на компьютере клиента, а web-приложения по своей природе следуют N-слойному подходу. Хотя возможны разные варианты, наиболее распространенными являются приложения, использующие три слоя: слой представление; слой бизнес-логика; слой доступ к данным (хранилище). Каждый слой включает набор компонент (наборов классов), выполняющих специальные функции.

На рис. 2.3 показана типовая архитектура web-приложения с набором наиболее часто применяемых компонент, сгруппированных по функциональным областям.

Слой представления обычно включает компоненты пользовательского интерфейса (UI) и логику представления. Слой бизнес-логики включает компоненты бизнес-логики, бизнес-процесса и бизнес-сущностей. Слой доступа к данным включает компоненты, реализующие доступ к требуемым данным и web-сервисам.

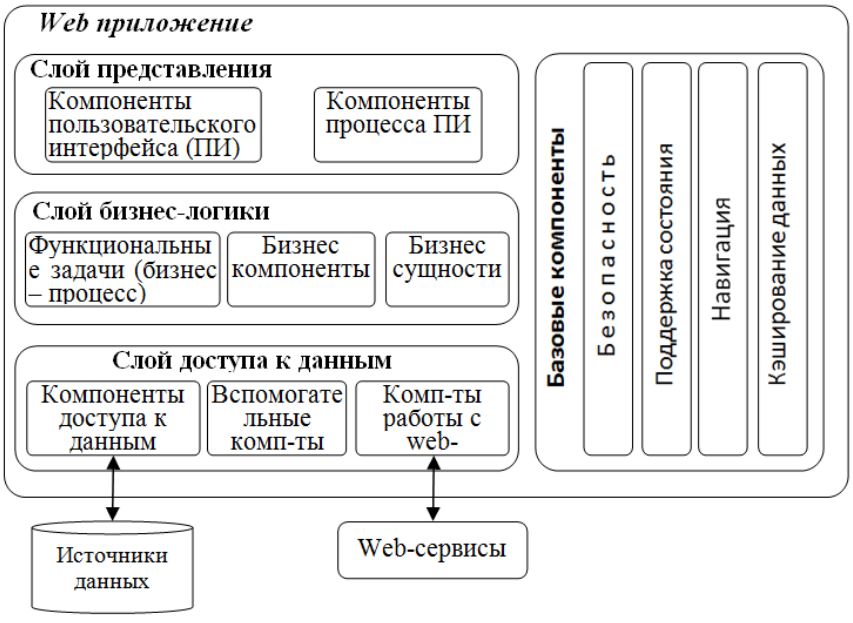


Рис.2.3. Типовая структура web-приложения

Web приложение также можно разделить на базовые и функциональные подсистемы. К базовым подсистемам web-приложения относятся:

- Единообразное оформление web-страниц, составляющих приложение.
- Поддержка состояния сеанса работы пользователей.
- Настройка web-страниц для разных пользователей (персонализация).
- Навигация между разными web-страницами.
- Обеспечение безопасности (аутентификация и авторизация, регистрация пользователей).
- Доступ к источникам данных.

А к функциональным подсистемам относятся:

- Управление контентом web-приложения (множеством документов, изображений, ссылок и т.п., которые доступны данному приложению). Web приложение должно предоставлять возможности загрузки контента на сервер (uploading), классификации, поиска и получения с сервера (downloading).
- Поиск информации в контенте приложения (в документах, файлах данных, базах данных).
- Поддержка взаимодействия пользователей (форумы, обмен сообщениями, рецензирование документов).
- Специальные возможности (расчеты в соответствии с бизнес-логикой, обработка изображений, обработка документов и т.п.).

Базовая функциональность во многом реализуется средой выполнения web-приложения, а для создания функциональных подсистем требуется составление программного кода.

2.4. Web-сервисы

Кроме web-приложений другим важным видом используемого в web-сети программного обеспечения являются ***web-сервисы*** (web services).

Web-сервисы представляют собой наборы методов, которые можно вызывать на выполнение с заданными параметрами с помощью HTTP запросов, а результаты их выполнения получать в виде HTTP ответов. Web сервисы поддерживаются, как и web-приложения, web-серверами. Они предназначены для использования любыми программами, которые могут формировать правильные HTTP запросы и понимать полученные HTTP ответы, но не пользователями (людьми), так как они не имеют графического пользовательского интерфейса. Такими программами могут быть web-приложения, либо другие web-сервисами, или даже локальными приложениями.

Web-сервисы, также, как и web-приложения, полностью реализуются с использованием технологий и стандартов web-сети. Они имеют URL адреса, с помощью которого к ним можно обращаться. Для передачи запроса на выполнение методов web-сервисов и возврата результатов используется язык XML. На рис. 2.4 показана общая схема взаимодействия web-приложения с web-сервисом.

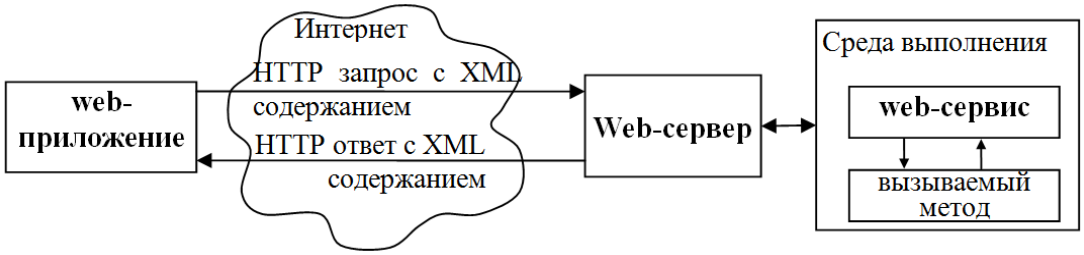


Рис. 2.4. Общая схема взаимодействия web-приложений с web-сервисами

Работа web-сервисов основывается на использовании данных, описанных на языке XML. Такие XML данные должны составляться в соответствии с простым протоколом доступа к объектам – SOAP (Simple Object Access Protocol). Данный протокол предоставляет стандартный и простой способ описания сообщений в формате XML (формат передачи данных). Он позволяет разработчикам описывать, что может выполнить сервис и делает предоставляемую функциональность доступной другим приложениям.

Примером очень простого web-сервиса может быть сервис Math, который включает две операции: сложение (Add) и вычитание (Subtract):

```
public int Add(int x, int y) { return x + y;}
public void Subtract(int x, int y, out int z) { z = x-y; return z;}
```

XML данные сформированные в соответствии с протоколом SOAP в качестве корневого элемента используют элемент «Envelope», в который включены два подэлемента «Header» (не обязательный) и «Body». В элементе «Header» описывается общая информация, связанная с запросом или ответом. А элемент «Body» в запросе содержит информацию о вызываемом методе (например: Subtract) и передаваемых ему параметрах (x и y). Составленный таким образом XML документ передается с помощью протоколов HTTP или SMTP

При использовании протокола HTTP, данный XML документ (Envelope) включается в тело HTTP запроса, которое следует через пустую строку после строки запроса (например, POST /services HTTP/1.1) и соответствующих заголовков, как показано на рис. 2.5.

```
POST   /MyWebSite/MathService.asmx      HTTP/1.1
Host: localhost
Content-Type: application/soap+xml; charset=utf-8
Content-Length: …

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
    <soap12:Body>
        <Subtract xmlns="http://tempuri.org/">
            <x>7</x>
            <y>3</y>
        </Subtract>
    </soap12:Body>
</soap12:Envelope>
```

Рис. 2.5. Пример HTTP запроса, включающего SOAP запрос к web-сервису

После получения данного запроса web-сервер передает управление среде выполнения, которая вызывает требуемый метод и передает ему полученные параметры. После выполнения данного метода формируется XML документ, содержащий результаты работы метода, который оформляется в соответствии с протоколом SOAP.

На рис. 2.6, показан HTTP ответ, содержащий результаты выполнения web-сервиса (Subtract), который выполняет вычитание значений первых двух параметров и возвращает результат в третьем параметре. Так как эти данные передаются в HTTP ответе, то они включаются после строки состояния и соответствующих заголовков.

```
HTTP/1.1   200   OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: …

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
    <soap12:Body>
        <SubtractResponse xmlns="http://tempuri.org/">
            <z>4</z>
        </SubtractResponse>
    </soap12:Body>
</soap12:Envelope>
```

Рис. 2.6. Пример HTTP ответа web-сервиса

Для того, чтобы пользователи сервисов (web-приложения) могли правильно составлять запросы и понимать ответы, web-сервисы также являются ответственными за предоставление метаданных, описываю-

щих сообщения, которые они могут принимать и создавать. Такие описания метаданных составляются с помощью специального языка описания web-сервисов WSDL (Web Services Definition Language), использующий XML формат.

Каждый web-сервис по специальному HTTP запросу предоставляет WSDL документ, в котором описывается все, что клиенту необходимо для организации работы с этим сервисом. WSDL-документ позволяет разработчикам описывать синтаксис вызова любого web-метода. Более того, этот документ позволяет использовать инструменты автоматического генерирования специальных программных средств (прокси-классов), которые делают вызов методов web-сервиса такими же простыми, как и применение объектов локальных классов.

Пример описания web-сервиса Math показан на рис. 2.7.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        targetNamespace="http://tempuri.org/"
        xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
        xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
        xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://tempuri.org/"
        xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
        xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
<wsdl:types>
   <s:schema targetNamespace="http://tempuri.org/" elementFormDefault="qualified">
   <s:element name="Add"> <s:complexType> <s:sequence>
      <s:element name="x" type="s:int" maxOccurs="1" minOccurs="1"/>
      <s:element name="y" type="s:int" maxOccurs="1" minOccurs="1"/>
   </s:sequence> </s:complexType> </s:element>
   <s:element name="AddResponse"> <s:complexType> <s:sequence>
      <s:element name="AddResult" type="s:int" maxOccurs="1" minOccurs="1"/>
   </s:sequence> </s:complexType> </s:element>
   <s:element name="Subtract"> <s:complexType> <s:sequence>
      <s:element name="x" type="s:int" maxOccurs="1" minOccurs="1"/>
      <s:element name="y" type="s:int" maxOccurs="1" minOccurs="1"/>
   </s:sequence> </s:complexType> </s:element>
   <s:element name="SubtractResponse"> <s:complexType> <s:sequence>
      <s:element name="z" type="s:int" maxOccurs="1" minOccurs="1"/>
   </s:sequence> </s:complexType> </s:element> </s:schema>
</wsdl:types>
<wsdl:message name="AddSoapIn">
   <wsdl:part name="parameters" element="tns:Add"/>
</wsdl:message>
<wsdl:message name="AddSoapOut">
   <wsdl:part name="parameters" element="tns:AddResponse"/>
</wsdl:message>
<wsdl:message name="SubtractSoapIn">
   <wsdl:part name="parameters" element="tns:Subtract"/>
```

```
</wsdl:message>
<wsdl:message name="SubtractSoapOut">
    <wsdl:part name="parameters" element="tns:SubtractResponse"/>
</wsdl:message>
<wsdl:portType name="MathServiceSoap">
    <wsdl:operation name="Add">
        <wsdl:input message="tns:AddSoapIn"/>
        <wsdl:output message="tns:AddSoapOut"/>
    </wsdl:operation>
    <wsdl:operation name="Subtract">
        <wsdl:input message="tns:SubtractSoapIn"/>
        <wsdl:output message="tns:SubtractSoapOut"/>
    </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="MathServiceSoap12" type="tns:MathServiceSoap">
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="Add">
        <soap12:operation style="document" soapAction="http://tempuri.org/Add"/>
        <wsdl:input> <soap12:body use="literal"/> </wsdl:input>
        <wsdl:output> <soap12:body use="literal"/> </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="Subtract">
        <soap12:operation style="document" soapAction="http://tempuri.org/Subtract"/>
        <wsdl:input> <soap12:body use="literal"/> </wsdl:input>
        <wsdl:output> <soap12:body use="literal"/> </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
<wsdl:service name="MathService">
    <wsdl:port name="MathServiceSoap12" binding="tns:MathServiceSoap12">
        <soap12:address location="http://localhost:61269/MyWebSite/MathService.asmx"/>
    </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

Рис. 2.7. Пример WSDL описания web-сервиса

В данном документе содержится описания различных элементов, составляющих web-сервис, таких, как: типы данных (<wsdl:types>), сообщения (<wsdl:message>), типы портов (<wsdl:port>), связывания (<wsdl:binding>) и сам сервис (<wsdl:service>). Такие описания обычно составляются и используются системами разработки и поэтому в данном пособии подробно не рассматриваются. Пример создания и использования web-сервиса с использованием технологии технологии ASP.Net рассмотрен в 4 главе.

3. Технологии разработки web-приложений

Способы разработки web приложений могут быть разделены на 3 большие категории:

1. Подходы, основанные на программировании или скриптах: внешние программы или скрипты; расширения web-сервера.
2. Подходы, основанные на использовании шаблонов web-страниц, включающих вставки кода скриптов и специальных серверных тэгов.
3. Объектные среды (каркасы, фреймверки, frameworks).

Хотя между этими категориями и имеются пересечения (а также различные мнения о том, к какой категории относится конкретная технология разработки), большинство широко известных подходов связана с одной конкретной категорией.

3.1. Программные подходы

В данном подходе web-приложением (динамическим ресурсом, связанным с URL адресом) является внешняя программа, составленная на некотором универсальном языке программирования высокого уровня (например, таком, как Java или C++) или скрипт, составленный с помощью скриптового языка (scripting language), выполнение которого производится также с помощью внешней программы – интерпретатора скриптов (script engine).

Основной проблемой с программным подходом к разработке web приложений является их ориентация на написание кода. Разметка HTML и другие конструкции форматирования встраиваются в логику работы программы с помощью операторов вывода.

Это ограничивает возможности web-дизайнеров вносить свой вклад в оформление создаваемой приложением страницы. Web-дизайнер может разрабатывать макет страницы, а программист должен затем преобразовывать его в код и связать со скриптом или программой. Для изменения практически любого элемента формируемой страницы требуется вмешательство программиста, касается ли это изменения логики работы программы, либо изменения оформления и расположения элементов страницы.

Внешние программы

Простейший способ динамически формировать web-страницы в ответ на HTTP запрос заключается в том, чтобы передать работу по решению требуемой задачи и формированию HTML страницы внешней программе, которая должна получать переданные в HTTP запросе входные параметры и сформировать выходную страницу на языке HTML.

Первой широко используемой, независимой от типа web сервера, программной технологией создания и выполнения web-приложений была технология Common Gateway Interface (CGI, общий шлюзовой интерфейс). Она определяла набор правил, которым должна следовать программа, чтобы она могла выполняться на разных HTTP серверах и операционных системах.

В соответствии с CGI технологией при поступлении в web-сервер HTTP запроса, который включает ссылку не на статическую страницу, а на CGI программу (например: prog.exe), создается новый процесс, в котором запускается требуемая прикладная программа. Технология CGI задает способ передачи такой программе параметров, входящих в состав HTTP запроса. Передача входных данных может выполняться либо с помощью фиксированного набора переменных среды (environment variables), которые могут создаваться одной программой и использоваться другими программами), либо через входные данные функции, с которой начинается работа программы (функция main()), а результаты работы программы (HTML страница) возвращаются с помощью стандартного потока вывода STDOUT. Пример простой CGI программы, написанной на языке C, которая формирует HTML страницу с перечнем переданных ей параметров, показан на рис. 3.1.

Технология CGI позволяет использовать любой язык программирования, который может работать со стандартными устройствами ввода/вывода. Кроме этого, CGI программы можно писать с использованием скриптовых языков, которые называются “CGI скриптами”. Примерами скриптовых CGI языков являются, например, Perl, Python или Tcl. При использовании скрипта web-сервер вызывает на выполнение внешнюю программу – интерпретатор скриптов (script engine), которой передаются данные HTTP запроса и имя файла, в котором содержится запрашиваемый пользователем скрипт. А затем данная программа выполняет указанный скрипт и возвращает серверу сформированную HTML страницу.

```
// Пример простой CGI программы:
#include "stdafx.h"
#include "cgi1.h"
#include <stdlib.h>
int main(int argc, TCHAR* argv[ ], TCHAR* envp[ ]) {
    printf("Content-Type: text/html\n\n"); printf("<HTML>\n");
    printf("<HEAD> <TITLE>Пример CGI-программы </TITLE></HEAD>\n");
    printf("<BODY>"\n); printf("Пример CGI-программы на C\n");
    int i=0;
    while(envp[i]) {
        // перечать параметров переданных программе в массиве envp
        printf("<p>значение параметра %s </p>",envp[i]);
        i++;
    }
    printf("</BODY>\n"); printf("</HTML>\n);
    return 0;
}
```

Рис. 3.1. Пример простой CGI программы на языке C, показывающий в HTML документе передаваемые ей параметры

Пример скрипта на языке Perl, формирующего HTML страницу с перечнем переданных ей параметров, показан на рис. 3.2.

```
#! /usr/local/bin/perl

sub ReadFormFields { . . . }
sub PrintFormFields {
    my $fieldsRef = shift;
    my $key, $value;
    print "Content-Type: text/html\n\n" ;
    print "<html>\n<head><title>hello</title></head>\n" ;
    print "<body>\n" ;
    foreach $key (keys(%$fieldsRef) ) {
        $value = $$fieldsRef{ $key} ;
        print " <h3>$key: $value</h3>\n";
    }
    print "</body>\n</html>\n";
}
&ReadFormFields( \%fields) ;
&PrintFormFields(\%fields) ;
exit 0;
```

Рис. 3.2. Пример скрипта на языке Perl, формирующий HTML страницу с переданными параметрами формы

Недостатки технологии CGI

Технология CGI является достаточно простым способом динамически формировать информацию в web-сети, но она имеет существен-

ные недостатки, которые делают ее не практичной в большинстве случаев:

- Основной проблемой является производительность: для каждого HTTP запроса к CGI программе web-сервер запускает новый процесс, который заканчивает работу только после завершения программы. Работа по созданию и завершению процессов является достаточно трудоемкой, что может очень быстро понизить производительность системы, кроме этого различные активные процессы начинают конкурировать за системные ресурсы, такие как оперативная память.
- Для составления и отладки CGI программ разработчик должен обладать достаточно большим опытом программирования на одном из языков, на которых можно программировать CGI программы.
- В CGI программах программный код и код разметки полностью перемешаны. Дизайнер должен знать программирование, чтобы менять структуру web-страниц.

Попыткой объединить переносимость CGI приложений с эффективностью является технология FastCGI. Данная технология основывается на простой идеи: вместо необходимости каждый раз запускать новый процесс для обработки CGI скрипта, FastCGI позволяет не закрывать процессы, связанные с CGI скриптами, после окончания обработки, а использовать их для обработки новых запросов к CGI программам. А это означает, что не требуется постоянно запускать и удалять новые процессы, так как один и тот же процесс может использоваться многократно для обработки запросов. Такие процессы могут инициализироваться только один раз при их создании.

Модули сервера, которые выполняют функциональность FastCGI, взаимодействуют с HTTP сервером с помощью своих собственных API. Эти API стараются скрыть детали реализации и конфигурирования от FastCGI приложений, но разработчики все равно должны знать особенности реализации технологии FastCGI, так как модули различных типов серверов не совместимы между собой.

Расширения web-серверов

Недостатки технологий CGI можно также преодолеть путем расширения возможностей web-серверов с помощью специальных компонентов. Используя такие расширения, программы, формирующие HTTP ответы, могут выполняться более эффективно, без необходимости их завершения после обработки каждого запроса и за счет использования общих ресурсов несколькими приложениями. Такие технологии обычно предоставляют возможность хранить в основной памяти данные сеансов

работы пользователей, которые взаимодействуют с приложением в течение большого числа HTTP запросов.

Интерфейс ISAPI

Для web-сервера Microsoft IIS (Internet Information Server) был разработан специальный программный интерфейс – ISAPI, позволяющий создавать приложения, расширяющие стандартные возможности данного web-сервера. ISAPI представляет собой библиотеку функций, с помощью которой программисты могут создавать web-приложения в виде DLL модулей (динамически подключаемых библиотек), формирующих HTML страницы. Такие web-приложения работают намного быстрее обычных CGI программ, так как они более тесно интегрированы в web-сервер.

С помощью технологии ISAPI могут создаваться два вида компонентов: *расширения и фильтры*, которые компилируются в DLL файлы, динамически запускаемые web-сервером. ISAPI-*расширения* могут связываться с вызовом файлов, имеющих специальные расширения, либо с файлами, содержащимися в заданных каталогах или во всем web сайте. ISAPI-*фильтры* используются для изменения или совершенствования функциональности IIS сервера. Обычно они обрабатывают (фильтруют) каждый поступающий HTTP запрос. Фильтры могут применяться для анализа и модификации исходящих HTTP ответов.

ISAPI приложения могут разрабатываться с помощью разных языков, поддерживающих экспорт стандартных C-функций, например, таких, как C, C++, Delphi.

Интерфейс Java Servlet API

Другой широко используемой технологией расширения архитектуры web-сервера является прикладной интерфейс Java Servlet API, который связывает web server с виртуальной машиной Java Virtual Machine (JVM). Виртуальная машина JVM поддерживает выполнение специальной Java программы (контейнер сервлетов), которая отвечает за управление данными сеанса работы и выполнение Java сервлетов.

Сервлеты это специальные классы на языке Java (программы), которые имеют доступ к информации из HTTP запросов. Они формируют HTTP ответы, которые возвращаются браузерам. На рис. 3.3 показан пример сервлета, который аналогичен CGI скрипту, показанному на рис. 3.1.

Контейнер сервлетов (среда выполнения) отвечает за получение от web сервера HTTP запросов на выполнение сервлетов; создание сеанса работы пользователя, если это требуется; вызов сервлета связанного с

HTTP запросом; передачу сервлету параметров, которые содержатся в HTTP запросе, представленных в виде Java объектов.

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class FormServlet extends HttpServlet {
    public void doGet( HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException {
        response.setContentType("text/html" ) ;
        PrintWriter out = response.getWriter();
        out.println("<html>\n<head><title>hello</title></head>" ) ;
        out.println("<body>");
        Enumeration e = request.getParameterNames() ;
        while (e. hasMoreElements() ) {
            String name = (String) e.nextElement() ;
            String value = request.getParameter(name) ;
            out.println(" <h3>" + name + &: & + value + " </ h3>" ) ;
        }
        out.println( "</body>\n</html>" ) ;
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException {
        doGet( request, response);
    }
}
```

Рис. 3.3. Пример сервлета, формирующего HTML страницу с переданными параметрами формы

В отличие от ISAPI расширений, технология Servlet API является переносимой между разными web-серверами, операционными системами и компьютерными платформами. Сервлеты выполняются одинаково в любой среде, которая предоставляет совместимый с ними контейнер сервлетов. Технология Servlet API используется большим количеством разработчиков и поддерживается многими известными web серверами.

3.2. Походы на основе шаблонов

Из примеров программ 3.1-3.3 видно, что они включают с помощью операторов вывода в формируемые HTML страницы статическую HTML разметку (тэги) и изменяемую информацию (получаемую в результате вычислений или выборки данных из баз данных). При изменении статической разметки логика формирования динамической информации не меняется, но в программу необходимо вносить изменения и выполнять их перекомпиляцию (если это не скрипт).

Подходы, основанные на *шаблонах* (template approaches, шаблонные подходы) используют в качестве адресуемых объектов (имеющий URL адрес) не программы или скрипты, а «***шаблоны***». По существу шаблоны являются HTML файлами с дополнительными “тэгами” (серверные, используемые только на стороне сервера), которые задают методы включения динамически формируемого контента. Таким образом, файл шаблона содержит HTML код, который описывает общую структуру страницы, и дополнительные ***серверные тэги***, размещенные таким образом, чтобы формируемой с их помощью содержание странице имело требуемый вид.

В конце 90-х годов многие компании разработали свои собственные технологии обработки шаблонов на стороне web-сервера, включающие скрипты. Компания Netscape предложила технологию LiveWire (которая развилась в язык Server-Side JavaScript), а другие компании разработали такие технологии, как NetDynamics, Dynamo и Cold Fusion (из этих технологий до настоящего времени используется только Cold Fusion).

В настоящее время к наиболее распространенным технологиям разработки web-приложений на основе шаблонов, относятся следующие: *Server-Side Includes (SSI), Cold Fusion*, *PHP*, *Active Server Pages (ASP)* и *Java Server Pages* (*JSP).*

Технология SSI

Технология вставок на стороне сервера – Server Side Includes (SSI) является старой технологией, которая появилась почти одновременно с технологией CGI. SSI предоставляет возможность вставки дополнительных файлов (или результатов выполнения CGI скриптов) в HTML страницу. Вставка в шаблон SSI инструкций выполняется с помощью следующего формата:

```
<!-- #instruction attr1="value1" attr2 ="value2" -->
```

Таким образом, SSI инструкции вставляются в виде HTML комментариев, тем самым гарантируя, что не обработанные на сервере инструкции будут игнорироваться браузером, получившим такой файл. Примерами SSI инструкций являются: echo – для вывода значений переменных среды; include – для вставки содержания других файлов; exec – для выполнения CGI программы на стороне сервера и включения сформированного ею HTML кода в шаблон.

На рис. 3.4 показан упрощенный CGI скрипт (аналогичный CGI скрипту 3.2, но формирующий только содержание HTML разметки).

```
#! /usr/local/bin/perl
sub ReadFormFields { . . . }
sub PrintFormFields
{
        my $fieldsRef = shift;
        my $key, $value;
        foreach $key (keys(%$fieldsRef) ) {
                $value = $$fieldsRef{ $key} ; print " <h3>$key: $value</h3>\n";
        }
}
&ReadFormFields( \%fields) ; &PrintFormFields(\%fields) ;
exit 0;
```

Рис. 3.4. Упрощенный CGI скрипт для использования с помощью SSI инструкций, показанных на рис. 3.5

А на рис. 3.5 показан шаблон, в который вставляется результат работы скрипта 3.4 в SSI страницу (которая обычно имеет расширение shtml).

```
<HTML>
   <HEAD><TITLE>hello</TITLE></HEAD>
   <BODY>
      <! -- #exec cgi http://mysite.org/cgi-bin/zip-ssi.cgi -->
   </BODY>
</HTML>
```

Рис. 3.5. SSI шаблон, использующий CGI программу, показанную на рис. 3.4

Технология SSI предоставляет простой и удобный способ добавить динамическое содержание к уже существующим страницам, без необходимости генерировать всю страницу целиком.

Технология Cold Fusion

Другой достаточно популярной технологией, основанной на шаблонах, является технология Cold Fusion, разработанная компанией Adobe. Пример шаблона описанного на основе данной технологии показан на рис. 3.6. В данном шаблоне используются специальные тэги, внешне очень похожие на HTML тэги, но начинающиеся с приставки “CF”. Такие тэги не передаются браузерам, а обрабатываются средой выполнения на стороне web-сервера.

```
<CFQUERY name="query1" datasource="oracle" . . . >
   SELECT id, columnX, columnY, columnZ
      FROM TABLE1
      WHERE id = #substitution-parameter#
</CFQUERY>
<CFIF query1.recordcount GT 0>
```

```
   <TABLE>
      <CFOUTPUT QUERY="query1">
         <TR>
            <TD>#columnX#</TD>
            <TD>#columnY#</TD>
            <TD>#columnZ#</TD>
         </TR>
      </CFOUTPUT>
   </TABLE>
</CFIF>
```

Рис. 3.6. Шаблон технологии Cold Fusion

Блок шаблона, выделенный тэгами <CFQUERY>, описывает запрос к базе данных (БД). Тэги <CFIF> выделяют блок шаблона, который должен быть включен в результирующую страницу, если выборка, полученная по запросу с именем query1, не будет пустая (количество элементов будет больше чем 0). В рамках данного блока, тэги <CFOUTPUT> описывают содержание строк HTML таблицы, которые должны повторяться (с подстановкой соответствующих значений выборки) для каждой строки в результирующем наборе запроса. (Отметим, что текст в блоке, выделенном тэгами Cold Fusion CFOUTPUT, который ограничен символами #, указывает на замещаемый параметр, например, #text#.)

Преимущество данного подхода заключается в том, что данный шаблон может создаваться и поддерживаться дизайнером страницы, который имеет базовые знания языка HTML и web-графики, но не имеет опыта программирования. Специальные тэги, которые являются “расширением” HTML в некоторой степени похожи на инструкции (тэги) SSI, тем, что позволяют быстро научиться использовать их дизайнерами web-страниц, которые имеют не большой опыт работы. Использование таких тэгов требует меньше опыта, чем написание программного кода.

Технология PHP Hypertext Preprocessor

Технология “PHP Hypertext Preprocessor” (хотя первоначально она соответствовала “Personal Home Page”) или просто PHP позволяет разработчикам встраивать программный код в шаблоны, с помощью языка, сходного с языком скриптов Perl. Ниже приведен пример фрагмента PHP шаблона:

```
<b>
<?php if ($xyz >= 3 ) { print $myHeading; }
      else {
?>
      DEFAULT HEADING
<?php } ?>
</b>
```

Данный шаблон эквивалентен следующему скрипту:

```
print " <b>";
if ( $xyz >= 3) { print $myHeading; }
else { print "DEFAULT HEADING" ; }
print " </ b>"
```

Из данного примера понятно, что текст, встроенный в блоки <?php. . . ?> обрабатывается PHP процессором, а текст стоящий вне таких блоков, обрабатывается, как аргумент переданный операторам print. (Такой же подход используется и в технологиях JavaServer Pages и ASP.Net Web Forms.)

Технология Active Server Pages

Компания Microsoft разработала технологию ASP (Active Server Pages), которая объединила возможности создания шаблонов, включающих скрипты, с доступом к набору OLE и COM объектов, имеющихся с операционной системе Windows, в том числе и к ODBC источникам данных. Данная технология, объединенная с бесплатным web сервером Internet Information Server (IIS), быстро стала популярной среди программистов, использующих Visual Basic, которые оценили возможность использования в шаблонах языка VBScript. Как и PHP шаблоны, ASP страницы могут включать блоки скриптов (использующими ссылки на COM объекты), вперемешку с HTML форматированием. В отличие от таких объектно-ориентированных языков, как Java или C++, язык, используемый в ASP страницах, был плоским, линейным и строго процедурным.

В отличии от технологии PHP, ASP не связан с одним конкретным скриптовым языком. В ASP в качестве стандартного языка используется язык Visual Basic Scripting Edition (VBScript), но может использоваться и язык JavaScript.

В ASP шаблоны (также, как и в PHP шаблоны) могут включаться блоки, выделенные с помощью тэгов <% … %>, которые содержат код скрипта, выполняемый интерпретатором ASP шаблонов, при формировании ответа. HTML разметка, который находится вне таких блоков, рассматривается как исходный HTML код и просто переписывается в формируемую HTML страницу. Кроме этого в начало шаблона могут добавляться директивы страницы, такие, как например, <% @LANGUAGE = VBScript %>, которая информирует систему обработки об используемом скриптовом языке.

На рис. 3.7, показана ASP страница (шаблон), которая включает два встроенных в нее «блока скриптов».

```
<% @LANGUAGE = VBScript %>
<% Set conn = Server.CreateObject("ADODB. Connection")
conn.open("Data Source=mydata; User ID=myname;Password=*****")
Set results = Server.CreateObject("ADODB.RecordSet")
Set results.ActiveConnection = conn
query = "SELECT X, Y, Z FROM TABLE1 WHERE X > 23"
results.Open query %>
<html> <head><title>Active Server Page Example</title></head>
<body id="c09-body-0003 " bgcolor="#ffffff" >
    <table>
        <tr>
            <td align="center">X</td>
            <td align="center">Y</td>
            <td align="center">Z</td>
        </tr>
    <%
        While Not results. EOF
            Response.Write "<tr>"
            Response.Write "<td>" &results(" X") & " </td>"
            Response.Write "<td>" &results(" Y") & " </td>"
            Response.Write "<td>" &results(" Z") & " </td>"
            Response.Write "</tr>"
        Wend
    %>
    </table>
</body>
</html>
```

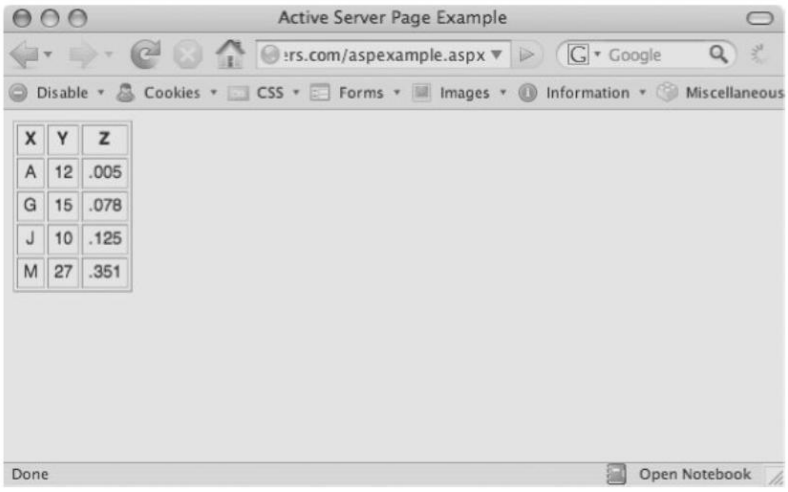


Рис. 3.7. Пример описания ASP страницы и полученный результат

Первый блок, расположенный в начале шаблона, создает соединение с БД и открывает его с использованием заданных параметров подключения; после этого создается объект results, который будет содержать данные полученные из БД и связывается с открытым соединением; затем описывается SQL запрос к БД query и с его помощью результаты

выборки из БД сохраняются в объекте results. Второй блок скриптов включен в описание HTML таблицы. Он выполняет вывод данных из записей результатов выборки results в строки таблицы, каждая ячейка которых содержит значения поля записи.

Тот факт, что технология ASP входила в состав web-сервера Microsoft IIS, сделало ее очень привлекательной для использования разработчиками, работающими в ОС Windows. В связи с популярностью технологии ASP, она была реализована и на других платформах, помимо Microsoft IIS. Основным преимуществом данной технологии является ускорение разработки и установки относительно простых web-приложений.

Технология Java Server Pages

Технология JSP была ответом компании Sun на популярность технологии Microsoft ASP. Пример простой JSP страницы (шаблон), показан на рис.3.8.

В первой строке данного примера показана директива страницы <%@ page …> (интересно обратить внимание на ее сходство с директивой web-формы ASP.Net), в которой указывается на импортирование классов из пакета java.io. В следующей строке выполняется объявление переменной. Код на языке Java выделяется (также как и в PHP и ASP) специальными последовательностями символов <% … %>.

```
<%@ page import="java.io.*" %>
<%! private CustomObject myObject; %>
<h1>My Heading</h1>
    <%
        for(int i = 0; i < myObj ect. getCount(); i++) { %>
            <p>Item #<%= i %> is ' <%= myObject.getItem(i) %>' . </p>
    <% } %>
```

Рис. 3.8. Пример простой JSP страницы

Как и технология PHP, выполнение JSP страниц реализуется с помощью препроцессора, который преобразует (транслирует) их в исходный код сервлета. HTML разметка, которая стоит вне выделенных блоков транслируется в операторы print языка Java (как показано на рис. 3.9).

```
package jsp._myapp ;
import java.io.*;
import java.util. *;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
public class _mypage extends HttpJspBase {
```

```
    private CustomObject myObject;
    public void _jspService(HttpServletRequest req, HttpServletResponse resp){
        ServletConfig config = getServletConfig() ;
        ServletContext application = config. getServletContext() ;
        Object page = this ; PageContext pageContext =
        JspFactory. getDefaultFactory( ).getPageContext(this,
                req, resp, null, true, 8192, true);
        JspWriter out = pageContext.getOut() ;
        HttpSession session = request.getSession( true);
        out.print("<h1>My Heading</h1>");
        for(int i = 0; i < myObject.getCount( ); i++) {
            out.print("<p>Item #" + i + " is ' " + myObject.getItem(i) + " '.</p>");
        }
    }
}
```

Рис. 3.9. Результат трансляции JSP страницы, показанной на рис. 3.8 в коде сервлета

Технология JSP развивалась и со временем к ней были добавлены такие новые возможности, как библиотеки JSP тэгов (***JSP taglib*)**. Библиотека тэгов ***taglib*** это набор специальных (серверных) JSP тэгов, которые не передаются в HTTP ответе браузеру, а используются при обработке JSP страницы в контейнере сервлетов на стороне web-сервера.

Фактически каждый специальный тэг это некоторая функциональность, для реализации которой, в противном случае, потребовалось бы включать с помощью встроенных блоков содержащих Java код. Например, двумя наиболее часто используемыми тэгами являются: <jsp:useBean> и <jsp:getProperty>. Префикс “jsp:” говорит о том, что это не HTML тэг, а специальный (серверный) тэг, который будет использоваться на стороне сервера. Тэг <jsp:useBean> позволяет разработчика встраивать в JSP страницу JavaBean объекты (созданные и наполненные приложением в ходе сеанса работы пользователя). К ним можно получить доступ и изменить их значения с помощью тэгов <jsp:getProperty> and <jsp:setProperty>. В ходе выполнения трансляции JSP страницы, которая выполняется до компиляции и выполнения созданного сервлета, выполняется преобразование таких специальных тэгов в Java код. Например, имеется следующий фрагмент JSP страницы:

```
<jsp:usebean id="myBean" class="mypackage.MyBean" scope="session"/>
. . .
<p>Значение свойства 'thing' равно
            <jsp:getProperty name="myBean" property="thing" />' .
</p>
```

Данный фрагмент будет транслироваться в следующий Java код:

```
MyBean myBean = (MyBean) session.getAttribute("myBean");
```

```
out.print("<p> Значение свойства 'thing' равно ' " +
        myBean.getThing().toString( ) + " ' . </p>";
```

Отметим синтаксические сложности, связанные с подстановкой переменных в среде JSP. Для доступа к свойству JavaBean должен быть включен специальный тэг <jsp:getProperty>. Альтернативным способом является использование синтаксиса <%= object.variable %>.

Технология Java Standard Tag Library

Компания Sun продолжила развивать технологию JSP, как платформу разработки, включив нее возможность создания своих собственных JSP тэгов. В результате этого появилось большое количество специализированных тэгов разработанных локально различными группами разработчиков, что привело к еще большей путанице. Первоначально компания Sun не предоставила ни стандартов, ни указаний по организации и структурированию специализированных тэгов (custom tags).

Однако затем компания предложила спецификацию библиотеки стандартных тэгов на языке Java – Java Standard Tag Library (JSTL). В нее были включены стандарты для тэгов выполнения итерации, условной обработки, доступа к БД и много других тэгов, выполнения форматирования. JSTL вначале была создана в качестве не обязательного дополнения к JSP, но позже она и связанные с ней языки выражений были включены в спецификацию JSP 2.0.

Тэги JSTL разделены на несколько категорий:

- Базовые тэги, предоставляющие стандартную функциональность, обычную для шаблонного подхода: включение, выполнение итераций, проверка условий и т.п.
- XML тэги, предоставляющие аналогичную функциональность в XML контексте, а также возможность обхода элементов XML документа с использованием выражений XPath.
- SQL тэги, предоставляющие средства для описания источников данных, предоставления запросов и выполнения итераций по полученным результатам выборки из БД.
- Форматирующие тэги, включающие функции интернационализации и локализации, а также средства для форматирования дат и чисел.

На рис. 3.10 показано использование JSTL тэгов для выполнения запроса к БД и представление полученных результатов в виде HTML таблицы. JSTL тэг <c:forEach> используется для выполнения цикла по строкам (rows), полученным из БД, и выполнения их отображения в HTML таблице (HTML тэги <tr>) с индивидуальными значениями коло-

нок, представленных с помощью JSP Expression Language в виде ячеек HTML таблицы (HTML тэги <td>).

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jstl/sql" prefix="sql" %>

<sql:setDataSource var="myDatabase" driver="com.mysql.jdbc.Driver"
   url="jdbc:mysql://localhost:3306/myschema scope="session" />
<sql:query var="contacts" dataSource="${myDatabase}" >
   select name, address, phone from contacts
</sql: query>
<html>
<head><title>JSTL Example</title></head>
<body id="c09-body-0004" bgcolor="#ffffff" >
   <table border="1" cellpadding="4">
      <tr>
         <td align="center"><b>Name</b></td>
         <td align="center"><b>Address</b></td>
         <td align="center"><b>Phone</b></td>
      </tr>
      <c:forEach var="contact" items="${contacts.rows} ">
         <tr>
            <td align="center">${contact.name} </td>
            <td align="center">${contact.address}</td>
            <td align="center">${contact.phone} </td>
         </tr>
      </c: forEach>
   </table>
</body>
</html>
```

Рис. 3.10. JSP страница, использующая технологию JSTL и полученный результат

SQL тэги в начале шаблона (<sql:setDataSource> и <sql:query>) устанавливают соединение с БД, возвращают результаты и назначают их переменным. Базовый тэг <c:forEach> выполняет цикл по результатам выполнения запроса к БД, показывает каждую полученную строку (запись) в виде строки HTML таблицы (<tr>), в которой каждая ячейка

(<td>) содержит значение для каждой колонки результата (можно отметить сходство с примером для технологии Cold Fusion на рис. 3.6.)

Подстановка параметров выполняется с помощью языка выражений (expression language, EL), связанного с JSTL, предоставляющего доступ к переменным, определенным в разных областях: на странице (page), в запросе, сеансе работы пользователя (session) или всего приложения в целом (application). Обращение к переменным выполняется следующим образом: ${scopeName.variableName}, где scopeName – область определения переменной (страница, сеанс или приложение). Такое обращение к переменным далее расширяется путем использования сложных объектов Java, включая массивы, коллекции и отображения (maps) (например, ${session.myObject.myMemberVariable}).

JSTL сделали возможным составление JSP страниц без включения в нее кода на языке Java. Тем самым данная технология предоставила полезные возможности для реализации компонента View в JSP Model 2, в особенности в такой объектной MVC среде, как Struts. Хотя JSTL имеет все возможности для создания полноценных web приложений, в основном она используется в качестве технологии разработки представлений (Views) в контексте объектных MVC сред.

3.3. Подходы на основе объектных сред

Обычные скриптовые технологии на стороне сервера используют различные объекты, но не позволяют разрабатывать и использовать собственные классы и создавать на их основе объекты. В связи с этим дальнейшее развитие web-технологий было связано с созданием специальных объектно-ориентированных технологий разработки web-приложений. Использование данных технологий позволяет сделать разработку web-приложений более сходной с разработкой обычных объектно-ориентированного программного обеспечения.

Объектные среды (фреймверки, frameworks) представляют собой следующий уровень совершенствования разработки web-приложений. Вместо объединения разметки и логики в единый модуль, объектные среды (frameworks) поддерживают принцип отделения содержания от представления. Модули ответственные за создание контента отделяется от модулей, которые показывают это содержание в конкретном формате.

Отделение содержания от представления является важным в связи с тем, что увеличивается гибкость приложения (возможность его изменения с небольшими затратами); улучшается разделение ответственности между web-дизайнерами и программистами.

В настоящее время есть два подхода к созданию объектно-ориентированных web-приложений:

- подходы, основанные на наборе специальных web-страниц (web-форм), связанных с описаниями классов, объекты которых будут создаваться и использоваться при их вызове (например: технология ASP.Net Web Forms**;** технология JavaServer Faces);
- подходы, основанные на использовании наборов классов, соответствующих шаблону Model-View-Controller (MVC) (например: технологии на основе языка Java – Tapestry, Struts, Spring и технология компании Microsoft – ASP.Net MVC).

Объектный подход на основе форм

Подход на основе web-форм является дальнейшим развитием скриптовых серверных технологий. В данном подходе в HTML документы добавляются специальные тэги, обрабатываемые на стороне сервера. Кроме этого, можно описывать и использовать собственные тэги в виде классов на универсальных языках программирования (Java, C#, VisualBasic и т.п.). Создание на стороне сервера объектной модели приложения, аналогичной объектной модели локального приложения.

Основными объектными подходами на основе форм являются следующие:

- технология Microsoft ASP.Net Web Form (рассматривается в 4 главе);
- технология JavaServer Faces (JSF).

Технология JavaServer Faces была ответом компании Sun на появление технологии ASP.Net (также, как JSP была ответом на популярность технологии ASP), и в частности на используемую в ней поддержку интерфейса пользователей.

В отличии от технология JSTL, которая уже предоставляла низкоуровневый подход к описанию структуры web-страниц, предлагая хорошо детализированные серверные тэги для условной обработки и выполнения циклов по результатам запросов к базам данных, технология JSF предоставляет более высокоуровневый подход с помощью укрупненных визуальных компонент, например, тэга <f:view>.

```
<%@ taglib prefix=" f" uri=" http://java.sun.com/jsf/core" %>
<%@ taglib prefix=" h" uri=" http://java.sun.com/jsf/html" %>
<html> <head><title>JSF Example</title></head>
<body id="c09-body-0002" bgcolor="#ffffff" >
    <f: view>
        <h: dataTable value=" #{bean1.contacts}" var=" contact"
                styleClass="tab" headerClass="header"
```

```
            rowClasses="oddRow, evenRow" >
          <h: column>
            <f:facet name="header" > <h:outputText value="Name" />
            </f:facet> <h:outputText value=" #{contact. name} " />
          </ h: column>
          <h: column>
            <f:facet name=" header" ><h: outputText value="Address" />
            </f:facet>
            <h:outputText value="#{contact.address}" />
          </ h: column>
          <h: column>
            <f:facet name="header" ><h:outputText value="Phone" />
            </f:facet>
            <h:outputText value="#{contact. phone}" / >
          </h:column>
        </h:dataTable>
    </f:view>
</body>
</html>
```

Рис.3.11. Пример JSF страницы аналогичной JSP странице в примере 3.10

На рис. 3.11 показан пример JSF страницы, аналогичной примеру 3.10, в которой выполняется результатов запроса к БД отображение с помощью тэга <h:dataTable>, который абстрагирует понятие HTML таблицы.

Подход на основе архитектурного шаблона MVC

В соответствии с архитектурным шаблоном MVC все классы, составляющие приложение (в том числе и web-приложение) делятся на три основные группы (компоненты): Модель (Model), Представление (View) и Контроллер (Controller). Логика работы web-приложения с использованием архитектуры MVC показана на рис. 3.12.

Каждый из этих компонентов отвечает за свои задачи:

- **Модель (Model)** – это набор классов, реализующих всю бизнес-логику web-приложения. Эти классы отвечают за обработку данных (сущностей), размещение их в БД, чтение из БД, а так же за взаимодействие между самими объектами, составляющими такие данные.
- **Представление (View)** – набор классов, отвечающих за интерфейс взаимодействия с пользователями (User Interface, UI). С их помощью формируются HTML страницы, показывающие пользователям данные Представление использует данные из Модели и предоставляет пользователям возможность выполнять их редактирование.
- **Контроллер (Controller)** – это связующее звено между первыми двумя компонентами. Классы данного компонента получают дан-

ные о запросе к серверу (например, значения, полученные из отправленной формы), и передает их в Модель для обработки и сохранения. После обработки полученных данных Контроллер выбирает, каким способом показать их клиенту с помощью использования некоторого класса из Представления.

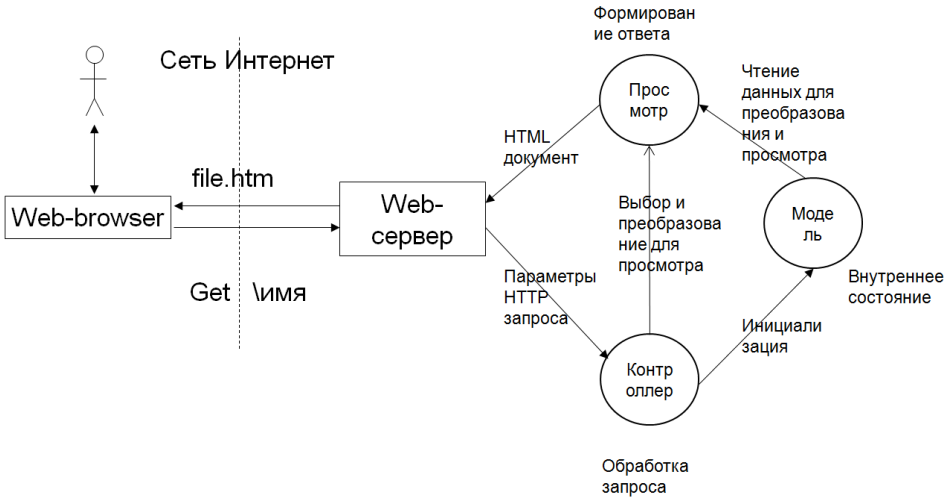


Рис. 3.12. Построение web-приложения на основе шаблона MVC

В результате такого разделения web-приложения на компоненты, разработчик получает полный контроль над формируемым HTML документов; упрощается структура приложения; облегчается задача выполнения тестирования приложения; достигается полное отделение логики работы приложения от представления данных

Примерами технологий разработки на основе MVC являются:

- технология Struts (основанная на языке Java);
- технология ASP.Net MVC, входящая в состав набора технологий ASP.Net платформы .Net Framework;
- технология Ruby on Rails (Ruby – язык программирования, а Rails – фреймворк, использующий данный язык) особенно популярная в последнее время.

4. Разработки серверных web-приложений с помощью ASP.Net

ASP.Net это набор технологий разработки web-приложений и web-сервисов, которая является частью инфраструктуры .NET Framework. ASP.Net включает следующие технологии:

- ASP.Net Web Forms – технология создания web-приложений с помощью web форм, использующих серверные элементы управления;

- ASP.Net Web Service – технология создания web-сервисов (ASMX сервисов), которые могут вызываться из web-приложений или Windows приложений;
- ASP.Net MVC – технология создания web-приложений с использованием шаблона Model-View-Controller.

Web-приложения и web-сервисы, разработанные с помощью ASP.Net, могут работать только под управлением web-сервера Internet Information Services (IIS), который является составной частью операционной системы Windows Vista, Windows 7, Windows Server 2008.

Среда разработки Visual Studio (входящая в состав платформы Microsoft .NET Framework) позволяет создавать Windows приложения и web-приложения, используя очень сходные способы работы, одинаковые языки программирования (C# и Visual Basic) и технологии доступа к данным (ADO.Net).

Технология ASP.Net Web Forms является объектным подходом на основе форм (см. описание в разделе 3.2). Она представляет собой развитие шаблонного подхода с использованием средств полностью объектно-ориентированного программирования. В соответствии с данной технологией ***web-приложение*** является виртуальной папкой web-сервера IIS, которая описывается в его файле конфигурации как «Web-приложение». Основным содержанием данной папки являются ***web-формы*** (файлы с расширениями *.aspx и *.aspx.cs), ***файл конфигурации приложения*** (файл web.config), файлы с дополнительным программным кодом (модули бизнес логики, модули доступа к данным) и различные ресурсы (изображения, мультимедиа файлы).

Основными компонентами создаваемых web-приложений являются ***web-формы***, которые похожи на Windows формы (класс Form из пространства имен System.Windows), т.к. также являются контейнерами для специальных серверных элементов управления (ЭУ). Кроме этого web-формы похожи и на шаблоны web-страниц, которые показаны на рис. 3.6 и 3.10. Они также включают специальные, серверные тэги, которые в данной технологии называются серверными элементами управления (ЭУ), так как по функциональности очень похожи на элементы управления ОС Windows.

Серверные ЭУ способны показывать данные и инициировать события, для которых могут быть созданы обработчики (методы, выполняющие обработку возникающих событий). При поступлении HTTP запроса к web-форме, ее программный код (и программный код серверных ЭУ) выполняет обработку этого запроса и формирует HTML код, который отправляются в качестве HTTP ответа.

В состав web-приложения также входят XML файлы **конфигурации**, содержащие параметры работы web приложения в среде выполнения, параметры безопасности, обработка приложением возникающих ошибок и т.п.

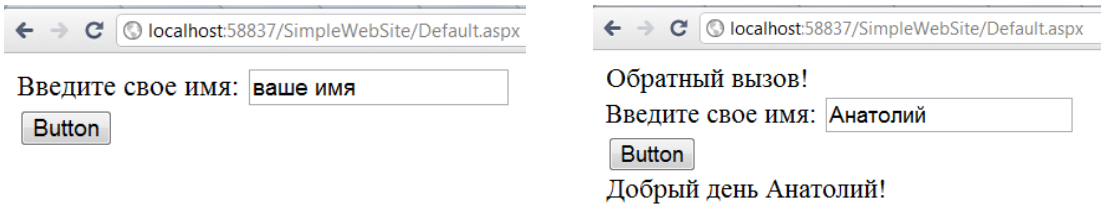
При первом обращении какого-либо пользователя к ресурсам папки, в которой расположено web-приложение, оно будет запускаться на выполнение под управлением ***сервера ASP.Net приложений*** (***среды выполнения***), который выполняется в специальном процессе. Каждое приложение выполняется в своем ***домене приложения*** (application domain). Домены приложений представляют собой аналоги процессов операционной системы, которые реализуются с помощью платформы .NET. Они гарантирует невозможность одного приложения оказывать влияние на другое приложение и, в тоже время, работать в рамках одного адресного пространства. В одном процессе могут выполняться много доменов приложений.

4.1. ASP.Net web-формы

Основными компонентами web-приложение, созданного с помощью технологии ASP.Net Web Forms, являются web-формы. Каждая web-форма состоит из ***шаблона*** и ***программного кода***, обычно расположенных в двух разных файлах:

- **Шаблон web формы** расположен в файле с расширением .aspx (например, Default.aspx)**,** и содержит смесь HTML кода (стандартных тэгов HTML языка) и специальных серверных тэгов (с помощью которых описываются серверные элементы управления). Такие шаблоны аналогичны ранее рассмотренным шаблонам технологии Cold Fusion и JavaServer Faces.
- **Программный код** расположен в файле, который имеет название аналогичное названию соответствующего ему шаблона, но с добавлением расширения .cs (если используется язык программирования C#). Например, Default.asmx.cs. В данном файле содержится описание класса производного от базового класса Page (из пространства имен System.Web.UI), соответствующего данной web-форме.

Рассмотрим в качестве примера очень простую web-форму, предлагающую пользователя ввести имя, а затем выполняющую его приветствие. Визуальный интерфейс и результаты работы данной формы показаны на рис. 4.1.



(a) (b)

Рис. 4.1. Пример простой формы:
(а) вид формы при начальном вызове; (б) вид формы при повторном обращении к форме после ввода имени и нажатия кнопки.

Шаблоны web-формы

Описание шаблона web-формы, показанной на рис. 4.1 приведено на рис. 4.2. Как видно из данного описания, в шаблоне формы, кроме обычной HTML разметки (html, body, form, div и т.п.), вставлены специальные тэги (asp:TextBox, asp:Button и asp:Label), которые называются серверными элементами управления. Такие тэги не передаются браузеру в HTTP ответе, а обрабатываются на сервере. При обработке запроса пользователя к шаблону (Default.aspx) для каждого серверного ЭУ создается объект, с которым может работать программный код формы (Default.aspx.cs).

Основным элементом всех шаблонов форм является элемент Form (имеющий атрибут runat="server"), т.е. каждая web-форма является страницей с HTML формой. В данной форме не задан атрибут Action, и поэтому при инициировании пользователем отправки данных элементов формы на сервер (например, при нажатии кнопки типа submit) они опять будут отправлены той же самой web-форме. Такое повторное обращение web-формы к самой себе называется ***обратной отправкой*** (post-back).

```
<%@ Page Language="C#" AutoEventWireup="true"
            CodeFile="Default.aspx.cs" Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server"> <title></title></head>
<body>
    <form id="form1" runat="server">
        <div>
            Введите свое имя:
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox><br />
            <asp:Button ID="Button1" runat="server" onclick="Button1_Click"
            Text="Button" /><br />
            <asp:Label ID="Label1" runat="server" Text="Label"
```

```
                    Visible="False"></asp:Label>
        </div>
    </form>
</body>
</html>
```

Рис. 4.2. Описание примера шаблона web-формы.

Шаблон web-формы также включает управляющую директиву, выделенную символами <%@ … %>, в которой описывается тип и параметры шаблона. Например, директива web-формы может имеет следующий вид:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
    Inherits="_Default" %>
```

В данной директиве задается язык, используемый в программном коде данной формы (C#); указывается файл, в котором содержится программный код (Default.aspx.cs) и имя класса, который соответствует данной странице (_Default), а также указывается, что связывания событий страницы (объекта Page) с методами обработки выполняется автоматически (AutoEventWireup="true", например, метод Page_Load автоматически связывается с событием Load для объекта класса Page).

В шаблоне могут быть заданы разные директивы, например такие, как Page, Master, WebService, Application, OutputCache, Control и др. Некоторые из них будут далее рассмотрены в пособии.

Программный код web-формы

Пример программного кода для шаблона, показанного на рис. 4.2, приведен на рис. 4.3.

Как видно из данного примера, в программном коде описан класс _Default, производный от базового класса System.Web.UI.Page. В данном классе автоматически (в самом программном коде это не указывается) для всех серверных тэгов формы (в данном случае: asp:TextBox, asp:Button и asp:Label) создаются соответствующие им программные объекты (в качестве их имен используются значения атрибутов id, в данном случае: Button1, Label1 и TextBox1), которые выполняет обработку действий пользователя и запись в HTTP-ответ вместо серверных тэгов, соответствующего им кода HTML разметки.

Созданный производный класс в основном содержит методы, вызываемые при возникновении различных событий, связанных с формой в целом (например, метод Page_Load() вызывается при возникновении события «загрузка формы») или связанные с включенными в форму

серверными элементами управления (например, метод Button1_Click вызывается при возникновении события «нажатие кнопки»).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e) {
        if (!IsPostBack)
            TextBox1.Text = "ваше имя";
        else
            Response.Write("Обратный вызов!");
    }
    protected void Button1_Click(object sender, EventArgs e) {
        Label1.Text = "Добрый день " + TextBox1.Text + "!";
        Label1.Visible = true;
    }
}
```

Рис. 4.3. Программный код примера web-формы.

Такой программный код web-формы исполняется средой выполнения ASP.Net.

Порядок выполнения запросов к web-форме

При первом обращении пользователей к шаблону формы, она преобразуется в DLL файл, содержащий класс, описанный в программном коде. В этот класс включаются поля, соответствующие серверным ЭУ, а также метод RenderControl(), который формирует HTML страницу ответа. Все статические HTML элементы шаблона формы выводятся в страницу ответа с помощью методов вывода Write(), а для всех серверных ЭУ вызываются имеющиеся у них методы RenderControl(), с помощью которых они записывают в HTML ответ соответствующий им HTML код (это может быть либо просто HTML элемент, либо HTML разметка и JavaScript сценарий, который будет выполняться в браузере пользователя). Пример сформированной с помощью такого программного кода HTML страницы показан на рис. 4.4.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
                "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title></title></head>
<body>
```

```
      <form method="post" action="Default.aspx" id="form1">
<div class="aspNetHidden">
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value =
"/wEPDwUKMjA3NjE4MDczNmRkac991xFznSlVYNLzEFsRbgfOg/L25R1l2kE+IKay6z
E=" />
</div>
<div class="aspNetHidden">
<input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION"
value="/wEWAwLft+uZBQLs0bLrBgKM54rGBnoxLiVzlAgERzPVlucFSAiRiOV4mzbWN
jJorx56E003" />
</div>
       <div style="height: 85px">
          Введите свое имя:
          <input name="TextBox1" type="text" value="ваше имя" id="TextBox1" />
          <br />
          <input type="submit" name="Button1" value="Button" id="Button1" /><br />
       </div>
    </form>
</body>
</html>
```

Рис. 4.4. HTML код получаемый браузером пользователя

Как видно на данном рисунке, все серверные ЭУ записали вместо себя определенный HTML код. Например, вместо asp:TextBox записан элемент <input type="text…>; вместо asp:Button записан элемент <input type="submit"…>; а вместо asp:Label нет ничего, так он не показывается (у него задан атрибут Visible="False").

Кроме этого в данный HTML код включены два скрытых поля __VIEWSTATE и __EVENTVALIDATION. Значения этих полей не используется браузером при работе с данной страницей, а при обратной отправке на сервер опять отсылаются web-приложению.

Скрытое поле __VIEWSTATE содержит (в атрибуте value) состояния элементов управления перед отправкой результата обработки запроса пользователю (у всех ЭУ есть методы получения и сохранения своих состояний в этом поле) и состояние представления (ViewState). Состояния элементов используются для определения возникающих на стороне браузера событий при работе пользователей с ЭУ (например, Click или SelectedIndexChanged). А состояние представления является одной из возможностей сохранения состояния работы пользователя (дополнительно к возможностям объекта Session).

Скрытое поле __EVENTVALIDATION является методом обеспечения безопасности, которая предотвращает возможность специальной отправки клиентом злонамеренных запросов. При обработке запроса производится сравнение содержания запроса со значением атрибута value поля __EVENTVALIDATION, на предмет отсутствия дополнительных по-

лей, добавленных на стороне клиента. А также, выполняется контроль выбора значения из известного на сервере списка.

При работе пользователя с данной HTML формой могут возникать события вызывающие обратную отправку (***postback***) к той же самой web-форме. Такие события обычно возникают при нажатии пользователем кнопки типа “submit” или при выполнении специального скрипта вызывающего такое действие (JavaScript функция – __doPostBack()).

Программный код web-формы при получении обратного запроса определяет (путем сравнения текущих значений элементов управления и значений, сохраненных в скрытом поле __VIEWSTATE), какие действия выполнил пользователь в браузере с HTML формой (например, изменил текст в элементе типа text, или выбрал другой пункт option в элементе select, и т.п.) и инициирует соответствующие события для серверных элементов управления. После обработки этих событий (если соответствующие обработчики включены в программный код web-формы) формируется новая HTML страница, которая отправляется пользователю. И весь процесс повторяется заново.

Переход пользователей к работе с другими web-формами выполняется в результате обработки событий или с помощью специальных элементов управления.

Класс Page

Программный код web-форms представляет собой класс, производный от класса Page. Данный класс отвечает за выполнение всех обработчиков событий и генерацию HTML кода, передаваемого пользователю. Однако при этом, каждый элемент управления отвечает за формирование своей части HTML кода, описывающего именно его внутри HTML страницы ответа. Класс Page управляет процессом создания HTML страницы, вызывая у каждого элемента управления специальный метод RenderControl(), который выполняет вывод в формируемую HTML страницу своей части HTML кода (возможно и JavaScrip кода).

Класс Page также предоставляет с помощью своих свойств доступ к объектам контекста обработки запроса, которые создаются средой выполнения. С помощью этих свойств выполняется программирование логики работы web-приложения. В табл. 4.1 приведены основные свойства и методы объекта Page.

Таблица 4.1

Основные свойства и методы объекта Page

Имя свойства	Описание
Application	доступ к объекту класса HttpApplicationState, входящему в состав контекста, который хранит состояние приложения и данные доступные всем ее пользователям
Request	доступ к объекту класса HttpRequest, входящему в состав контекста, который предоставляет доступ к данным, описывающим запрос
Response	доступ к объекту класса HttpResponse, входящего в состав контекста, для записи данных в ответное сообщение сервера
Server	доступ к объекту класса HttpServerUtility, входящего в состав контекста, который содержит вспомогательные методы для обработки запроса
Session	доступ к объекту класса HttpSessionState, входящего в состав контекста, который дает возможность хранить в коллекции данные, доступные текущему сеансу работы пользователя
Cache	доступ к глобальному объекту класса Cache среды выполнения, позволяющему выполнить временное сохранение данных в кэше
ViewState	доступ к коллекции ViewState, в которой содержится текущее состояние представления выполняемой web-формы
Controls	доступ к коллекции элементов управления расположенных в web-форме
User	Объект, поддерживающий интерфейс IPrincipal, который содержит информацию о пользователе, выполнившем запрос

Классы Response и Request

Наиболее часто используемыми объектами, доступными в классе Page являются объекты Response и Request. Объект Request (экземпляр класса HttpRequest), входящий в состав контекста обработки HTTP запроса, содержит всю информацию, присланную браузером в обрабатываемом запросе. Свойства и методы Request позволяют решать задачи, связанные с аутентификацией пользователя, приема файлов от клиента, определение типа браузера клиента. Ниже приведен перечень основных свойств объекта Request:

- Browser – определение номера версии запрашивающего браузера, возможность поддержки им куки и другой служебной информации;

- Cookies – коллекция полученных в запросе объектов куки (объекты класса HttpCookie);
- Files – коллекция файлов, присланных пользователем (upload);
- QueryString – коллекция пар «имя – значение», содержащая все переменные строки запроса, присланной клиентом.

Для определения типа браузера клиента нужно использовать свойство Browser у объекта Request. Например, следующий код включает в формируемый HTTP-ответ тип браузера, приславшего запрос:

```
Response.Write("Ваш браузер – " + Request.Browser.Browser);
```

Объект Response (экземпляр класса HttpResponse) используется для формирования ответа web-приложения на запрос клиента. С помощью свойств и методов объекта Response можно управлять кэшированием сформированных HTML-страниц перед отправкой пользователю, изменять содержимое куки (объектов класса HttpCookie), читать и записывать неструктурированные данные, передаваемые пользователю. Основными свойстами и методами объекта Response являются следующие:

- свойство Cashe – позволяет выполнять кэширование ответов перед отправкой их пользователю;
- свойство Cookies – коллекция куки, которые будут передаваться пользователям в HTTP-ответах;
- метод Write() – позволяет выполнять вывод текста в формируемый HTML документ.

4.2. Серверные элементы управления

Серверные элементы управления (СЭУ) являются базовой частью архитектуры ASP.Net Web Forms. По существу, СЭУ являются классами библиотеки платформы .Net Framework, которые представляют собой визуальные элементы web-формы. Некоторые из этих классов являются достаточно простыми и непосредственно соответствуют конкретному HTML тэгу (HTML элементу управления). Другие СЭУ являются сложными и формируют набор HTML тэгов и скрипт на языке JavaScript.

Серверные элементы управления должны обязательно располагаться внутри HTML элемента form, имеющего атрибуты method="post" и runat="server" (т.е. данный элемент является серверным HTML элементом).

Все серверные элементы управления делятся на следующие группы:

- HTML элементы управления.
- Web элементы управления (или web ЭУ).
- Пользовательские элементы управления.

Серверные HTML элементы управления имеет прямое соответствие с обычными HTML тэгами и в основном предназначены для быстрого перехода от разработки простых HTML страниц с формами к web-формам.

Наиболее используемыми и обладающими наибольшие возможности являются серверные web-элементы управления.

В ASP.Net разработчик может создавать и собственные элементы управления (user server controls). Разработка и использование таких ЭУ в пособии не рассматривается. Полное описание создания пользовательских ЭУ приведено в [5, 9].

Серверные HTML элементы управления

Серверные HTML ЭУ используют такой же синтаксис, как и HTML тэги, только имеют специальный атрибут runat=server. В результате добавления такого атрибута, для HTML элемента создается объект соответствующего класса, имя которого совпадает со значением атрибута id и набором свойств, соответствующих HTML атрибутам тэга. Например, для тэга

```
<input id="myName" type="text" runat="server" />
```

создается объект с именем myName. Значение атрибута value данного тэга может быть задано следующим образом:

```
void Page_Load(object sender, EventArgs e) {
      myName.Value = "Петров";
}
```

После обработки web-формы объект myName сформирует следующий HTML код, который будет передаваться браузеру:

```
<input name="myName" id="myName" type="text" value="Петров" />
```

Список основных серверных HTML ЭУ, соответствующих им классов и их свойств, приведен в табл. 4.2.

Таблица 4.2

Список основных серверных HTML элементов управления

HTML тэг	Класс	Основные элементы класса
<a runat="server">	HtmlAnchor	HRef, Target, Title, Name, событие ServerClick
<button runat="server">	HtmlButton	CausesValidation, ValidationGroup, событие ServerClick
<form runat="server">	HtmlForm	Enctype, Method, Target, DefaultButton, DefaultFocus

Окончание табл. 4.2

HTML тэг	Класс	Основные элементы класса
<img runat="server">	HtmlImage	Align, Alt, Border, Height, Src, Width
<input type="button" runat="server">	HtmlInputButton	Type, Value, CausesValidation, ValidationGroup, событие ServerClick
<input type="reset" runat="server">	HtmlInputReset	Type, Value
<input type="submit" runat="server">	HtmlInputSubmit	Type, Value, CausesValidation, ValidationGroup, событие ServerClick
<input type="checkbox" runat="server">	HtmlInputCheckBox	Checked, Type, Value, событие ServerClick
<input type="file" runat="server">	HtmlInputFile	Accept, MaxLength, PostedFile, Size, Type, Value
<input type="image" runat="server">	HtmlInputImage	Align, Alt, Border, Src, Type, Value, CausesValidation, ValidationGroup, событие ServerClick
<input type="radio" runat="server">	HtmlInputRadioButton	Checked, Type, Value, событие ServerChange
<input type="text" runat="server">	HtmlInputText	MaxLength, Type, Value, событие ServerChange
<select runat="server">	HtmlSelect	Multiple, SelectedIndex, Size, Value, DataSource, DataTextField, DataValueField, Items (коллекция), событие ServerChange
<table runat="server">, <td runat="server">	HtmlTable	Align, BgColor, Border, BorderColor, CellPadding, CellSpacing, Height, NoWrap, Width, Rows (коллекция)
<th runat="server">	HtmlTableCell	Align, BgColor, Border, BorderColor, ColSpan, Height, NoWrap, RowSpan, VAlign, Width
<tr runat="server">	HtmlTableRow	Align, BgColor, Border, BorderColor, Height, VAlign, Cells (коллекция)
<textarea runat="server">	HtmlTextArea	Cols, Rows, Value, событие ServerChange
Любой другой HTML тэг с атрибутом runat="server"	HtmlGenericControl	Нет

Серверные HTML ЭУ поддерживают два типа событий: ServerClick (возникает при нажатии на ЭУ) и ServerChange (возникает при изменении содержания ЭУ). Связывание событий серверного HTML ЭУ выполняется с помощью атрибута OnServerClick или OnServerChange, которому присваивается имя метода-обработчика события. Обработчик события имеет стандартную сигнатуру (такую же, как и для Windows ЭУ). Например, для следующего ЭУ задан обработчик события с именем myName_change:

```
<input id="myName" runat="server" type="text"
                                OnServerChange = "myName_change"/>
```

В файле программного кода должен быть создан метод, имеющий следующий вид:

```
protected void myName_change(object sender, EventArgs e) {
      Response.Write("Введено имя = " + myName.Value);
}
```

Серверные web элементы управления

Наиболее часто используемыми серверными ЭУ являются серверные web-ЭУ. Они имеют больше возможностей для создания пользовательского интерфейса и организации взаимодействия пользователей с web-формой. Всего в ASP.Net имеется более 80 таких элементов управления. Серверные элементы управления в шаблоне web-формы (файл *.aspx) задаются с помощью специальных серверных тэгов, имеющих в простом случае следующий вид:

```
<asp:[имя_эл-та] id= [имя] [атр1]=[знач1] [атр2]=[знач2] … runat="server" />
```

У всех серверных ЭУ имеется префикс “asp:”, который указывает, что данный тэг является не HTML тэгом, а обрабатывается на стороне web-сервера. Атрибуты ЭУ соответствуют свойствам и событиям объектов, которые для них создаются. Запись атрибутов должна соответствовать следующим основным требованиям:

- для каждого серверного ЭУ должны быть заданы атрибуты id = “имя” и runat="server";
- атрибуты разделяются пробелами и порядок их следования произвольный;
- атрибуты, соответствующие событиям имеют название «on[*имя_события*]» и их значением является название метода, выполняющего обработку данного события.

Например, описание серверного элемента Button (кнопка), вставленного в шаблон формы будет выглядеть следующим образом:

```
<asp:Button id="Button1" text="Button" onclick="Button1_Click" runat="server" />
```

Как видно из этого примера, после префикса asp: следует тип элемента управления Button, который соответствует классу Button, описанному в .NET Framework; значение атрибута id задает имя объекта (Button1), с помощью которого к нему можно обращаться в программном коде; атрибута text используется для задания соответствующего свойства у объекта Button1, а с помощью атрибута onclick задется имя метода, выполняющего обработку события click (щелчок).

Все обработчики события для ЭУ должны иметь стандартную сигнатуру (такую же, как и для Windows ЭУ). Например, обработчик событий для нажатия кнопки может иметь следующий вид:

```
protected void Button1_Click(object sender, EventArgs e) {
    Label1.Text = "Добрый день " + TextBox1.Text + "!";
    Label1.Visible = true;
}
```

Здесь параметр sender это ссылка на объект, инициирующий данное событие (объект Button1), а e – параметры события (в данном случае не используются).

Все серверные ЭУ, а также и класс Page являются производными от базового класса Control из пространства имен System.Web.UI, поэтому все они наследуют его свойства и методы (табл. 4.3).

Таблица 4.3

Основные свойства и методы базового класса Control

Свойство	**Описание**
ID	Имя ЭУ, с помощью которого к нему можно обращаться.
Page	Ссылка на объект Page, который содержит данный ЭУ.
Parent	Ссылка на родительский объект (Page или Control) который содержит данный ЭУ.
Visible	Логическая переменная, которая определяет, показывать ли данный ЭУ на форме; если значение false, то ЭУ не видим на данной странице.
Controls	Коллекция ЭУ, расположенных на web-форме, или в ЭУ. Элемент управления (например, Panel), содержащийся в данной коллекции сам может включать свои дочерние ЭУ.
BackColor	Получить или задать фоновый цвет.
BorderColor	Получить или задать цвет рамки.

Окончание табл. 4.3

BorderStyle	Одно из значений перечисления BorderStyle: Dashed, Dotted, Double, Groove, Ridge, Inset, Outset, Solid и None.
BorderWidth	Получить или задать ширину границы.
CssClass	Задает класс CSS стиля, используемого данным ЭУ.
Enabled	Получить или задать состояние ЭУ. Если false, то элемент не активный и показывается серым цветом.
Font	Возвращает объект с информацией о шрифте, используемом для ЭУ, показывающего текст.
ForeColor	Получить или задать цвет ЭУ.
Tooltip	Получить текстовое сообщение, которое отображается когда пользователь проводит мышью над данным ЭУ.
Height	Получить или задать высоту ЭУ.
Width	Получить или задать ширину ЭУ.
Метод	**Описание**
FindControl()	Поиск дочернего ЭУ с заданным именем в текущем ЭУ и во всех содержащихся в нем ЭУ. Если такой дочерний ЭУ будет найден, то возвращается ссылка общего типа, которую нужно преобразовать к требуемому типу.
HasControls()	Проверка, имеет ли данный ЭУ дочерние ЭУ. Данный ЭУ должен быть контейнерным ЭУ, чтобы иметь дочерние ЭУ (например, таким, как Panel).
DataBind()	Связывание ЭУ и все его дочерние ЭУ с заданным источником данных.
RenderControl()	Формирование HTML кода для данного ЭУ на основе текущего состояния. Этот метод напрямую не вызывается, а его вызывает объект Page, при создании HTML кода всей страницы.

Обработка событий элементов управления

У каждого серверного ЭУ имеется связанное с ним событие. При выполнении пользователем каких-то действий, с соответствующим ему HTML кодом, инициируются события. При этом выделяются следующие типы событий:

- ***события-действия***, которые вызывают немедленную обратную отправку запроса на web-сервер, например: событие Click у ЭУ Button;

- ***события-изменения***, которые по умолчанию не вызывают немедленную обратную отправку запроса на web-сервер, примерами таких событий являются TextChanged (изменение текста в TextBox), CheckedChanged (изменения состояния элемента CheckBox) или SelectedIndexChanged (изменение выбранного пункта в элементе ListBox или ComboBox).

События изменения не вызывают обратной отправки на web-сервер для немедленной обработки. Их обработка будет выполняться после того, как произойдет событие действие (по умолчанию это нажатие кнопки типа submit). Однако при обработке событий на сервере вначале выполняется обработка событий-изменений (не в порядке их появления в браузере, а в порядке следования соответствующих ЭУ в шаблоне), и только после этого обрабатывается событие-действие.

Событие-изменение элемента управления можно сделать событием-действием. Для этого в описании данного ЭУ нужно задать атрибут AutoPostBack="true". В этом случае при возникновении в браузере события-изменения будет выполняться немедленная обратная отправка (это делается с помощью скрипта, который вызывает отправку данных формы на сервер). Например, если в описании ЭУ TextBox включен атрибут AutoPostBack="true":

```
<asp:TextBox ID="TextBox1" runat="server" AutoPostBack="True">
        Иванов Петр </asp:TextBox>
```

то при изменении содержания данного элемента и переходе к другому ЭУ будет выполняться событие-действие, приводящее к обратной отправке запроса на сервер.

Классификация элементов управления

В ASP.Net имеется достаточно много серверных web-ЭУ. Невозможно быстро освоить все серверные ЭУ. В связи с этим важно понимать их классификацию и возможные области применения. Серверные web-ЭУ можно классифицировать по их форме и по выполняемым функциям.

По форме все серверные ЭУ можно разделить на простые и составные.

По выполняемым функциям серверные web-элементы управления можно разделить на следующие группы:

- ***стандартные элементы управления***, предназначенные для создания базового интерфейса пользователей (standart controls);
- ***элементы управления для выполнения навигации*** по web-приложению, позволяющие организовывать переходы пользователей

между web-формами приложения для решения различных задач (navigation controls);

- ***проверочные элементы управления***, позволяющее выполнять проверку вводимых пользователями данных на стороне клиента и сервера (validation controls).

- ***элементы управления для работы с данными***, позволяющие показывать и организовывать работу пользователей с базами данных (data controls);

- ***элементы управления поддержки безопасности работы пользователей*** с web-приложением, позволяющие выполнять аутентификацию и авторизацию пользователей (login controls);

Каждый web-элемент управления в результате обработки web-формы формирует HTML код, который включается в ответ пользователю, а также он имеет набор свойств и событий.

Описание некоторых стандартных web ЭУ приведено в табл. 4.4.

Таблица 4.4

Основные классы стандартных web ЭУ

Тэг серверного ЭУ	**Сформированный HTML код**	**Основные свойства и события**
<asp:Button>	<input type="submit"/> или	Text, CausesValidation, PostBackUrl, ValidationGroup; **событие:** Click
<asp:CheckBox>	☐	AutoPostBack, Checked, Text, TextAlign; **событие:** CheckedChanged
<asp:FileUpload>	<input type="file">	FileBytes, FileContent, FileName, HasFile, PostedFile; **метод:** SaveAs()
<asp:Image>	<img/>	AlternateText, ImageAlign, ImageUrl
<asp:ImageButton>	<input type="image"/>	CausesValidation, ValidationGroup; **событие** Click
<asp:ImageMap>	<map>	HotSpotMode, HotSpots (коллекция), AlternateText, ImageAlign, ImageUrl
<asp:HiddenField>	<input type="hidden">	Value
<asp:HyperLink>	<a>...	ImageUrl, NavigateUrl, Target, Text
<asp:Label>	...	Text, AssociatedControlID
<asp:LinkButton>	<a><img/>	Text, CausesValidation, ValidationGroup; **событие** Click

Окончание табл. 4.4

Тэг серверного ЭУ	Сформированный HTML код	Основные свойства и события
<asp:Panel>	...	BackImageUrl, DefaultButton, GroupingText, HorizontalAlign, Scrollbars, Wrap
<asp:RadioButton>	☐	AutoPostBack, Checked, GroupName, Text, TextAlign; **событие** CheckedChanged
<asp:HiddenField>	<input type="hidden">	Value
<asp:HyperLink>	<a>...	ImageUrl, NavigateUrl, Target, Text
<asp:Label>	...	Text, AssociatedControlID
<asp:LinkButton>	<a><img/>	Text, CausesValidation, ValidationGroup; **событие** Click
<asp:Panel>	...	BackImageUrl, DefaultButton, GroupingText, HorizontalAlign, Scrollbars, Wrap
<asp:RadioButton>	☐	AutoPostBack, Checked, GroupName, Text, TextAlign; **событие** CheckedChanged
<asp:TextBox>	____ или <textarea> ...</textarea>	AutoPostBack, Columns, MaxLength, ReadOnly, Rows, Text, TextMode, Wrap; **событие** TextChanged

Составные элементы управления

Стандартные серверные ЭУ описываются в виде элементов, которые не включают подэлементы. Но кроме этого есть составные ЭУ, которые включают подэлементы, описывающие данные и шаблон их отображения. Примером составных ЭУ являются ***списочные ЭУ*** (list contros), которые показывают заданный список элементов данных или набор элементов из источника данных (записей) с использованием для каждого из них фиксированного шаблона (ListBox, BulletedList, CheckBoxList, RadioButtonList, DropDownList).

Все списочные ЭУ наследуются от базового класса ListControl. Они предоставляют фиксированный, встроенный шаблон отображения элементов данных. К списочным относятся следующие ЭУ:

- ListBox – одновременно показываемый список элементов, из которых пользователь может выбрать одно или несколько значений;
- DropDownList – выпадающий список элементов (комбо-бокс);

- CheckBoxList – список флажков, у которых пользователь может задать состояние «выбран/не выбран»;
- RadioButtonList – список альтернатив из которого пользователь может выбрать одно значение;
- BulletedList – маркированный список элементов.

Отображение каждого элемента данных определяется типом списочного ЭУ. Например, маркированный список может быть описан следующим образом:

```
<asp:BulletedList runat="server" bulletstyle="Square">
   <asp:ListItem>Раз</asp:ListItem>
   <asp:ListItem>Два</asp:ListItem>
   <asp:ListItem>Три</asp:ListItem>
</asp: BulletedList>
```

Далее в пособии будут рассматриваться и другие составные ЭУ, в особенности при пояснении работы с данными, с помощью таких ЭУ, как GridView и ListView.

Проверочные элементы управления

Многие элементы управления используются для ввода данных, которые затем могут сохраняться в БД на сервере. Одним из важных этапов получения данных от пользователя является выполнение проверки их правильности, для исключения передачи на сервер неправильных данных.

Критерии проверки могут быть самыми разными, начиная с того, вводились ли данные пользователем вообще и заканчивая проверкой типа данных. В основном проверка вводимых данных осуществляется на стороне клиента, так как в этом случае он может сразу же, еще до отправки данных на сервер, получить уведомление о содержащихся ошибках. Однако, независимо от того, осуществлялась ли проверка вводимых данных на стороне клиента, также выполняется проверка и на стороне сервера.

В ASP.Net имеется группа элементов управления, предназначенных для проверки вводимых данных, так называемых «***верификаторов***». Эти элементы могут быть связаны с любым элементу управления, выполняющим ввод данных. После связывания, верификатор выполняет автоматическую проверку вводимых данных на стороне клиента и сервера (в процессе жизненного цикла web-формы). Если данные, вводимые в элемент ввода данных, не удовлетворяют условию, заданному в верификаторе, то верификатор препятствует отправке данных web-страницы на сервер.

В ASP.Net имеются следующие элементы управления, предназначенных для осуществления проверки вводимых данных:

- RequiredFieldValidator – проверяет наличие введенных данных в элемент управления;
- RangeValidator – проверяет нахождение значения элемента управления в пределах заданного диапазона
- RegularExpressionValidator – определяет соответствие значения данного элемента управления определенному регулярному выражению;
- CompareValidator – сравнивает значение текущего элемента управления с константой или значением другого элемента управления;
- CustomValidator – выполняет заданную операцию проверки достоверности на стороне клиента, либо на стороне сервера для реализации собственной логики проверки вводимых данных;
- ValidationSummary – отображает информацию на странице, либо во всплывающем окне с сообщениями об ошибках для каждого элемента управления, проверка которого завершилась ошибкой.

Допускается использование нескольких элементов управления проверкой ввода данных, связанных с одним элементом ввода данных. С помощью верификаторов можно проверить вводимые данные в такие элементы управления как TextBox, ListBox, DropDownList, RadioButtonList, HtmlInputText, HtmlTextArea, HtmlSelect.

Верификаторы добавляют скрипт для проверки вводимых данных на стороне клиента. При успешной проверке данных на стороне клиента, подобная проверка будет выполнена повторно и на стороне сервера.

Самым простым валидатором является RequiredFieldValidator. Он предназначен для проверки того введены ли данные в связанное с ним поле или нет. Это значит, что если в соответствующем текстовом поле ввода не будет символов отличных от пробелов, возникнет ошибка ввода. В качестве демонстрации работы данного валидатора создадим следующую страницу:

```
<body>
<form id="form1" runat="server">
   <div>
      <asp:Label ID="Label1" runat="server" Text="Имя"></asp:Label>
      <asp:TextBox ID="tb_Name" runat="server"></asp:TextBox>
      <asp:RequiredFieldValidator ID="RequiredFieldValidator1"
         runat="server" ErrorMessage="Необходимо ввести имя пользователя"
         ControlToValidate="tb_Name">*</asp:RequiredFieldValidator><br />
      <br />
      <asp:Button ID="btn_OK" runat="server" Text="OK" Width="89px" />
   </div>
```

```
</form>
</body>
```

Если при нажатии на кнопку OK поле ввода имени пользователя будет пустым, рядом с полем ввода появится символ красной звездочки. Нетрудно заметить, что при этом, обратная отправка данных на сервер не производится (рис. 4.5).

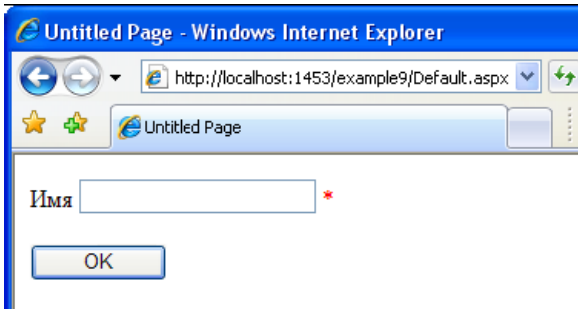


Рис. 4.5. Отображение страницы, содержащей валидатор в окне браузера

Для одного и того же ЭУ можно задать несколько проверочных элементов. Для отображения итоговой информации о результатах всех проверок вводимой информации, необходимо использовать элемент управления ValidationSummary. Он выводит значение ErrorMessage каждого валидатора, для которого эта проверка завершилась неудачей. Итоговая информация может отображаться либо на странице, либо в отдельном окне. Для указания данного режима необходимо установить свойства ShowMessageBox и ShowSummary. При значении свойства ShowMessageBox=true, сообщение выводится в отдельном окне, если же задано свойство ShowSummary=true, то данное сообщение выводится на самой странице. При отображении итоговой информации на странице можно задать некоторые дополнительные параметры с помощью свойства DisplayMode, а также задать заголовок для итоговой информации с помощью свойства HeaderText. Пример использования элемента управления ValidationSummary проказан ниже:

```
<asp:ValidationSummary ID="ValidationSummary" runat="server"
   DisplayMode = "List" HeaderText="В результате проверки выявлены
   следующие ошибки:" />
```

4.3. ASP.Net web-приложение

В самом общем виде, среда выполнения ASP.Net web-приложений является системой обработки HTTP запросов (ASP.NET HTTP runtime), которая взаимодействует с web-сервером с помощью ISAPI расширения. Она получает от web-сервера приходящий HTTP запрос и передает его внутреннему последовательному процессу обработки (конвейеру). При

выполнении различных шагов данного процесса инициируются различные события, для которых разработчик может создать обработчики и подключить требуемый программный код.

Среда выполнения отвечает за маршрутизацию запросов через процесс обработки. Несколько взаимосвязанных объектов, расширенных с помощью создания производных классов или с помощью интерфейсов, доступны для настройки на решение конкретной задачи. Система обработки ASP.Net полностью написана в виде управляемого кода (managed code) и все расширения также являются управляемым кодом.

Контекст web-приложения

При обработке запросов к web-приложению среда выполнения создается его контекст – набор объектов, которые могут использоваться в web-приложении. Описание основных классов, объекты которых составляют контекст web-приложения, приведен в табл. 4.5.

Таблица 4.5

Классы контекста web-приложения

Класс	Описание
HttpApplicationState	объекты данного класса хранят состояние приложения и данные доступные всем пользователям приложения
HttpRequest	объекты данного класса сдержат всю информацию о текущем HTTP запросе и предоставляют методы работы с ней
HttpResponse	объекты данного класса содержат код формируемой HTML страницы ответа и предоставляют методы работы с ним
HttpServerUtility	объекты данного класса предоставляют вспомогательные методы для обработки запроса
HttpSessionState	объекты данного класса содержат данные пользователя доступные в текущем сеансе работы
IPrincipal	интерфейс к объекту, который содержит информацию о текущем пользователе (если она поддерживается)
HttpContext	объекты данного класса содержит всю информацию о контексте выполнения запроса
Cache	глобальные объекты данного класса позволяют выполнять временное хранение данных в кэше; среда выполнения сама определяет, когда эти данные могут быть удалены
ProfileCommon	объекты данного класса содержат информацию специфичную для текущего пользователя (если она поддерживается)

Доступ к объектам контекста выполняется в основном с помощью свойств класса Page.

События web-приложения

При обработке запроса к web-приложению, среда выполнения инициирует большой набор событий:

- события, связанные с web-приложением и сеансом работы пользователя;
- события, связанные с web-формой в целом (события класса Page, жизненный цикл страницы);
- события, связанные с серверными ЭУ (события соответствующих им классов, жизненный цикл ЭУ).

Для всех событий можно написать обработчики – методы, которые будут вызываться средой выполнения web-приложения при наступлении данного события.

С web-приложением в целом связано большое количество событий, которые делятся на два типа:

- события, возникающие в специальных случаях, как например: начало и окончание работы web-приложения (Application_Start и Application_End) и начало и окончание сеанса работы пользователя (Session_Start и Session_End);
- события, возникающие при обработке каждого HTTP запроса (таких событий более 20): например, BeginRequest, AuthenticateRequest, EndRequest и т.п.

Для создания обработчиков таких событий нужно в web-приложение добавить файл global.ascx (команда Website=>AddNewItem), и включить в него методы – обработчики событий. Заготовки обработчиков событий для специальных случаев уже включены в файле global.ascx, а обработчики событий для HTTP запросов можно добавить. Связь события с обработчиком выполняется по имени метода обработчика. Методы обработчики должны иметь специальные имена следующего вида: Application_OnXxxx(), где Xxxx – название обрабатываемого события. Например, обработчик события окончания обработки HTTP запроса EndRequest имеет следующий вид:

```
protected void Application_OnEndRequest() {
   Response.Write("<hr />Обработка HTTP запроса закончилась в " +
            DateTime.Now.ToString());
}
```

При обработке запроса к web-форме среда выполнения также инициирует набор событий. Для обработки таких событий можно создать методы в программном коде web-формы (в файле *.aspx.cs).

Прежде всего, при создании объекта класса Page возникает событие Page_Init. В обработчике данного события можно записать код, выполняющий начальную инициализацию страницы. Однако, данное событие нельзя использовать для инициализации элементов управления, размещенных на форме, так как они еще не созданы. После этого инициируется событие Page_Load. Большинство web-форм используют это событие для заполнения полей данными и задания начальных свойств элементам управления. В процедуре обработки данного события можно определить, была ли данная форма вызвана впервые, или обращение к ней выполняется повторно в результате обратной отправки (postback), произошедшей при нажатии пользователем кнопки, либо другого элемента управления, размещенного на странице. Для этого нужно проверить свойство IsPostBack объекта Page, которое будет иметь значение false при первом вызове данной страницы и true при обратных отправках. Задание начальных значений элементам web-формы выполняется только при первом вызове формы. При последующих вызовах страницы в результате обратной отправки значения ЭУ задаются автоматически на основе данных из скрытого ЭУ __VIEWSTATE, содержащего сериализацию всех значений ЭУ.

На основе сравнения исходных и новых значений ЭУ, переданных в HTTP запросе инициируются события для серверных ЭУ, размещенных на странице. Для них в программном коде web-формы также можно создать обработчики событий.

После вызова события Page_Load происходит так называемая проверка достоверности страницы. Необходимость такой проверки возникает тогда, когда пользователь ввел в элементы управления, расположенные на странице данные, которые впоследствии необходимо сохранить или использовать для обработки. В идеале проверка достоверности должна происходить на стороне клиента для того, чтобы пользователь был проинформирован о проблемах с вводом данных перед их отправкой на сервер, т.к. это позволяет уменьшить объем информации, передаваемой по сети и ускорить процесс обмена данными с сервером. Однако, независимо от того, была ли произведена проверка достоверности данных на стороне клиента или нет, ее необходимо осуществлять и на стороне сервера. В том случае, если проверка достоверности выявила ошибки во введенных данных, ASP.Net уведомит об этом пользователя

и не позволит осуществлять дальнейшую работу со страницей до устранения ошибок.

Следующим шагом обработки web-формы является обработка всех событий, инициированных пользователем с момента последней обратной отправки. Например, если в web-форме имеется кнопка Button и текстовым полем TextBox (со значением свойства AutoPostBack равным в false, заданным по умолчанию). При изменении текста в текстовом поле и нажатии кнопки инициируется обратная отправка данных страницы на сервер. В момент обработки данной формы средой выполнения инициируют следующие события:

1. Page.Init
2. Page.Load
3. TextBox.TextChanged
4. Button.Click
5. Page.PreRender
6. Page.Unload

Событие PreRender инициируется после того, как сервер обработал все события страницы, но генерация ее HTML кода еще не произошла. Обычно это событие используется ASP.Net для связывания элементов управления с источником данных непосредственно перед созданием HTML кода и отправкой его клиенту. После этого будет формироваться HTML код, который отправляется браузеру пользователя.

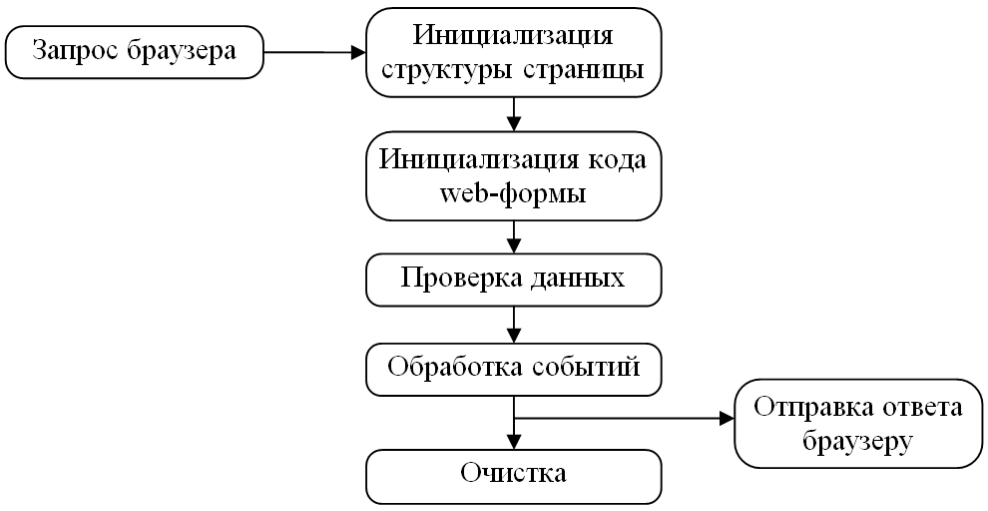


Рис. 4.8. Жизненный цикл web-формы ASP.Net

Обработка запроса заканчивается освобождением всех ресурсов, занятых данной web-формой (этап «Очистка»), при этом инициируется событие Page_Unload.

Описанная последовательность событий представляет собой жизненный цикл web-формы, который показан на рис.4.8.

Структура каталога приложения

Каталог (папка) web-приложения может содержать большое количество используемых ресурсов, таких, как скомпилированные сборки, файлы данных, таблицы стилей, изображения, XML файлы и т.д.

В табл. 4.6 перечислены основные подкаталоги, которые обычно создаются для ASP.Net web-приложения.

Таблица 4.6

Основные подкаталоги ASP.Net приложения

Подкаталог	Описание
Bin	Содержит все предварительно скомпилированные сборки .NET, которые обычно представляют собой DLL библиотеки.
App_Code	Содержит классы исходного кода, динамически компилируемые для использования в рамках приложения. Обычно эти файлы кода представляют собой отдельные компоненты, такие как библиотеки доступа к данным, web-сервисы и т.п.
App_Data	Содержит файлы данных, включая XML файлы и файлы SQL Express.
App_Themes	Содержит файлы с темами, используемыми web-приложением.
<собственные подкаталоги>	Подкаталоги, в которых хранятся web-формы для выполнения специфических задач web-приложения.

Для задания пути к файлам виртуального каталога используется префикс "~/". Например, "~/Default.aspx" (файл из того же самого подкаталога) или "~/Styles/Site.css" (файл из подкаталога более низкого уровня – Style)

Конфигурирование ASP.Net-приложения

Задание параметров работы среды выполнения, и различных данных, требуемых для работы самого web-приложения, называется конфигурированием web-приложения. В ASP.Net конфигурирование выполняется с помощью набора XML-файлов конфигурации, которые наследуются друг от друга. Каждый XML-файл содержит набор установо - параметров работы web-приложения. Наследование файлов конфигурации означает, что дочерний файл конфигурации использует все установки, которые сделаны в родительском файле, но установки дочернего файла

заменяют аналогичные установки, сделанные в родительском файле. Пример наследования файлов конфигурирования показан на рис. 4.9.

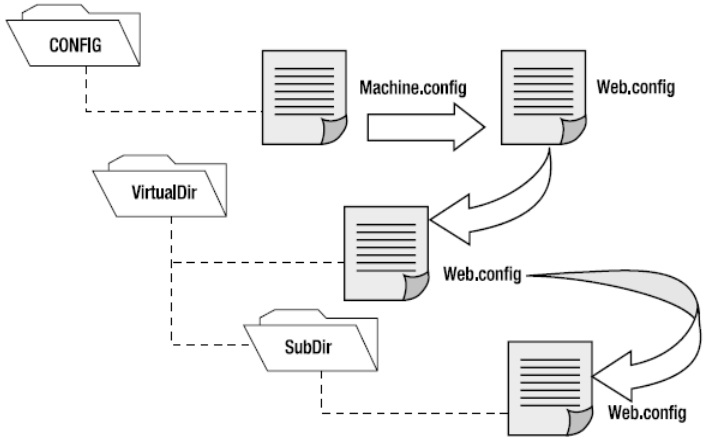


Рис. 4.9. Наследование файлов конфигурации

Конфигурирование начинается с файла machin.config, хранящемся в системном каталоге c:\Windows\Microsoft.NET\Framework\[версия]\Config, в котором задаются параметры запуска и функционирования среды выполнения. Следующим за machin.config является конфигурационный файл web.config (в том же самом системном каталоге) который содержит дополнительные установки, применяемые для всех ASP.Net приложений web-сервера. Все web-приложения наследуют установки из этих двух файлов. Каждое приложение также имеет собственные файлы конфигурации web.config. Один файл должен быть включен в корневой виртуальный каталог web-приложения. Кроме этого, для задания специфических установок для подкаталогов (например, для задания прав доступа к размещенным в них web-формам) в них также включаются свои файлы web.config, установки которых применимы только для данных подкаталогов. Самый простой файл конфигурации имеет следующий вид:

```
<?xml version="1.0"?>
<configuration>
   <system.web>
      <compilation debug="true" targetFramework="4.0" />
   </system.web>
</configuration>
<system.web>
```

Элемент <system.web> содержит все специфические для ASP.Net приложения установки. Эти установки используются сервером ASP.Net приложений для задания особенности работы данного web-приложения.

В табл. 4.7 приведены основные дочерние элементы, которые могут быть включены в элемент <system.web> и цель их использования.

Таблица 4.7

Некоторые основные элементы файла конфигурации

Элементы	Описание того, что содержится в элементе
authentication	Способ аутентификации пользователей, запрашивающих формы приложения.
authorization	Правила доступа пользователей к ресурсам web-приложения или текущего каталога (с помощью подэлементов deny и allow).
compilation	Используемая версия платформы .NET (с помощью атрибута targetFramework) и требуется ли генерировать отладочную информацию при компиляции форм (с помощью атрибута debug), чтобы можно было отлаживать web-приложение в Visual Studio. Кроме этого, могут указываться с помощью подэлементов <assemblies> дополнительные сборки, которые может использовать web-приложение из каталога Bin или GAC).
customErrors	Задание адреса специальной web-формы, которая будет показываться при возникновении ошибок (например, при ошибке 404 (страница не найдена).
membership	Конфигурирование средств управления информацией об учетных записях пользователей (membership).
pages	Задание значений параметров по умолчанию для web-форм приложения (они могут переопределяться в web-формах с помощью директив Page).
profile	Конфигурирование средств поддержки специфической информации о пользователях приложения.
roleManager	Конфигурирование средств работы с ролями пользователей для обеспечения безопасности приложения.
sessionState	Конфигурирование средств поддержки состояния сеанса работы пользователей.
trace	Задание параметром отладочной трассировки работы web-приложения, которая позволяет просматривать диагностическую информацию на странице (или собирать ее отдельно).

Кроме элемента <system.web> в конфигурационный файл могут также включаться следующие элементы:

- <connectionStrings> – для описания набора строк соединения с базами данных;
- <appSettings> – для задания любых параметров, требуемых для настройки логики работы web-приложения.

В следующем примере показано описание строки соединения с именем MyConnString и параметра приложения с именем MyParam1:

```
<connectionStrings>
   <add name="MyConnString"
        connectionString="Data Source=localhost\sqlexpress;
        Initial Catalog=aspnetdb; Integrated Security=True"
        providerName="System.Data.SqlClient"/>
</connectionStrings>
<appSettings>
   <add key="MyParam1" value="xxxxx" />
</appSettings>
```

Установка web приложения и организация виртуальных каталогов приложений

Для работы ASP.Net приложения в реальных условиях (а не в среде разработки) необходимо разместить все его файлы (web-формы, файлы конфигурации, программный код и различные ресурсы) в виртуальном каталоге на web-сервера IIS и указать, что это web-приложение. Для этого нужно для данного каталога выполнить команду (с помощью контекстного меню) – «Преобразовать в приложение» (иконка данного виртуального каталога сменится с  на ).

После этого при обращении пользователей к ресурсам данного виртуального каталога web-сервер будет запускать среду выполнения (ASP.Net Application Server) и запускать в *домене приложения* требуемый программный код.

Web приложения, расположенные в разных виртуальных каталогах, будут выполняться в разных доменах.

Создание web-приложений с помощью Visual Studio

Обычно web-приложения ASP.Net разрабатываются с помощью интегрированной среды разработки Microsoft Visual Studio. Данная система включает средства управления проектами, редактор исходного текста, конструкторы пользовательского интерфейса, мастера, компиляторы, компоновщики, инструменты, утилиты, документацию и отладчики.

В Visual Studio предлагается два способа создания web-приложений, с использованием технологии ASP.NET Web Forms:

- Разработка с использованием проекта (проектная разработка). Аналогично проектам создания Windows приложений. Visual Studio генерирует файл проекта с расширением .csproj (если код пишется на языке C#), в котором фиксируются составляющие проект файлы и сохраняются некоторые отладочные параметры. При запуске Web-проекта,

прежде чем запустить web-браузер, Visual Studio сначала компилирует весь написанный разработчиком код в одну сборку.

- Разработка без использования проекта (создание web-сайта). Это альтернативный подход, который подразумевает создание просто web-сайта безо всякого файла проекта. При таком подходе Visual Studio предполагает, что каждый файл в каталоге web-сайта (и всех его подкаталогах) является частью web-приложения. В этом случае Visual Studio не требуется предварительно компилировать код. Вместо этого ASP.NET компилирует уже сам web-сайт при первом запросе какой-нибудь входящей в его состав формы.

В обоих способах имеется два основных шаблона проекта:

- ASP.Net Empty Web Application (или Site) – проект и web-сайт содержат только один файл – web.config, другие файлы нужно создавать по мере разработки web-приложения;
- ASP.Net Web Application (или Site) – проект и web-сайт содержат полный каркас web-приложения, включающий:
  - ✓ мастер-страницу (site.master) и два файла со страницами содержания (default.aspx и about.aspx), которые основываются на данной мастер-странице;
  - ✓ файл конфигурации web.config;
  - ✓ файл с заготовками для отработчиков событий, связанных с web-приложением global.asax;
  - ✓ подкаталог Account, содержащий формы для создания и ведения учетных записей пользователей;
  - ✓ подкаталог Scripts с файлами, содержащими библиотеку скриптов jquery, которая поставляется с Visual Studio;
  - ✓ подкаталог Styles с файлом, содержащим первоначальную каскадную таблицу стилей Site.css.

Рассмотрим процесс создание нового пустого web-приложения без использования проекта. Для этого следует выполнить команду «File=>New Web Site=>ASP.Net Empty Web Site». После задания местоположения и имени папки Visual Studio создаст новое web-приложение. Данное web-приложение не будет содержать ни одной web-формы. Для добавления web-формы нужно выбрать команду «Web Site=>Add New Item», выбрать шаблон нового файла “Web Form” и задать ему имя. У первой web-формы – начальной страницы, лучше выбрать имя, используемое web-сервером по умолчанию – Default.aspx.

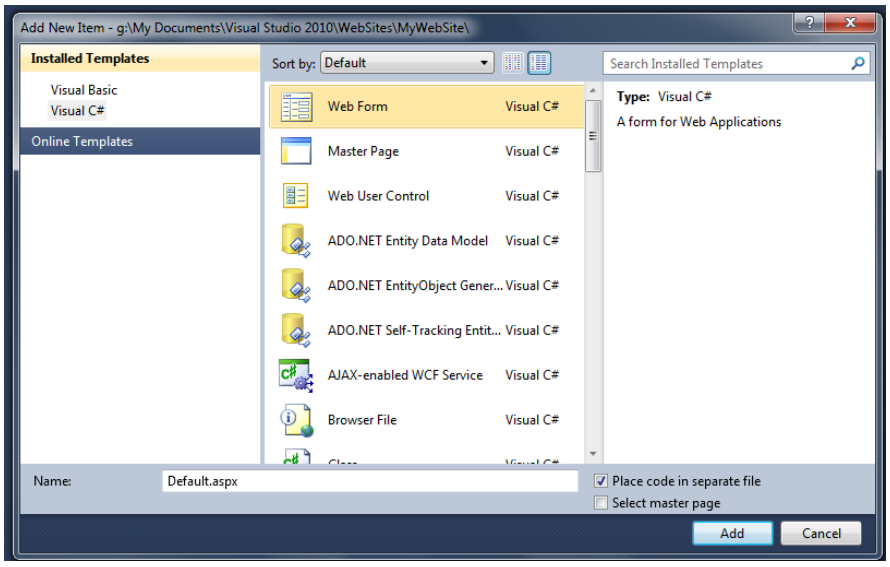


Рис. 4.10. Добавление новой web-формы к web-приложению

В результате выполнения данной команды будет создан файл с шаблоном web формы (Default.aspx) и файл с программным кодом, связанный с данным шаблоном (Default.aspx.cs). После этого шаблон web-формы можно просматривать и редактировать в двух основных режимах ***design*** (графический режим) и ***source*** (режим исходного кода). Для отображения web-формы в одном из режимов ее просмотра нужно щелкнуть по имени web-формы в окне «Solution Explorer», а затем в нижей части окна редактирования выбрать режим: Design, Source или Split (для отображение формы в обоих режимах).

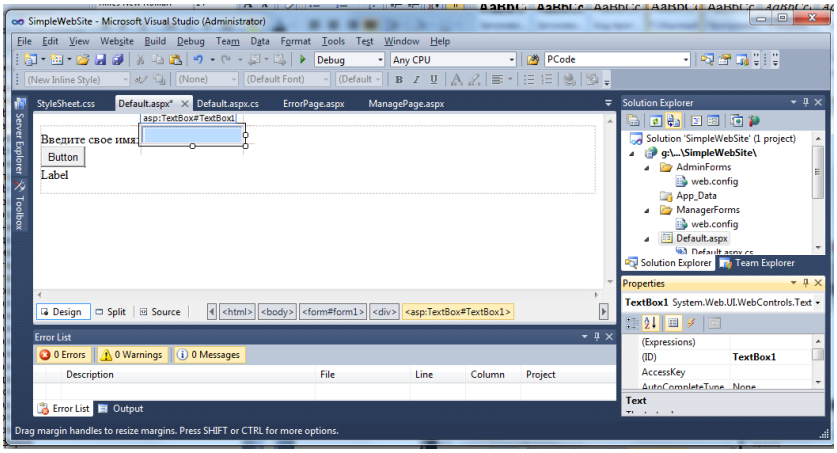


Рис. 4.11. Отображение формы, показанной на рис. 4.2 в графическом режиме редактирования

В графическом режиме шаблон web формы отображается в таком виде, как он будет показываться в браузере пользователя (рис. 4.11). С использованием данного режима на форме можно очень просто разме-

щать серверные ЭУ, задавать их начальные параметры и создавать заготовки для обработчиков событий.

Однако, не все задачи разработки web-формы можно решить в графическом режиме. Поэтому часто требуется редактировать web-документы в режиме исходного кода (рис. 4.12). Если хорошо знать HTML, то этот способ может оказаться даже удобнее использования графических инструментов. Поддерживаемая Visual Studio технология IntelliSense помогает завершать написание HTML элементов, создаваемых в режиме исходного кода.

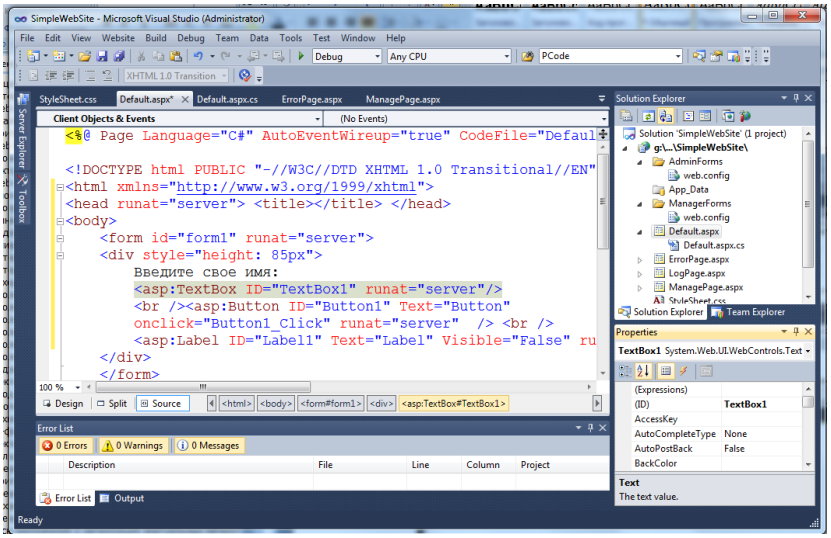


Рис. 4.12. Отображение формы, показанной на рис. 4.11 в режиме исходного кода

При работе в этих режимах редактирования web-формы на ней можно размещать серверные ЭУ, которые имеются в инструментальной панели (Toolbox), с использованием стандартной операции «перетащи и урони» (drag-and-drop), как это обычно делается при проектировании Windows-форм. После добавления ЭУ в форму можно открыть окно с их свойствами (с помощью контекстной команды Properties) (рис. 4.13 a).

В окне свойств можно просматривать установленные значения свойств серверного ЭУ, выполнять их корректировку и создавать обработчики событий (рис. 4.13 b).

Для просмотра программного кода web-формы нужно в ее контекстном меню выбрать команду «View code». Пример отображения программного кода web-формы показан на рис. 4.14.

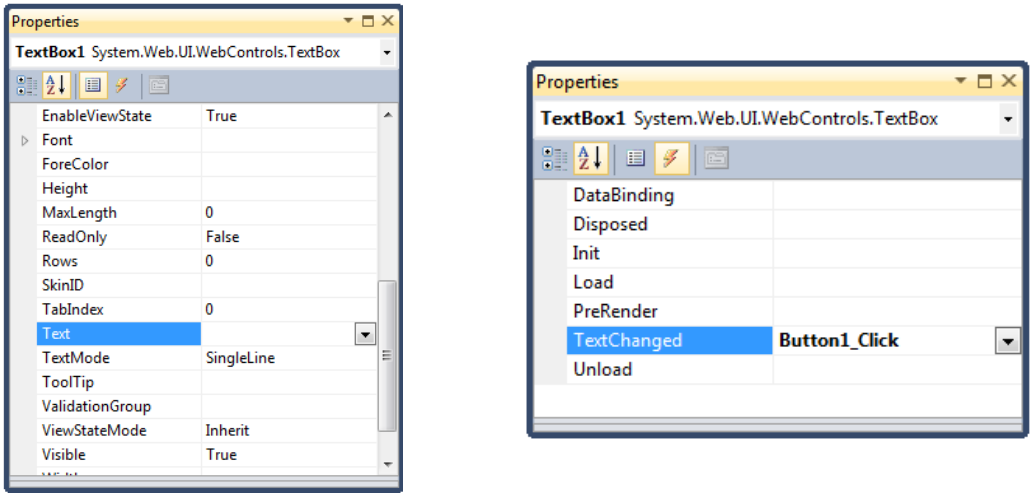


Рис. 4.13. Работа со свойствами ЭУ:
(a) окно свойств серверного элемента управления TextBox
(b) создание обработчика для события TextChanged для серверного элемента управления TextBox

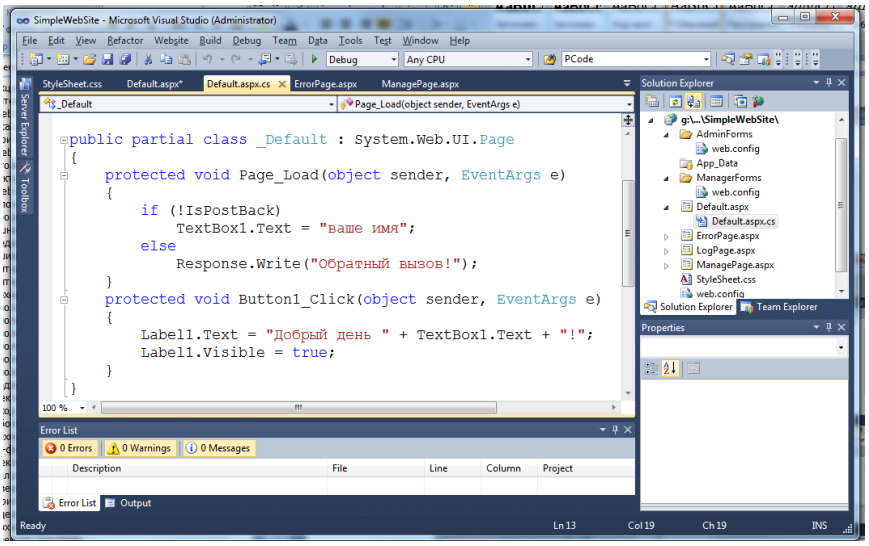


Рис. 4.14. Отображение программного кода web-формы, показанной на рис. 4.3

4.4. Разработка интерфейса пользователей

Задача построения пользовательского интерфейса является одной из самых важных в процессе разработки любого приложения. Процесс создания пользовательского интерфейса в web-приложении имеет ряд существенных отличий от традиционных Windows-приложений.

При построении web-приложения необходимо учитывать тот факт, что его интерфейс отображается в окне браузера, и, следовательно, ограничен его возможностями. Кроме того, как известно, браузер

способен отображать страницу, описанную с помощью языка HTML. В настоящее время использование HTML без языка CSS считается неэффективным, т.к. CSS способен значительно расширить базовые возможности языка HTML, связанные с позиционированием элементов внутри страницы.

Позиционирование элементов управления

Основной сложностью при разработке пользовательского интерфейса web-приложения является позиционирование его элементов на поверхности страницы.

Для позиционирования ЭУ на странице можно использовать все способы HTML проектирования, такие, как: параграфы (<p>), разрывы строк (
), таблицы и CSS стили. Visual Studio предполагает, что ЭУ добавляются на форму с помощью гибкого «потокового» (“flow”) позиционирования, так чтобы содержание формы увеличивалось и сжималось динамически без создания проблем взаимного расположения элементов.

Однако можно использовать и ***абсолютное позиционирование***, позволяющее располагать элементы в точно заданных позициях. Для этого используется команда «Format=>Set Position=>Absolute», которая добавляет в описание ЭУ атрибут style, например: style="z-index: 1; left: 10px; top: 41px; position: absolute". После этого данный ЭУ можно произвольно перемещать по всей форме, а Visual Studio будет менять его координаты.

Однако редко имеет смысл абсолютно позиционировать отдельные ЭУ. Чаще задается абсолютное позиционирование для контейнеров, например, тэгов , а затем используется потоковое позиционирование для размещения содержания внутри них. Например:

```
<div style="POSITION: absolute; left: 100px; top: 50px; width:200px">
    . . .
</div>
```

Позиционирование с помощью таблиц

Кроме этого взаимное расположение элементов в web-форме можно задавать с помощью таблицы. Для добавления таблицу в форму нужно выполнить команду «Table=>Insert Table», в результате чего появится диалоговое окно с параметрами вставляемой таблицы.

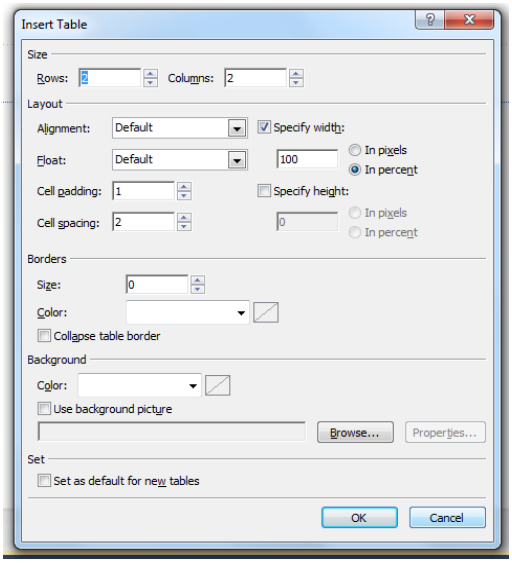


Рис. 4.15. Диалог создания таблицы на форме

По умолчанию создается таблица с невидимыми границами (с 0 толшиной). С помощью такой таблицы можно задать относительное размещение элементов на форме.

Позиционирование с помощью CSS

Более совершенным способом относительного размещения элементов web-формы является использование каскадных таблиц стилей (CSS) и мастер страниц (см. раздел “Язык каскадных таблиц стилей CSS”).

Для использования в web-форме приложения каскадной таблицы стилей ее нужно добавить с помощью команды «Website=>Add New Item=>Style Sheet». Затем с помощью окна «CSS Outline» (вызывается с помощью команды «View=>Other Window=>Document Ounline») в таблицу стилей можно добавлять различные CSS правила и задавать их свойства.

Для использования правил таблицы CSS в web-форме, ее нужно связать с таблицей с помощью тэга <link>, например:

```
<link href="StyleSheet.css" rel="stylesheet" type="text/css" />
```

В Visual Studio проще всего это сделать, перетащив css файл из окна «Solution Explorer» в область дизайнера формы (или в раздел <head> исходного текста).

Для использования конкретного правила для элемента управления, нужно его имя задать в свойстве CssStyle, например:

```
<asp:Label ID="Label1" runat="server" Text="Данная метка использует правило heading1." CssClass="heading1"></asp:Label>
```

Связывание элементов формы с правилами проще выполнять с использованием окна «Apply Styles».

Темы web-приложения

При разработке web-приложения достаточно большое внимание необходимо уделять его дизайну. Хорошо продуманный дизайн способствует повышению эффективности, увеличению простоты и удобства работы с ним пользователей. При этом дизайн всех страниц приложения должен быть понятным и одинаковым, что фактически означает необходимость стандартизации каждого элемента управления, используемого на сайте.

Одинаковое оформление содержания web-форм, позволяет воспринимать их в качестве единого web-приложения. Однако, с увеличением количества web-форм, входящих в web-приложение, увеличиваются трудоемкость поддержания их единообразного оформления, поэтому необходимы средства, способные упростить процесс создания и поддержания единообразного оформления большого количества страниц. К таким средствам можно отнести CSS – каскадные таблицы стилей, являющиеся стандартом при оформлении HTML элементов управления. Однако правила CSS не могут быть применимы к серверным ЭУ. Для задания свойств серверных ЭУ используются специальные средства ASP.Net, называемые ***темами (themes)***.

Темы, отличаются от CSS тем, что используются на стороне сервера, а не на стороне браузера, как CSS. Так как темы применимы к элементам управления, а не к элементам HTML, с их помощью можно задать оформление практически любого свойства ЭУ.

В одном web-приложении можно создать одну или несколько тем, которые могут связываться с web-формами приложения как статически, так и динамически. В ASP.Net определена специальная папка для хранения тем. При создании темы, ее необходимо поместить в отдельную папку, имя которой должно совпадать с именем темы, саму же папку необходимо поместить в каталог App_Themes, находящийся в корневом каталоге web-приложения. В том случае, если приложение содержит определение для нескольких тем, каждая из них должна быть помещена в отдельную папку.

Тема состоит из одного или нескольких файлов оформления с расширением “.skin”. Данный файл является простым текстовым файлом, в котором описываются свойства ЭУ, используемых в web-формах приложения. Описания свойств ЭУ (дескрипторы) походят на описание самих ЭУ в шаблонах web-форм (файлй aspx), однако, в отличие от самих

элементов, дескрипторы не имеют атрибута id. Они содержат лишь те свойства ЭУ, значения которых необходимо установить.

Например, наиболее типичным описанием для файла оформления является определение цвета фона или цвета переднего плана элемента управления. В этом случае, достаточно задать значения свойств BackColor или ForeColor соответственно. Таким образом, для элемента управления TextBox строка описания дескриптора может выглядеть следующим образом:

```
<asp:TextBox runat="server" forecolor="blue" backcolor="lightsteelblue"/>
```

Для добавления темы в проект web-приложения, необходимо выполнить следующие действия:

1. Выполнить команду главного меню «WebSite=>Add New Item», в открывшемся окне выбрать шаблон «Skin File», нажать OK. Visual Studio предупредит о том, что создаваемый файл оформления будет помещен в папку App_Themes и спросит, нужно ли добавить ее в проект в том случае, если она не существует.
2. Ввести дескрипторы описания серверных элементов управления приложения (пример описания оформления элементов TextBox приведен выше).

Созданная тема может быть добавлена ко всем страницам web-приложения, к файлам подкаталога или только к некоторым web-формам.

Для задания темы для всех web-форм приложения (или для web-форм подкаталога) нужно в соответствующем конфигурационном файле web.config, в котором находятся web-формы, определить используемую тему с помощью директивы pages.

```
<system.web>
    <pages theme="myTheme" />
    …
</system.web>
```

В этом случае тема будет применена ко всем формам web-приложения (или подкаталога).

В случае, если тему необходимо применить только к выбранной web-форме, то необходимо выбрать объект DOCUMENT в окне свойств страницы и ввести в свойство Theme имя темы, которую необходимо применить к данной странице. При этом Visual Studio скорректирует директиву Page данной страницы:

```
<%@ Page ….. Theme="myTheme">
```

При задании темы страницы, ASP.Net в момент выполнения страницы на сервере подменяет значения свойств элементов управления, web-формы, на соответствующие значения, которые описаны в дескрипторах темы. Таким образом, тема является более приоритетной по отношению к свойствам элемента управления.

Мастер страницы

Для задания одинаковой структуры всем web-формам приложения используются ***мастер-страницы***. Мастер-страницы являются шаблонами, которые содержат общие для всех форм элементы и специальные элементы (хотя бы один) ContentPlaceHolder (держатели места для содержания), определяющие местоположение областей, в которых будет размещаться содержание других web-форм (***страниц-содержания***). Например:

```
<asp:ContentPlaceHolder ID="HeadContent" runat="server">
</asp:ContentPlaceHolder>
```

Страница содержания в директиве Page должна включать атрибут MasterPageFile, который связывает ее с используемой мастер страницей, например:

```
MasterPageFile="~/Site.master"
```

Кроме этого, страница содержания дожна включать один или несколько элементов <asp:Content>, описывающих содержание (контент, HTML разметку и серверные ЭУ), которое включается в связанную с ней мастер страницу. Связь между ЭУ <asp:Content> страницы содержания и ЭУ <asp:ContentPlaceHolder> мастер страницы задается с помощью атрибута ContentPlaceHolderID. Например содержание следующего ЭУ:

```
<asp:Content ID="HeaderContent" runat="server"
             ContentPlaceHolderID= "HeadContent">
…
</asp:Content>
```

будет размещаться в том месте мастер-страницы, где находится ЭУ ContentPlaceHolder, имеющий ID со значением "HeadContent".

Обычно, мастер страница содержит фиксированные элементы, одинаковые для всех страниц, и заполнитель содержимого для остальной части страницы. Наиболее типичными фиксированными элементами являются верхний и нижний колонтитулы, панель навигации, панель меню и т.д. Страница содержания вставляет в такие элементы мастер страницы некоторое переменное содержимое.

В качестве примера рассмотрим мастер страницу site.master создаваемую в шаблоне ASP.Net Web Site (или Web Application), показанную на рис. 4.16.

```
<%@ Master Language="C#" AutoEventWireup="true" CodeFile="Site.master.cs"
Inherits="SiteMaster" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
            "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head runat="server"><title></title>
    <link href="~/Styles/Site.css" rel="stylesheet" type="text/css" />
    <asp:ContentPlaceHolder ID="HeadContent" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
  <form runat="server">
      <div class="page"><div class="header">
        <div class="title"><h1>My ASP.NET Application</h1></div>
        <div class="loginDisplay">
            …
        </div>
        <div class="clear hideSkiplink">
            <asp:Menu ID="NavigationMenu" runat="server" CssClass="menu"
            EnableViewState="false" IncludeStyleBlock="false"
            Orientation="Horizontal">
                <Items>
                    <asp:MenuItem NavigateUrl="~/Default.aspx" Text="Home"/>
                    <asp:MenuItem NavigateUrl="~/About.aspx" Text="About"/>
                </Items>
            </asp:Menu>
        </div>
        </div>
        <div class="main">
            <asp:ContentPlaceHolder ID="MainContent" runat="server"/>
        </div>
        <div class="clear"></div>
      </div>
    <div class="footer"></div>
  </form>
</body>
</html>
```

Рис. 4.16. Мастер страница Site.master из шаблона ASP.Net Web Site

В данной мастер-странице описаны два ЭУ типа ContentPlaceHolder, с идентификаторами HeadContent (в заголовке страницы) и MainContent с идентификатом MainContent (в теле страницы). На каждой странице также содержатся такие ЭУ, как asp:LoginView (подключение пользователя к web-приложению) и asp:Menu (меню web-приложения для перехода между web-формами). Кроме этого, данная мастер-страница испол-

зует каскадную таблицу стилей Site.css: <link href="~/Styles/Site.css" rel="stylesheet" type="text/css" />, которая задает блочную верстку формируемого HTML кода, было ранее описано в разделе «Блочная верстка HTML документов».

Мастер-страница Site.master используется, например, страницей содержания Default.aspx (рис. 4.17).

```
<%@ Page Title="Home Page" Language="C#" MasterPageFile="~/Site.master"
AutoEventWireup="true"CodeFile="Default.aspx.cs" Inherits="_Default" %>

<asp:Content ID="HeaderContent" runat="server"
            ContentPlaceHolderID="HeadContent"></asp:Content>
<asp:Content ID="BodyContent" runat="server"
            ContentPlaceHolderID="MainContent">
    <h2>Welcome to ASP.NET!</h2>
    <p>To learn more about ASP.NET visit <a href="http://www.asp.net"
    title="ASP.NET Website">www.asp.net</a>.</p>
    <p>You can also find
        <a href="http://go.microsoft.com/fwlink/?LinkID=152368&amp;clcid=0x409"
            title="MSDN ASP.NET Docs">documentation on ASP.NET at MSDN</a>
    </p>
</asp:Content>
```

Рис. 4.17. Страница-содержания Default.aspx из из шаблона ASP.Net Web Site

В данной web-форме описаны два ЭУ asp:Content, которые ссылаются на элементы asp:ContentPlaceHolder с помощью атрибутов ContentPlaceHolderID. При обращении пользователя к web-форме Default.aspx получается результат, который показан на рис. 4.18.

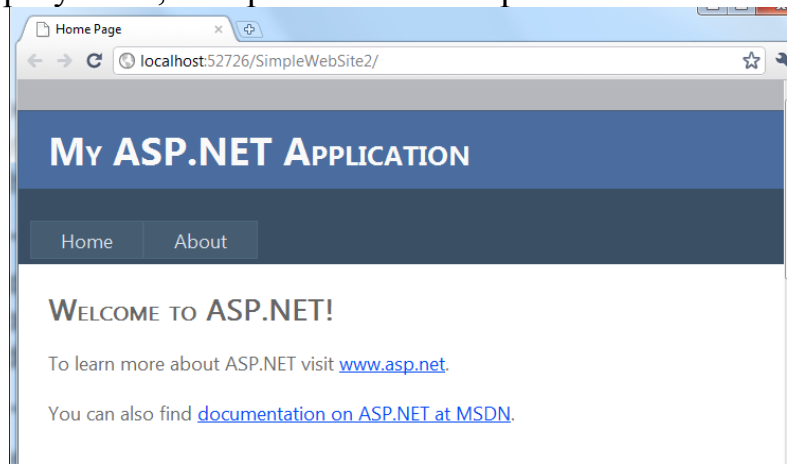


Рис. 4.18. Результат выполнения запроса пользователя к web-форме Default.aspx

Для того, чтобы создать новую web-форму в данном web-приложению (например, Page2.aspx) ее нужно ее добавить к web-приложению и при этом указать, что она будет использовать мастер-страницу (установить флажок «Select master page»), а затем выбрать используемую мастер-страницу (в данном случае – Site.master). После этого ссылку на новую страницу содержания нужно включить в меню:

```
<asp:Menu ID="NavigationMenu" runat="server" CssClass="menu"
      EnableViewState="false" IncludeStyleBlock="false" Orientation="Horizontal">
   <Items>
      <asp:MenuItem NavigateUrl="~/Default.aspx" Text="Home"/>
      <asp:MenuItem NavigateUrl="~/About.aspx" Text="About"/>
      <asp:MenuItem NavigateUrl="~/Page2.aspx" Text="Новая страница"/>
   </Items>
</asp:Menu>
```

В результате этих действий созданная страница содержания может заполняться серверными ЭУ и другим содержанием обычным образом. Полученный результат показан на рис. 4.19.

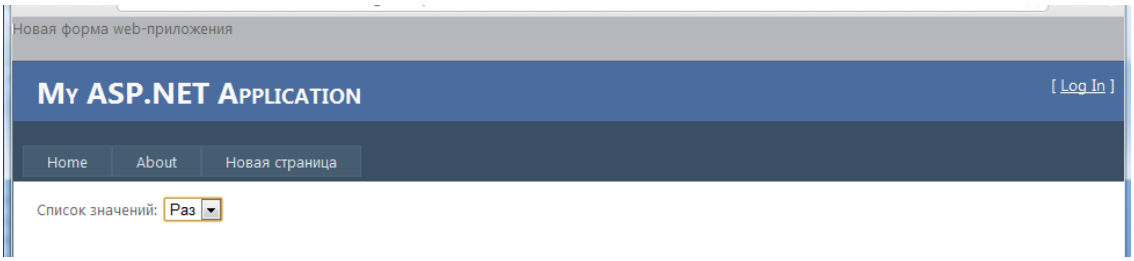


Рис. 4.19. Использование новой web-формы со стандартной мастер-страницей

4.5. Поддержка состояния сеанса работы пользователей

Одной из сложностей разработки web-приложения является то, что протокол HTTP является протоколом без сохранения состояния. Это означает, что переданные в HTTP запросах данные на web-сервере автоматически не сохраняются, а уничтожаются после их обработки на сервере. Однако среда выполнения ASP.Net приложений предоставляет много возможностей сохранения полученных от пользователей данных. Это достигается за счет использования таких средств, как:

- состояние отображения (коллекция ViewState);
- состояния сеанса работы пользователя (объект Session);
- состояние web-приложения (объект Application).

Состояние отображения web-формы ViewState выполняет автоматическое сохранение значений свойств ЭУ и доступно только внутри данной web-формы. Значения состояния отображения передаются в

HTTP ответе web-приложения (скрытое поле __VIEWSTATE) и возвращаются опять web-приложению в HTTP запросе браузера.

Для сохранения введенных пользователем в web-форму данных, чтобы ими можно было пользоваться в других web-формах одного и тоге же приложения, их необходимо сохранить в объектах с более глобальной областью видимости, таких, как: Application и Session. Переменные состояния Application доступны всем пользователям данного web приложения и могут рассматриваться как глобальные переменные, обращение к которым возможно из любых сеансов. Переменные состояния Session доступны только в рамках одного сеанса, следствием чего является то, что они оказываются доступными только одному пользователю.

В переменных состояния можно хранить данные любого типа. В силу того, что переменные состояния являются фактически глобальными переменными, для работы с ними, желательно выработать стратегию их использования в приложении.

Сохранение состояния web-формы

Прежде всего, необходимо рассмотреть возможность использования состояния вида (ViewState). Его целесообразно использовать в том случае, когда необходимо организовать хранение данных в пределах одной страницы. Все элементы управления используют состояние вида по умолчанию для сохранения значений свойств между операциями обратной отправки данных. Здесь же возможно организовать хранение своих собственных данных, состоящих из простых типов и специальных объектов.

Состояние отобажения реализовано с помощью коллекции, которая яляется словарем. Это означает, что данные хранятся в формате имя-значение. Каждый элемент, при этом индексируется с помощью уникального строкового имени. Следующий пример добавляет в коллекцию ViewState элемент с именем Name и присваивает ему значение “Иван”.

```
ViewState["Name"]="Иван";
```

При этом, если в коллекции до этого не существовало элемента с именем Name, то он добавляется, если же такой элемент был, его значение заменяется новым.

Для извлечения элемента из коллекции необходимо использовать имя элемента. Кроме того, т.к. коллекция ViewState позволяет сохранять данные состоящие не только из простых типов, но и специальные объекты (в общем случае любые объекты), во время извлечения значения элемента, необходимо преобразовать его тип к тому, который будет из-

влекаться. Следующий пример позволяет извлечь значение элемента Name и преобразовать его в строку.

```
string name;
if (ViewState["Name"]!=null)
      name=(string)ViewState["Name"];
```

Проверка на наличие элемента коллекции необходима, т.к. при обращении к несуществующему элементу коллекции возникает исключение NullReferenceException.

При работе с состоянием отображения следует учитывать два обстоятельства. Во-первых, количество информации, сохраняемой в состояние вида не должно быть большим, т.к. это приводит к увеличению объема передаваемых данных от сервера к клиенту и наоборот. В случае необходимости сохранения большого количества данных лучше воспользоваться средствами базы данных, либо использовать состояние сеанса. Во-вторых, в ViewState нельзя сохранять критично важные данные (данные, доступ пользователя к которым необходимо запретить), т.к. они легко могут быть декодированы и прочитаны, более того, опытный пользователь сможет изменить эти данные при осуществлении запроса на обратную отправку. В этом случае лучше воспользоваться состоянием сеанса.

Кроме этого ViewState не позволяет сохранять информацию, которая будет использоваться несколькими страницами. В этом случае лучше воспользоваться состоянием сеанса, объектами куки, либо строкой запроса.

Для понимания ViewState нужно рассмотреть исходный код, который web-приложение отправляет браузеру. Для этого необходимо вызвать контекстное меню страницы, отображенной в браузере и выбрать пункт «Просмотр HTML-кода». Пример текста HTML кода страницы, отправляемого в ответ на ввод имени и нажатие кнопки показан на рис. 4.4.

Объект Session

Объект Session предназначен для поддержки состояния сеанса работы пользователя. Он используется для хранения любого типа пользовательских данных, которые необходимо сохранять между последовательностью запросов одного и того же пользователя.

Пользовательские данные при этом сохраняются в формате «имя=значение». Такой способ, в частности, можно применять при создании интернет магазина, в котором покупатель перед покупкой складывает товары в виртуальную корзину, которая после завершения сеан-

са (перехода на другую страницу, либо завершения работы с браузером) должна быть удалена. Для сохранения данных о выбранных товарах можно воспользоваться объектом Session.

При подключении пользователя к приложению, создается отдельный сеанс работы и отдельная коллекция данных. По умолчанию объекты Session хранятся в оперативной памяти сервера, но есть и другие возможности хранения состояния в отдельном сервисе или даже базе данных. Как и другие параметры web- приложения, способ хранения состояния задается в конфигурационном файле. Более подробно данная тема описана в [5, 9].

Следует отметить, что возможности использования объектов Session имеют и свою цену. Даже при маленьком объеме хранимых данных сеанса, их использование может оказать влияние на производительность работы приложения в том случае, если к web-приложению одновременно будут обращаться сотни или даже тысячи пользователей.

Работа с состоянием сеанса практически аналогично работе с состоянием вида, за исключением того, что вместо ключевого слова ViewState используется Session. Например, для сохранения объекта user в памяти сеанса, необходимо выполнить следующий код:

```
Session["user"]=user;
```

Для восстановления сохраненного объекта user необходимо воспользоваться следующим кодом:

```
user=(User)Session["user"];
```

Состояние сеанса уничтожается в следующих случаях:

1. если пользователь закрывает браузер;
2. по истечении 20 минут с момента последней активности пользователя;
3. при явном завершении сеанса из программного кода с помощью вызова метода Session.Abandon().

Объект Application

Объект Application во многом аналогичен объекту Session, но хранит состояние всего приложения. Данные, которые хранятся в объекте Application являются глобальными, так как они доступны во всех сеансах работы пользователей приложения. Состояние приложения похоже на состояние сеанса, т.к. хранит информацию на сервере, позволяет сохранять объекты такого же типа и использует формат имя-значение для хранения данных.

Одной из наиболее часто используемых возможностей объектов Request и Response является работа с куки (cookie). Куки являются одним из возможных способов сохранения информации во время работы приложения для ее дальнейшего использования. Например, для проверки того, поддерживает ли браузер клиента куки необходимо выполнить следующий код:

```
if (!IsPostBack) {
   if (Request.Browser.Cookies)
      Response.Write("Ваш Браузер поддерживает cookies");
   else
      Response.Write("Ваш Браузер не поддерживает cookies");
}
```

В следующем примере, при наличии в браузере поддержки создается куки UserName, которому присваивается значение «Иванов Петр»:

```
protected void Page_Load(object sender, EventArgs e) {
   if (!IsPostBack) {
      if (Request.Browser.Cookies)
         if (Request.Cookies["UserName"] != null){
            Session["User"] = Request.Cookies["UserName"].Value;
            Response.Write(Request.Cookies["UserName"].Value);
         }
         else{
            HttpCookie uname = new HttpCookie("UserName");
            uname.Value = "Иванов Петр";
            Response.Cookies.Add(uname);
         }
      else
         Response.Write("Ваш Браузер не поддерживает cookies");
   }
}
```

Используемый в предыдущем примере куки, будет сохраняться до тех пор, пока пользователь не закроет окно браузера, при этом он будет отправляться с каждым запросом. При необходимости сохранения куки в течение определенного времени, необходимо установить дату истечения срока действия куки. В следующем примере куки будет храниться в течение одного года:

```
uname.Expires=DateTime.Now.AddYears(1);
```

Если необходимо удалить куки, то для него нужно задать «просроченную» дату истечения срока действия. Это можно сделать следующим образом: uname.Expires=DateTime.Now.AddDays(-1);

4.6. Навигация по web-приложению

Любое web-приложение представляет собой достаточно сложную совокупность взаимосвязанных web-форм. Хорошо спроектированное web-приложение обладает хорошей системой навигации, позволяющей легко переходить от одной web-формы к другой, а также решать другие задачи. ASP.Net обладает достаточно большими возможностями, позволяющими реализовывать сложные системы навигации с помощью набора специальных элементов управления. Рассмотрим наиболее важные и распространенные из них.

Программный переход между web-формами

Одной из часто возникающих проблем при разработке web-приложений является передача информации от одной страницы к другой и от одного приложения к другому приложению. Существует несколько способов решения этой проблемы, одним из которых является использование строки запроса, при котором данные передаются в URL адресе.

Преимущество такого подхода заключается в том, что строка запроса проста по своей структуре и не вызывает нагрузки на сервер, с помощью такого механизма можно легко переносить информацию с одной страницы на другую. Недостаток заключается в том, что с помощью строки запроса возможно передавать только информацию, представленную в виде простых строк, содержащих символы, которые допускается использовать в URL адресе.

Для передачи информации в строке запроса, ее необходимо поместить в URL адрес страницы, к которой должен произойти переход. Это возможно сделать используя элемент управления HyperLink, либо воспользоваться оператором Response.Redirect().

Например, для того, чтобы перейти на страницу login.aspx и передать в строке запроса переменную username необходимо выполнить следующий код:

```
string FirstName="Иван";
string LastName="Иванов";
Response.Redirect("login.aspx?username="+FirstName+" " + LastName);
```

Для передачи нескольких параметров в строке запроса, параметры необходимо разделять знаком амперсанд – '&'. С учетом этого, предыдущий пример можно переделать так, чтобы имя и фамилия пользователя передавались отдельно. Для этого изменим строку Response.Redirect следующим образом:

```
Response.Redirect("login.aspx?firstname=" + FirstName +
                  "&lastname=" + LastName);
```

Для извлечения строки запроса, необходимо использовать метод QueryString объекта Request. Для извлечения значений параметров, передаваемый в предыдущий примерах, необходимо использовать следующий код:

```
string FN=Request.QueryString["firstname"];
string LN=Request.QueryString["lastname"];
```

Карта сайта

В том случае, когда web-приложение содержит большое количество web-форм то целесообразно использовать в карту сайта. Карты сайта является удобным способом описания структуры web-приложения, а также позволяет ее показывать пользователям помощью нескольких элементов управления. Эти элементы управления расположены в разделе Navigation панели инструментов (Toolbox). Основными элементами управления, предоставляющими возможность показывать карту сайта, являются: SiteMapPath, Menu, TreeView. Все эти элементы предназначены для решения одной и той же задачи – предоставления возможности пользователю web-приложения осуществлять навигацию по web-формам. Различие между ними заключается в способах отображения ссылок на соответствующие формы.

Для использования любого их перечисленных компонентов, необходимо определить структуру приложения, которая хранится отдельно. В качестве хранилища структуры обычно выступает XML файл. В Visual Studio существует уже определенная структура XML файла, предназначенная для хранения структуры приложения, это файлы типа .sitemap. Содержимое данного файла достаточно простое, что позволяет легко вносить данные о структуре приложения. Разделение элементов, отвечающих за отображение данных и элементов, содержащих эти данные позволяет значительно упростить редактирование приложения – для изменения дизайна достаточно изменить настройки визуальных компонентов, для внесения дополнений – добавить данные в XML файл.

Рассмотрим пример добавления навигации по страницам web-приложения с использованием карты сайта. Прежде всего, необходимо добавим XML файл, и описать в нем данные о структуре приложения.

Для добавления XML файла можно использовать стандартную команду «Website =>Add New Item». В открывшемся окне нужно выбрать тип файла «Site Map». В результате этого будет создан файл

Web.sitemap, содержащий заготовки для ввода структуры web-приложения:

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
    <siteMapNode url="" title="" description="">
        <siteMapNode url="" title="" description="" />
        <siteMapNode url="" title="" description="" />
    </siteMapNode>
</siteMap>
```

Как видно из исходного кода, карта сайта начинается с корневого узла <siteMap>. Элементы структуры описываются в тэгах <siteMapNode>. С помощью этих тэгов можно указывать иерархию элементов web-приложения. Для указания вложенных элементов их просто необходимо расположить внутри соответствующего тэга <siteMapNode>. Свойства каждого тэга необходимы для задания соответствующих значений. Из примера выше видно, что каждому элементу соответствует три свойства: url, title, description. Их назначение очевидно: url используется для указания интернет адреса страницы, которой соответствует этот элемент; title задает наименование элемента, отображаемое элементом управления; description – описание элемента, которое отображается в виде всплывающей подсказки при наведении указателя мыши на соответствующий элемент.

В качестве примера простейшей карты сайта создадим иерархию страниц, состоящую из шести элементов. Самым верхним элементом иерархии будет домашняя страница, ссылающаяся на файл Default.aspx. Все остальные элементы будут вложены в него. Кроме того, в элемент «Страница 4» должны быть вложены еще два элемента. Пример карты сайта, описывающей данную структуру, приведен ниже.

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
  <siteMapNode url="Default.aspx" title="Домашняя" description="">
    <siteMapNode url="Page2.aspx" title="Страница 2"
      description="Перейти к странице 2" />
    <siteMapNode url=" Page3.aspx" title="Страница 3" description=""/>
    <siteMapNode url=" Page4.aspx" title="Страница 4">
      <siteMapNode url="Page5.aspx" title="Страница 5"/>
      <siteMapNode url="Page6.aspx" title="Страница 6"/>
    </siteMapNode>
  </siteMapNode>
</siteMap>
```

После того, как карта сайта определена, становится возможным использование элементов управления, связанных с ней для отображения

данных о структуре приложения. Для этого возможно использование таких элементов управления как TreeView, Menu, SiteMapPath. Элементы TreeView и Menu имеют одинаковую функциональность и предназначены для вывода элементов структуры приложения в окне браузера. Различие заключается в форме представления данных. Элемент SiteMapPath предназначен для отображения текущего положения пользователя в иерархии web-приложения и позволяет ему переходить вверх по иерархии на более высокий уровень. Рассмотрим примеры использования все трех элементов управления.

Элементы управления TreeView и Menu

Для использования таких ЭУ необходимо создать мастер страницу, содержащую несколько областей, и добавить Menu или TreeView в одну из областей, например:

```
<div class="leftCol">
   <asp:Menu ID="Menu1" runat="server"> </asp:Menu>
   <asp:TreeView ID="TreeView1" runat="server"> </asp:TreeView>
</div>
```

После этого необходимо вставленные элементы управления связать с источником данных, содержащим карту сайта. Для этого необходимо в свойстве «Choose Data Source» выбрать пункт <New data source…> и в открывшемся окне выбрать Site Map, как показано на рис. 4.20.

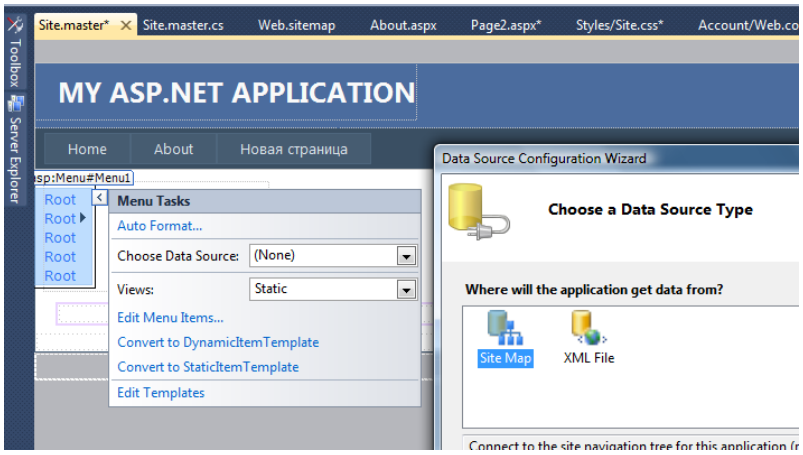
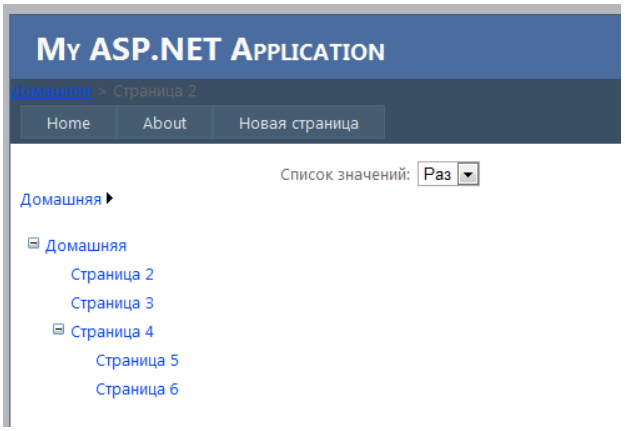


Рис. 4.20. Определение источника данных ЭУ Menu.

После этого код описания ЭУ будет иметь следующий вид:

```
<div class="leftCol">
   <asp:Menu ID="Menu1" runat="server" DataSourceID="SiteMapDataSource1">
   </asp:Menu>
   <asp:TreeView ID="TreeView1" runat="server"
            DataSourceID="SiteMapDataSource1"> </asp:TreeView>
   <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" />
</div>
```

Результат добавления ЭУ для навигации по карте сайта будет показываться следующим образом (после перехода на Страницу 2):



`Рис. 4.21. Добавление элементов TreeView и Menu.

Элементы управления TreeView и Menu обладают дополнительными свойствами, с помощью которых возможно изменить их внешний вид, в соответствии с дизайном web-приложения, например:

```
<asp:Menu ID="NavigationMenu" runat="server" CssClass="menu"
EnableViewState="false" IncludeStyleBlock="false" Orientation="Horizontal">
      <Items>
            <asp:MenuItem NavigateUrl="~/Default.aspx" Text="Home"/>
            <asp:MenuItem NavigateUrl="~/About.aspx" Text="About"/>
      </Items>
</asp:Menu>
```

Элемент управления SiteMapPath

В отличие от TreeView и Menu, SiteMapPath отображает только текущее положение пользователя в иерархии страниц web-приложения с возможностью перехода к верхнему уровню иерархии. Добавим в пример элемент SiteMapPath. Для этого необходимо добавить еще одну строку между заголовком и содержимым мастер страницы и просто поместить в нее элемент SiteMapPath. Результат показан на рис. 4.21 (переход на «Страницу 2»).

4.7. Работа web приложений с базами данных

Web-приложения в основном являются приложениями, функционирование которых основывается на активной работе с данными.

Взаимодействие web-приложений с базами данных (БД) выполняется, также как и Windows приложений с помощью технологии ADO.Net [4] и других технологий, как например Linq-to-SQL и Entity Framework. Работа с использованием ADO.Net может выполняться, как в режиме с поддержкой соединения (работа с провайдерами данных – классы Connection, Command и DataReader), так и отсоединенном режиме (классы DataAdapter, DataSet, DataTable, DataView и т.п.). Применяя такие классы можно выполнять запросы к БД и получать выборки данных (например, объекты классов DataReader или DataTable), которые затем будут использоваться в программном коде требуемым образом (для выво).

Основными средствами взаимодействия web-приложений с данными являются следующие:

1. источники данных и компоненты связывания с источниками данных;
2. связывание серверных ЭУ с источниками данными;
3. выполняемые выражения вида <%# [выражение] %> и <%$ [выражение] %>;
4. шаблоны описания серверных ЭУ с использованием выполняемых выражений <%# Eval(…) %>.

Источники данных

В web-приложении в качестве источников данных могут использоваться разные .NET классы, и не только те, которые организуют взаимодействие с БД. В ASP.NET, любой объект, который поддерживает интерфейс IEnumerable может быть связан с серверными ЭУ. Интерфейс IEnumerable определяет минимальный API, который необходим для перебора содержания источника данных. Однако, многие связываемые объекты в действительности реализуют более совершенные версии IEnumerable, такие итерфейсы, как ICollection и IList [4]. В частности серверные web ЭУ могут быть связаны с объектами следующих классов:

- объекты коллекции (включая словари (Dictionary), хэш-таблицы (Hashtable) и массивы (Array));
- объекты класса DataReader провайдеров ADO.NET;
- объекты контейнерных классов ADO.NET, таких, как DataSet, DataTable и DataView;

- любые объекты, которые получается в результате обработки запроса LINQ, поддерживающие интерфейс IQueryable.

Кроме объектов перечисленных классов, источниками данных могут быть специальные ***компоненты связывания с источниками данных*** (data source components).

Компонент связывания с источником данных (КСИД) это серверный элемент управления, который предназначен для взаимодействия со связанными с данными элементами управления и для скрытия сложности программного связывания ЭУ с источником данных. Такие компоненты не только предоставляют данные для серверных ЭУ, но они также поддерживают выполнения этими ЭУ других общих операций с данными, таких, как добавление новых даных, удаление, сортировка и обновление. Каждый КСИД является оберткой для некоторого провайдера данных – реляционных БД, XML документов, специальных объектных моделей или созданных разработчиком классов. Поддержка созданных разработчиком классов означает, что можно напрямую связывать ЭУ с существующими классами, например, с классами уровня бизнес-логики или классами доступа к данным.

Все компоненты связывания с источниками данных поддерживают выборку данных с помощью метода Select. Данный метод возвращает объект, реализующий интерфейс IEnumerable.

Типы основных компонентов-источников данных описаны в табл. 4.8.

Таблица 4.8

Основные компоненты связывания с источниками данных

Класс	**Описание**
AccessDataSource	Предоставляет соединение с базой данных Microsoft Access. Наследуется от компонента SqlDataSource, но работает с MDB файлами и использует провайдер данных Jet 4.0 OLE DB для соединения с базой данных.
EntityDataSource	Позволяет использовать результаты обработки запросов по технологии Entity Framework.
LinqDataSource	Позволяет использовать результаты работы любого провайдера, который поддерживает LINQ запросы, включая и технологию LINQ-to-SQL. Данный компонент позволяет задать контекст данных, название таблицы, параметры отображения полей таблицы на свойства класса и условия выборки (where clause).

Окончание табл. 4.8

Класс	Описание
ObjectDataSource	Позволяет связываться с созданными разработчиками объектами .Net классов бизнес-логики, которые возвращают данные. Предполагается, что эти классы соответствуют специальному шаблону проектирования, и включают, например, конструктор без параметров и методы, которые работают определенным образом.
SqlDataSource	Предоставляет соединение с провайдером данных ADO.NET, который возвращает SQL данные, включая источники данных доступные с помощью технологий OLE DB и ODBC. Имя провайдера и строка соединения задается с помощью свойств компонента.
SiteMapDataSource	Позволяет связываться с любым провайдером, который предоставляет информацию о карте сайта. По умолчанию провайдер предоставляет данные по карте сайта, которые содержатся в XML файле хранящемся в корневой папке web-приложения.
XmlDataSource	Позволяет связываться с XML файлами и строками files and strings with or without schema information.

Для конфигурирования компонента SqlDataSource можно воспользоваться мастером, запускаемым при нажатии на ссылку «Configure Data Source».

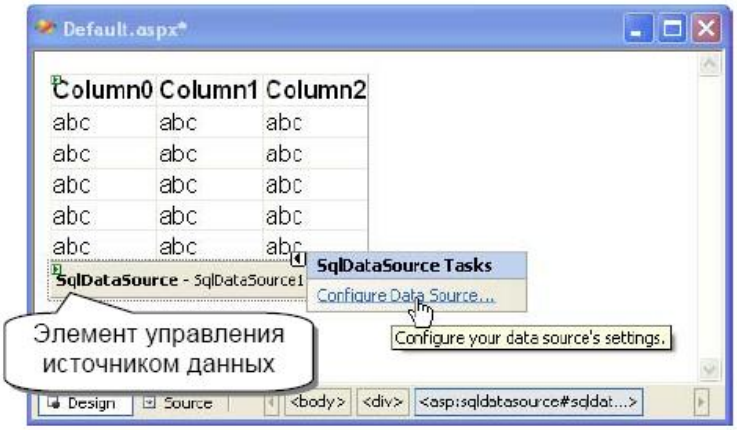


Рис. 4.22. Использование компонента управления источником данных.

На первом шаге конфигурирования элемента SqlDataSource необходимо определить соединение с базой данных. Если существует уже созданное соединение с источником данных, то его просто можно выбрать из списка, в противном случае, можно создать и настроить новое подключение к БД, нажав на кнопку «New Connection».

На втором шаге, мастер предлагает сохранить сгенерированную строку подключения к БД в файле web.config. При необходимости можно отказаться, а также изменить наименование переменной, содержащей строку подключения.

Третий шаг мастера позволяет определить запросы к БД, используемые SqlDataSource для выполнения операций над данными. В частности, предлагается выбрать таблицу и сгенерировать запрос на выборку из нее данных. Здесь также можно определить и свои собственные запросы на выборку, добавление, удаление и обновление данных или указать используемую хранимую процедуру.

Мастер может также сгенерировать стандартные запросы, используемые для добавления, удаления и обновления значений в связанном источнике данных, если выбраны поля, являющиеся первичными ключами таблицы. Для построения этих запросов можно воспользоваться кнопкой Advanced. При этом будет открыто окно, в котором можно установить два свойства. Ссылка «Generate INSERT, UPDATE, and DELETE statements» задает режим создания соответствующих запросов.

Четвертый шаг мастера позволяет протестировать созданный запрос и, после нажатия на кнопку Finish завершить настройку элемента управления источником данных.

В результате выполнения этих операций в описании шаблона формы будет добавлено описание компонента связывания с источником данных, например:

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
    ConnectionString="<%$ ConnectionStrings:Northwind %>"
    SelectCommand="SELECT [EmployeeID], [LastName], [FirstName], [Address], [City], [PostalCode] FROM [Employees]">
</asp:SqlDataSource>
```

После описания такого компонента, с ним можно связывать элементы управления. Для этого его свойству DataSourceID необходимо задать имя объекта SqlDataSource.

Следует иметь в виду, что каждый компонент связывания с источником данных может использоваться для соединения только с одной таблицей базы данных. Если в web-приложении требуется независимо извлекать или обновлять информацию из нескольких таблиц, необходимо использовать несколько таких компонент.

Выполняемые выражений в шаблоны web-формы

В шаблоны web-форм .aspx помимо статического текста и серверных ЭУ могут включаться выполняемые выражения. Такие выражения

после выполнения включают вместо себя в web-формы некоторые значения. Имеется два типа выполняемых в шаблоне web-формы выражений:

- ***$-выражения*** (имеют формат <%$ [выражение] %>);
- ***#-выражения*** (имеют формат <%# [выражение] %>).

Выполняемые $-выражения (<%$ … %>) могут включаться только в тэги серверных ЭУ для задания их свойств. Такие выражения выполняются в момент обращения к странице. Имеются два основных вида таких выражений:

- для получения значения из раздела <appSettings> файла web.config <%$ AppSettings:[имя] %>: . Например, выражение <%$ AppSettings:appName %> используется для получения значения переменной appName из файла config.web:

```
<appSettings>
    <add key="appName" value="xxxxx" />
</appSettings>
```

- для получения строки соединения из раздела <appSettings> файла web.config <%$ ConnectionStrings:[имя] %>: . Например, выражение <%$ ConnectionStrings:Northwind %> используется для получения текстовой строки соединения с именем Northwind из файла config.web:

```
<connectionStrings>
  <add name="Northwind" connectionString="Data Source=
    localhost\sqlexpress; Initial Catalog=Northwind; Integrated Security = True"
    providerName="System.Data.SqlClient" />
    . . .
</connectionStrings>
```

Visual Studio содержит встроенный построитель $-выражений, позволяющий извлекать пользовательские настройки приложения и информацию о строке подключения, расположенные в файле web.config.

Выполняемые выражения <%# … %>. Выражение, записаное в таком формате, будет выполняться только после вызова метода DataBind() для страницы Page, которая содержит ЭУ, или для самого ЭУ. Данное выражение может использовать переменные, константы и вызовы открытых (public) и защищенных (protected) методов программного кода. Оно может включаться в любое место шаблона web-формы .aspx и обычно используется для задания свойств элементов управления (не только текстовых, но и любых, например, Color, NavigateUrl и т.п.). Например:

```
<asp:Label id=“label1” Text=<%# “Результат равен ” + (1 + 2) +
```

“, текущее время ” + DateTime.Now.ToLongTimeString() %> runat="server" />

В результате выполнения метода DataBind():

```
public void Page_Load(object s, EventArgs e) {
      if (! Page.IsPostBack)
            Page.DataBind();
}
```

на странице будет показано текстовое сообщение, вида:

Результат равен 3, текущее время 14:05:27

Имеется **специальный вид #-выражений** <%# Eval() %>**,** которые позволяют включать в шаблоны составных ЭУ данные из связанной с ним выборки. Данное выражение имеет следующий полный синтаксис:

<%# DataBinder.Eval(Container.DataItem, expression) %>

В данном выражении на выполнение вызывается статический метод Eval() класса DataBinder. Данный метод производит выполнение заданного выражения expression (обычно название поля обрабатываемой записи) относительно текущего выполняемого объекта Container.DataItem (это текущий объект (обрабатываемая запись), который используется в шаблоне составного ЭУ).

Параметр Container.DataItem ссылается на объект, для которого оценивается второй параметр expression. Свойство DataItem представляет собой объект в контейнере текущего контекста. Обычно контейнер является текущим экземпляром элемента данных, например, объект DataGridItem, который готов к использованию для формирования HTML кода. Второй параметр обычно является строкой с именем поля, значение которого нужно получить у объекта элемента данных. Это может быть выражением, которое включает индексы и имена свойств.

Показанный ранее код обычно повторяется, всегда в одной и той же форме. В разных страницах меняются только второй параметр expression и форматирующая строка (третий параметр, который не показан). В связи с этим имеется следующая компактная форма записи метода DataBinder.Eval(): <%# Eval(expression) %>.

Основные свойства ЭУ связываных с данными

Одним из наиболее эффективных способов использования в web-приложении данных из разных источников является возможность их связывания с элементами управления. В ASP.Net имеется большая группа серверных элементов управления, которые можно связывать с источниками данных. Такие ЭУ называются ***связанными с данными элементами управления.***

Связывание с данными (data binding) является процессом получения данных из некоторого источника и присвоения их свойствам элементов управления. При формировании такими ЭУ кода разметки (при рендеринге), в нее будут включаться связанные данные.

Таблица 4.9

Основные свойства связывания элементов управления с данными

Свойство	Описание
DataSource	Ссылка на объект, который содержит коллекцию элементов. Этот объект должен реализовывать интерфейс, который поддерживает связывание данных, обычно это ICollection.
DataSourceID	Используя это свойство ЭУ можно связать с компонентом связывания с источником данных. В этом случае компонент будет создавать требуемый объект данных автоматически. Для одного ЭУ нельзя одновременно использовать оба свойства DataSource и DataSourceID.
DataMember	Если источник данных содержит несколько коллекций данных (например, объект DataSet может содержать несколько DataTable), свойству DataMember задается имя коллекции данных из которой будут извлекаться данные.
DataTextField	Некоторые ЭУ могут показать только одно значение элемента данных (например, списочные ЭУ). Свойство DataTextField задает поле (при работе с записями) или свойство (при работе с объектами) элемента данных, которое содержит значение для отображения на странице.
DataValueField	Это свойство аналогично свойству DataTextField, но заданное значение элемента данных, которое не отображается на странице, а сохраняется в атрибуте value используемого HTML тэга. Это позволяет получить данное значением в коде программы. Основное назначение данного свойства хранить уникальные ID или первичные ключи, чтобы их можно было использовать для получения дополнительных данных, когда пользователь выберет данных элемент.

Связанные с данными ЭУ не являются специальной группой ЭУ. Это просто серверные web-ЭУ, которые имеют несколько специальных свойств, с помощью которых эти ЭУ можно заполнить набором данных. Разработчики в любое время могут связать ЭУ с коллекцией данных или с компонентом связывания с источником данных путем задания значе-

ний этих свойств. Описание основных свойств связывания ЭУ с данными приведено в табл. 4.9.

Однако для изменения внешнего вида (представления) ЭУ не достаточно простого задать значения этим свойствам. Реальный процесс связывания ЭУ с данными начинается при вызове метода DataBind() у ЭУ или у объекта Page. Для ЭУ выполнение операции связывания с данными означает обновление его внутреннего состояния, чтобы отражать коллекцию значений, присвоенную его свойствам. И, наконец, при формировании ЭУ соответствующей ему разметки (в результате вызова на выполнение метода RenderControl()), в эту разметку будут включаться данные, связанные с ЭУ.

Например, можно связать с данными списочный серверный ЭУ, включенный в шаблон web-формы:

```
<asp:DropDownList id="ddList1" runat="server">
```

Для этого нужно следующим образом задать источник данных в виде объекта класса SqlDataReader:

```
protected void Page_Load(object sender, EventArgs e){
   if (!IsPostBack) {
      ddList1.DataSource = GetData();
      ddList1.DataTextField = "country";
      ddList1.DataValueField = "country";
      ddList1.DataBind();
   }
}
// Метод выборки данных из базы данных с помощью объекта DataReader
SqlDataReader GetCityData() {
   string connectionString = "Data Souce=localhost/sqlexpress;
            Initial Catalog=Northwind; Integrated Security = True";
   SqlConnection connection = new SqlConnection (connectionString);
   connection.Open();
   string commandText = "select distinct country from Customers";
   SqlCommand command = new SqlCommand(commandText, connection);
   SqlDataReader dr = command.ExecuteReader();
   return dr;
}
```

В результате выполнения данного кода будет получен выпадающий список со всеми странами, где есть клиенты (Customers) компании из базы данных Northwind.

Недостатком данного кода является то, что строка соединения содержится в программном коде. Если потребуется изменить расположение БД или используемый провайдер, то такой код надо будет корректировать. Лучше строку соединения хранить в файле конфигурации (ко-

торую может корректировать администратор web-приложения), а в коде ее считывать из файла конфигурации. Это можно сделать с помощью объекта WebConfigurationManager, который определен в пространстве имен System.Web.Configuration следующим образом:

```
using System.Web.Configuration;
…
string connectionString = WebConfigurationManager.ConnectionStrings ["North-
wind"] .ConnectionString;
```

То же самое связывание списочного ЭУ с источником данных можно выполнить без выполнения программирования используя компоненты связывания с источниками данных. Для этого нужно создать компонент связывания с источником данных (разместить его на форме) и выполнить настройку:

```
<asp:SqlDataSource ID="SqlDS1" runat="server"
    ConnectionString="<%$ ConnectionStrings:Northwind %>"
    SelectCommand="SELECT select distinct country from Customers ">
</asp:SqlDataSource>
```

А затем у ЭУ DropDownList в качестве источника указать созданный компонент:

```
<asp:DropDownList id="ddList1" DataSourceID="SqlDS1" runat="server">
```

В результате получим аналогичный результат.

Виды связанных с данными серверных ЭУ

Кроме списочных ЭУ, в ASP.NET имеется два основных вида связанных с данными ЭУ:

- ***повторяющие ЭУ*** (iterative controls) – позволяют показать поля элементов источника данных (записей) с помощью специально заданных шаблонов (Repeater, DataList, DataGrid); такие ЭУ появились в первых версиях технологии ASP.Net;
- ***представляющие ЭУ*** (view controls) – более совершенные ЭУ, которые предоставляют большие возможности по различным способам отображения данных из разных источников; они позволяют также выполнять с данными стандартные операции, такие, как удаление и корректировка (DetailsView, FormView, GridView, ListView).

Все связанные с данными ЭУ имеют общую структуру:

```
<asp:[связанный-с-данными-ЭУ] DataSourceID=[компонент связывания с
данными] …>
    [свойства элемента управления]
    [шаблоны описания отображения записей из источника данных]
```

</asp:[связанный-с-данными-ЭУ]>

Как видно из данного описания, они имеют ссылку (свойство DataSource или DataSourceID) на источник данных и в них описываются ***шаблоны отображения записи источника данных*** в формируемой странице. Такое отображение описывается с помощью специальных шаблонов.

Таким образом, все web-приложение, разработанное по технологии ASP.Net Web Forms состоит из множества шаблонов: шаблонов форм и шаблонов отображения различных серверных ЭУ.

Основными ЭУ, связанными с данными, являются ***DetailsView, FormView, FormView*** и ***ListView***.

ЭУ DetailsView позволяет автоматически формировать отображение одной записи из связанного с ним источника данных (ИД), позволяет ключить в это отображение кнопки перехода между записями. DetailsView может быть связан с любым компонент связывания с ИД, и может выполнять набор операций по работе с данными. Он позволяет добавлять новые данные, изменять и удалять данные из используемого ИД. В большинстве случаев для выполнения любой из этих операций не требуется выполнять программирование. Разработчик может настраивать пользовательский интерфейс ЭУ DetailsView, путем выбора наиболее подходящей комбинации полей данных и стилей с помощью Visual Studio. Однако разработчик не имеет больших возможностей управлять кодом разметки.

ЭУ FormView может рассматриваться в качестве варианта ЭУ DetailsView, который поддерживает шаблоны. Данный ЭУ также показывает одновременно одну запись, выбранную из связанного с ЭУ ИД, и может показывать кнопки для выполнения переходов между записями. В отличие от ЭУ DetailsView, ЭУ FormView не выполняет никакого автоматического формирования разметки и требует от разработчика описания шаблонов формирования разметки для каждого показываемого элемента. ЭУ FormView может поддерживать любые базовые операции, которые предоставляет ИД. Отметим, что FormView требует, чтобы разработчика описал с помощью шаблонов все элементы разметки, а не только то, что он хочет изменить. Данный ЭУ не имеет встроенных возможностей рендеринга, а может только выводить заданный пользователем шаблон.

ЭУ GridView является наследником ЭУ DataGrid и позволяет показывать данные в виде таблицы. Он предоставляет тот же самый набор базовых возможностей, а также большой список расширений и улучшений. ЭУ DataGrid, который все еще поддерживается в ASP.NET, является

очень мощным и универсальным ЭУ. Однако, у него есть один большой недостаток: он требует, чтобы разработчик писал большой объем кода, даже для выполнения простых операций, таких как разделение на страницы, сортировка, корректировка или удаления данных. ЭУ GridView был разработан для преодоления этого недостатка и для реализации двух-стороннего связывания данных, без необходимости написания большого количества программного кода. Данный ЭУ может выполнять напрямую обновление ИД.

ЭУ ListView полностью основывается на шаблонах и позволяет управлять всеми деталями пользовательского интерфейса, с помощью шаблонов и свойств. Работа ЭУ ListView очень сходна по поведению с такими связанными с данными ЭУ, как FormView или DataList. Однако, в отличие от них, ЭУ ListView никогда автоматически не создает разметку пользовательского интерфейса. Каждый тэг разметки, который данный ЭУ формирует полностью находится под управлением разработчика. ЭУ ListView может быть связан с любым компонентом связывания с ИД и может выполнять имеющийся у него набор операций с данными. Он позволяет автоматически выполнять разбивку данных на страницы, корректировку данных, добавление новых данных и удаление.

В качестве примера связанного с данными элемента управления рассмотрим ЭУ GridView.

Серверный элемент управления GridView

ЭУ GridView позволяет показывать содержания ИД в табличном формате. Каждое поле данных ИД показывается в виде таблицы, а каждая запись в отдельной строке. GridView поддерживает большой набор свойств, которые описывают поведение, параметры визуализации, стиль, состояние и шаблоны разметки. В табл. 4.10 описаны основные свойства, которые влияют на поведение ЭУ GridView.

Таблица 4.10

Основные свойства, задающие поведение ЭУ GridView

Свойство	Описание
AllowPaging	Логическое значение, которое включает или выключает поддержку разбивки данных по страницам (paging).
AllowSorting	Логическое значение, которое включает или выключает выполнение сортировки строк таблицы по кзначениям полей.
AutoGenerateCo-lumns	Логическое значение, которое указывает, выполнять или нет автоматическое создание колонок таблицы для каждого поля в ИД. По умолчанию true.

Окончание табл. 4.10

Свойство	Описание
AutoGenerateDeleteButton	Логическое значение, которое включает или выключает вставку в каждую строку колонки с кнопкой, позволяющей удалять текущую строку из ИД.
AutoGenerateEditButton	Логическое значение, которое включает или выключает вставку в каждую строку колонки с кнопкой, позволяющей выполнять корректировку значений текущей строки в ИД.
AutoGenerateSelectButton	Логическое значение, которое включает или выключает вставку в каждую строку колонки с кнопкой Select, позволяющей пользователям выполнять выбор строк (возникает событие SelectedIndexChanged).
SortDirection	Задание направления сортировки.
SortExpression	Задает текущее выражение сортировки, которое используется упорядочения записей.

Конфигурирование колонок

Основным содержанием ЭУ GridView является набор специальных элементов, которые описывают колонки показываемой таблицы. Каждая колонка таблицы может быть описана с помощью с помощью специальных серверных подэлементов управления (т.е. элементов, которые включаются в другие серверные элементы управления).

Свойство Columns является коллекцией (набором) объектов DataControlField. Колонки таблицы задаются в виде связанных с данными полей. Разработчик может определить колонки либо декларативно (в шаблоне) или программно (в программном коде). Колонки данных показываются в таблице в том порядке, в котором они появляются в коллекции. Для декларативного определения колонок в файле .aspx, нужно использовать тэг <Columns>, как показано ниже:

```
<columns>
    <asp:boundfield datafield="customerid" headertext="ID" />
    <asp:boundfield datafield="companyname" headertext="Company Name" />
</columns>
```

В табл. 4.11 описываются классы колонок, которые могут использоваться в коллекции columns.

Таблица 4.11

Основные типы колонок в ЭУ GridView

Тип	Описание
BoundField	Тип колонки, которая показывает значение заданного поля в виде простого текста. Данный тип используется по умолчанию.
ButtonField	Данная колонка показывает значение заданного поля в виде командной кнопки. Можно выбрать стиль отображения данной кнопки: обычеая кнопка (push), гиперссылка (link) или изображение (image).
CheckBoxField	Данная колонка показывает значение заданного поля в виде флажка (check box). Используется для отображения полей имеющих тип Boolean.
HyperLinkField	Данная колонка показывает значение заданного поля в виде гиперссылки. При нажатии на такую гиперссылку браузер буде переходить по заданному URL адресу.
ImageField	Данная колонка показывает значение заданного поля в виде свойства Src HTML тэга <img>. Содержанием связанного с колонкой поля должен быть URL адрес файла, содержащего изображение.
TemplateField	Показывает заданный разработчиком шаблон отображения связанного с колонкой поля источника данных. Данный тип колонки используется тогда, когда нужно создать колонку специального вида. Данный шаблон может включать любое количество полей данных вместе с текстом, изображениями и другими ЭУ.

Простые колонки

Элемент управления BoundField описывает колонку таблицы, связанную с полем источника данных, значение которой показывается в виде простого текста. Для задания отображаемого поля используется свойство DataField. С помощью свойства DataFormatString можно описать строку форматирования для отображения значений поля. Свойство NullDisplayText позволяет указать текст, который будет показываться, если поле имеет значение null. С помощью свойства Visible можно указать, чтобы объект класса BoundField не выводился на формируемой странице, а с помощью свойства ReadOnly можно запретить возможность редактирования данного поля.

Для вывода названия колонки в заголовке таблицы или текста нижней части колонки (в подвале, footer sections), нужно задать значения для свойств HeaderText и FooterText, соответственно. Если вместо тек-

стового названия колонки требуется показывать изображение, то в этом случае нужно задать значение свойству HeaderImageUrl.

Рассмотрим пример связывания ЭУ GridView с компонентом связывания с ИД. Если создан и настроен следующий компонент связывания с источником данных:

```
<asp:SqlDataSource ID="SqlDS1" runat="server"
      ConnectionString="<%$ ConnectionStrings:Northwind %>"
      SelectCommand="SELECT [ProductID], [ProductName], [UnitPrice],
            [QuantityPerUnit] FROM [Products]" >
</asp:SqlDataSource>
```

то его можно связать с ЭУ GridView следующим образом:

```
<asp:GridView ID="GridView1" runat="server" DataSourceID="SqlDataSource1">
</asp:GridView>
```

В этом случае свойство AutoGenerateColumns будет иметь значение true и для всех полей данных будут созданы колонки типа BoundField (рис. 4.23).

ProductID	ProductName	UnitPrice	QuantityPerUnit
1	Chai	18,0000	10 boxes x 20 bags
2	Chang	19,0000	24 - 12 oz bottles
3	Aniseed Syrup	10,0000	12 - 550 ml bottles
4	Chef Anton's Cajun Seasoning	22,0000	48 - 6 oz jars
5	Chef Anton's Gumbo Mix	21,3500	36 boxes
6	Grandma's Boysenberry Spread	25,0000	12 - 8 oz jars
7	Uncle Bob's Organic Dried Pears	30,0000	12 - 1 lb pkgs.
8	Northwoods Cranberry Sauce	40,0000	12 - 12 oz jars

Рис. 4.23. Автоматически сформированные колонки ЭУ GridView

Товары	Цена	Упаковка
Chai	18,0000	10 boxes x 20 bags
Chang	19,0000	24 - 12 oz bottles
Aniseed Syrup	10,0000	12 - 550 ml bottles
Chef Anton's Cajun Seasoning	22,0000	48 - 6 oz jars
Chef Anton's Gumbo Mix	21,3500	36 boxes
Grandma's Boysenberry Spread	25,0000	12 - 8 oz jars
Uncle Bob's Organic Dried Pears	30,0000	12 - 1 lb pkgs.

Рис. 4.24. Отображение данных в ЭУ GridView с использованием колонок BoundField

Для настройки ЭУ можно отменить автоматическое формирование колонок (задать AutoGenerateColumns=“false”) и включить описание простых колонок. Например:

```
<Columns>
      <asp:BoundField DataField="ProductName" HeaderText="Товары" />
      <asp:BoundField DataField="UnitPrice" HeaderText="Цена" />
      <asp:BoundField DataField="QuantityPerUnit" HeaderText="Упаковка" />
</Columns>
```

В результате получим отображение данных, показанное на рис. 4.24.

Колонки с кнопками

ЭУ ButtonField используется для создания колонки, которая содержит для каждой строки ЭУ Button (кнопку), инициирующий событие “щелчек”. С помощью такой кнопки можно инициировать на стороне сервера некоторое действие, которое нужно выполнить для заданной строки. При нажатии кнопки, выполняется обратная отправка на сервер и инициирование в web-приложении события RowCommand. Для этого события можно создать обработчик, который будет выполнять некоторые действия связанные с данной строкой.

На рис. 4.25 показан пример использования кнопок в строках данных.

Товары	Цена	
Chai	18,0000	Купить
Chang	19,0000	Купить
Aniseed Syrup	10,0000	Купить
Chef Anton's Cajun Seasoning	22,0000	Купить
Chef Anton's Gumbo Mix	21,3500	Купить

Рис. 4.25. Использование колонки с кнопками в ЭУ GridView

Данный ЭУ описывается с помощью следующего кода разметки:

```
<Columns>
   <asp:BoundField DataField="ProductName" HeaderText="Товары" />
   <asp:BoundField DataField="UnitPrice" HeaderText="Цена" />
   <asp:ButtonField ButtonType="Button" Text="Купить"
          CommandName="Add"/>
</Columns>
```

В данном примере, информация о каждом товаре показывается с помощью колонки BoundField. Для кнопки можно задать разные стили отображения (ButtonType) – Button (обычная кнопка), Link (ссылка) или Image (изображение). Когда пользователь щелкает по кнопке в одной из строк с описанием товаров, выполняется инициирование события RowCommand. Связывания события с обработчиком выполняется обычным способом с помощью свойства onRowCommand, Например: onRowCommand="GridView1_RowCommand1".

Если в строке будет несколько кнопок, то свойство CommandName позволит определить, какая из кнопок была нажата. Свойству CommandName должно быть задано любая уникальная строка символов, которую знает программный код, чтобы определить действия связанные

именно с данной кнопкой. При этом в свойстве CommandArgument параметра события (тип GridViewCommandEventArgs) будет передаваться номер строки, в которой выполнился щелчок кнопкой.

Например, колонка в виде кнопки может позволить добавлять товары в корзину пользователя. Например:

```
protected void GridView1_RowCommand1(object sender,
            GridViewCommandEventArgs e) {
   if (e.CommandName.Equals("Add")) {
      // получить индекс строки, в которой нажата кнопка (первая строка 0)
      int index = Convert.ToInt32(e.CommandArgument);
      // например, добавляем данный товара в корзину покупателя
      // AddToShoppingCart(index);
} }
```

В данном примере кнопочная колонка показывает фиксированный текст для всех элементов данных. Это задается с помощью свойства Text класса ButtonField. Если требуется связать текст кнопки с некоторым полем текущего элемента данных, то нужно задать свойству DataTextField имя поля.

Описанные шаблонами колонки

ЭУ TemplateField позволяет включить в таблицу колонку, содержание ячеек которой описывается с помощью заданного разработчиком шаблона разметки. При описании такого шаблона могут использоваться специальные тэги. Описание поддерживаемых данной колонкой тэгов приведено в табл. 4.12.

Таблица 4.12

Тэги, используемые в описании шаблона TemplateField

Тэги	Описание
ItemTemplate	Тэг для описания содержания и отображения ячеек колонки в нечетных строках таблицы.
AlternatingItem-Template	Тэг для описания содержания и отображения ячеек колонки в четных строк таблицы. Если данный шаблон не задается, то используется шаблон ItemTemplate.
HeaderTemplate	Тэг для описания содержания и отображения заголовка (header) колонки.
FooterTemplate	Тэг для описания содержания и отображения нижней части колонки (footer).
EditItemTemplate	Тэг для описания содержания и отображения ячейки в режиме редактирования. Данный шаблон должен включать ЭУ для ввода данных (input fields) и, возможно, валидаторы.

Описание шаблона может включать серверные ЭУ, строки текста, выражения связывания с данными. Выражения связывания с данными <%# Eval()%> позволяют вставлять значения полей текущей строки данных. В шаблоне можно использовать любое количество полей данных. Например, для колонки “Товары” можно описывать следующий специальный шаблон отображения:

```
<Columns>
   <asp:TemplateField headertext="Товары">
      <ItemTemplate>
         <b><%# Eval("ProductName")%></b> <br />
         в упаковке <%# Eval("QuantityPerUnit")%>
      </ItemTemplate>
   </asp:TemplateField>
   <asp:BoundField DataField="UnitPrice" HeaderText="Цена" />
</Columns>
```

В результате данные из ИД будут отображаться следующим образом:

Товары	Цена
Chai в упаковке 10 boxes x 20 bags	18,0000
Chang в упаковке 24 - 12 oz bottles	19,0000
Aniseed Syrup в упаковке 12 - 550 ml bottles	10,0000
Chef Anton's Cajun Seasoning в упаковке 48 - 6 oz jars	22,0000
Chef Anton's Gumbo Mix в упаковке 36 boxes	21,3500

Рис. 4.26. Отображение в колонке полей с помощью специального шаблона

Разбивка данных на страницы (Paging Data)

Для выполнения разделения данных на страницы в ЭУ GridView требуется только задать свойству AllowPaging значение true. Кроме этого можно задать либо количество строк на странице (свойство PageSize) или количество страниц (свойство PageCount). В этом случае в таблице показывается только набор записей из ИД и внизу показывается строка перехода к другим страницам (наборам) данных.

```
<asp:GridView ID="GridView1" runat="server" DataSourceID="SqlDataSource1"
      AllowPaging="True" PageSize="5">
```

В результате данные из ИД будут отображаться следующим образом:

Товары	Цена	
Chai	18,0000	Купить
Chang	19,0000	Купить
Aniseed Syrup	10,0000	Купить
Chef Anton's Cajun Seasoning	22,0000	Купить
Chef Anton's Gumbo Mix	21,3500	Купить
1 2 3 4 5 6 7 8 9 10 ...		

Рис. 4.27. Отображение табличных данных с разбивкой на страницы

Характеристики разделения страницы можно в значительной степени описывать с помощью тегов <PagerSettings> и <PagerStyle>. Например, можно задавать переходы между страницами с помощью номеров страниц или с помощью кнопок выполняющих переходы между страницами.

Сортировка данных

Для того, чтобы ЭУ GridView позволял выполнять сортировку строк таблицы по значениям в колонках, нужно в описании данного элемента добавить свойство AllowSorting="true", а в описание колонок добавить свойство SortExpression, которому обычно задается название поля данных, с которым связана колонка. В результате добавления этих свойств к приведенному ранее примеру, заголовки столбцов становятся кнопками (типа Link) при нажатии на которые строки таблицы будут сортироваться в соответствии со значения полей данной колонки (при одном нажатии по возрастанию, при следующем нажатии по убыванию).

Товары	Цена	
Cote de Blaye	263,5000	Купить
Thuringer Rostbratwurst	123,7900	Купить
Mishi Kobe Niku	97,0000	Купить
Sir Rodney's Marmalade	81,0000	Купить
Carnarvon Tigers	62,5000	Купить
1 2 3 4 5 6 7 8 9 10 ...		

Рис. 4.28. Сортировка данных по убыванию цены товаров

Редактирование данных

Если связанной с ЭУ GridView ИД поддерживает обновление, то данный ЭУ может автоматически выполнять редактирование данных. Для того, чтобы это можно было сделать нужно в описании компонента, связывающего с источником данных добавить команды Insert, Update и Delete (это проще всего сделать с помощью мастера настройки компо-

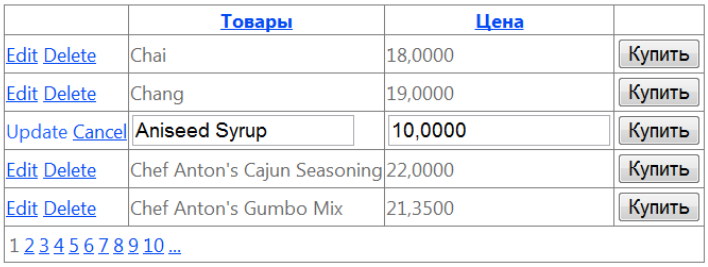
нента используя кнопку «Advanced» и задав опцию «Generate Insert, Update, и Delete statements», при этом должно быть выбрано ключевое поле таблицы).

ЭУ GridView может показывать колонку с соответствующими кнопками для каждой строки таблицы. Для реализации возможности редактирования данных нужно в описании ЭУ GridView задать свойство AutoGenerateEditButton="true". В этом случае ЭУ GridView выводит дополнительную колонку, как это показано на рис. 4.29.

	Товары	Цена	
Edit Delete	Chai	18,0000	Купить
Edit Delete	Chang	19,0000	Купить
Edit Delete	Aniseed Syrup	10,0000	Купить
Edit Delete	Chef Anton's Cajun Seasoning	22,0000	Купить
Edit Delete	Chef Anton's Gumbo Mix	21,3500	Купить
1 2 3 4 5 6 7 8 9 10 ...			

Рис.4.29. Отображение данных, позволяющее выполнять их редактирование и удаление

При выполнении пользователем щелчок по кнопки Edit, соответствующая ей строка переходит в режим редактирования и пользователь может менять данные в полях текущей записи.



	Товары	Цена	
Edit Delete	Chai	18,0000	Купить
Edit Delete	Chang	19,0000	Купить
Update Cancel	Aniseed Syrup	10,0000	Купить
Edit Delete	Chef Anton's Cajun Seasoning	22,0000	Купить
Edit Delete	Chef Anton's Gumbo Mix	21,3500	Купить
1 2 3 4 5 6 7 8 9 10 ...			

Рис.4.30. Выполнение редактирования строки таблицы в ЭУ GridView

4.8. Безопасность web-приложения

Для поддержки безопасности работы web-приложения должны выполнять аутентификацию и авторизацию пользователей. Процесс ***аутентификации*** предназначен для того, чтобы убедиться, что пользователь является тем, за кого он себя выдает. Для этого обычно web-приложение запрашивает у пользователя пароль (текстовую строку), который сравнивает со строкой, которая хранится в web-приложении. Если эти строки совпадают, считается, что пользователь прошел аутентификацию.

В процессе ***авторизации*** определяется, какие задачи может выполнять аутентифицированный пользователь и к каким ресурсам он

имеет доступ. Для всех ресурсов web-приложения могут быть заданы правила, которые устанавливают права пользователей для работы с ними. Для удобства авторизации пользователей выполняется разделение пользователей по ролям. Под ролью понимается группа пользователей имеющих одинаковые права.

В ASP.Net можно использовать два способа аутентификации:

- ***Встроенная Windows аутентификация***. Данный вид аутентификации выполняется web-сервером IIS и может применяться только для клиентов, работающих в локальной сети, использующей ОС Windows. В этом случае используются учетные записи пользователей, созданые в ОС Windows.
- ***Аутентификация на основе форм.*** Данный вид аутентификации предполагает создание в web-приложении специальной базы данных учетных записей и ролей пользователей.

Наиболее распространенным способом аутентификации web-приложений является аутентификация на основе форм. При использовании такой аутентификации для web-приложения создается специальная база данных, в которой хранится вся информация об учетных записях пользователей.

Для реализации аутентификации на основе форм требуется выполнить следующие действия:

- Создать базу данных для хранения учетных записей пользователей.
- Настроить конфигурационные файлы, расположенные в корневом каталоге и подкаталогах web-приложения.
- Использовать элементы управления, выполняющие работу с учетными записями пользователей.

Создание базы данных учетных записей пользователей

Базу данных для хранения учетных записей пользователей можно создать двумя способами:

- Если на компьютере разработчика установлена СУБД SQL Express (2005 или 2008), то при выполнении команды «Website=>ASP.Net Configuration» будет создана файл базы данных AspNetDb.mdf, который будет сохранен в папке App_Data. Кроме этого в файл конфигурации будут включены все описания, требуемые для организации работы web-приложения с данной БД.
- Если СУБД SQL Express (2005 или 2008) на компьютере разработчика не установлен, а также, если требуется использовать не локальный файл БД, а серверную СУБД, то базу данных можно создать с

помощью утилиты aspnet_regsql.exe. Данная утилита находится в папке, в которой находтся все файлы используемой версии платформы Microsoft.Net. Например: C:\Windows\Microsoft.NET\Framework\v4.0.30319.

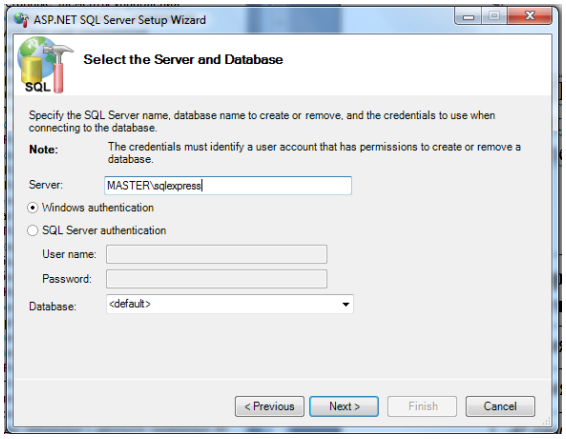


Рис. 4.31. Задание параметров подключения к SQL серверу в утилите aspnet_regsql.exe

При запуске данной утилиты нужно указать параметры подключения к используемой СУБД SQL Server (имя сервера, имя пользователя, пароль, имя создаваемой базы данных).

В результате выполнения данной утилиты будет создана база данных, включающая все требуемые таблицы и хранимые процедуры для поддержки безопасности работы web-приложения.

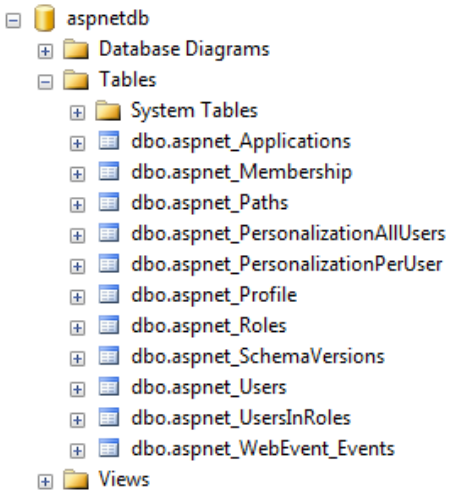


Рис. 4.32. Состав таблиц в базе данных, созданной утилитой aspnet_regsql.exe

После создания БД с помощью утилиты aspnet_regsql.exe необходимо выполнить конфигурирование параметров безопасности web-приложения добавив нужную информацию в файл конфигурации.

Конфигурирование параметров безопасности web-приложения

Для того, чтобы web-приложение могло использовать созданную БД необходимо в элементе <connectionStrings> файла конфигурации описать строку соединения с БД, а также описать провайдеров, поддерживающих работу с учетными записями (элемент <membership>) и ролями (элемент <roleManager>) пользователей.

Пример описания строки соединения и провайдеров показан ниже:

```
<connectionStrings>
   <add name="MyConnString" connectionString="Data Source=(local)\sqlexpress;
   Initial Catalog=aspnetdb;Integrated Security=True"
   providerName="System.Data.SqlClient" />
   . . .
</connectionStrings>

<membership defaultProvider="MyMembershipProvider">
   <providers>
   <add name="MyMembershipProvider" connectionStringName="MyConnString"
         enablePasswordRetrieval="false" enablePasswordReset="true"
         maxInvalidPasswordAttempts="5" minRequiredPasswordLength="5"
         minRequiredNonalphanumericCharacters="0"
         requiresQuestionAndAnswer="false" requiresUniqueEmail="false"
         passwordFormat="Hashed"
         type="System.Web.Security.SqlMembershipProvider" />
   </providers>
</membership>

<roleManager enabled="true" defaultProvider="MyRoleProvider">
   <providers>
      <add name="MyRoleProvider" connectionStringName="MyConnString"
      type="System.Web.Security.SqlRoleProvider" />
   </providers>
</roleManager>
```

Смысл атрибутов в данном описании во многом понятен по их имени. Например: minRequiredPasswordLength – устанавливает количество минимальную требуемую длину пароля пользователя.

Для задания параметров поддержки безопасности web-приложения используются разделы <authentication> и <authorization> конфигурационного файла. В разделе <authentication> задается способ выполнения аутентификации пользователей и определяются его параметры. Например, аутентификация с использованием форм может быть задана следующим образом:

```
<authentication mode="Forms">
      <forms name="MyCookieName" loginUrl="LogPage.aspx" timeout="20"/>
</authentication>
```

В элементе form атрибуту loginUrl задается ссылка на web-форме, к которой нужно будет перенаправить запрос не прошедщего аутентификацию пользователя (в данном случае это web-форма LogPage.aspx), а атрибут timeout задает время между запросами (в минутах), после которого пользователю потребуется заново выполнить аутентификацию.

Для того, чтобы web-приложение начало требовать аутентификацию пользователей нужно указать, что только аутентифицированные пользователи могут использовать ресурсы данного каталога. Для этого используется описание авторизации, которое выполняется с помощью элемента <authorization>. Приведенное ниже описание указывает, что только аутентифицированные пользователи могут использовать ресурсы каталога, в котором находится конфигурационный файл:

```
<authorization>
    <deny users="?" />
</authorization>
```

Задание параметров аторизации доступа к ресурсам web-приложения

Пользователи отнясящиеся к разным ролям имеют разные права доступа к ресурсам web-приложения. Например, пользователи-администраторы (роль admin) могут иметь право создавать учетные записи для новых пользователей, а другие пользователи такого права могут и не иметь.

Для управления правами доступа к различным web-формам используется авторизация. Обычным подходом является размещение файлов, которые требуют специальных разрешений в отдельные подкаталог. А после в каждый подкаталог включается файл web.config, в котором задаются правила авторизации. Такие правила описываются в элементе <authorization> раздела <system.web> файла web.config. Они включают списки пользователей и ролей, которым разрешен доступ к ресурсам подкаталога, и списки пользователей и ролей, которым запрешен доступ к ресурсам подкаталога:

```
<authorization>
  <deny users="разделенный запятыми список пользователей"
        roles="разделенный запятыми список ролей" />
  <allow users="разделенный запятыми список пользователей"
        roles="разделенный запятыми список ролей"/>
</authorization>
```

Кроме этого, есть два специальных символа для задания пользователей:

- Символ (?) обозначает всех анонимных пользователей. Например, правило <deny users="?" /> говорит о том, что доступ анонимных пользователей не разрешен.
- Символ (*) обозначает всех пользователей. Например, правило <allow users="*" /> говорит о том, что всем пользователи доступ к ресурсу разрешен.

Если в разделе задано <authorization> несколько правил, то система последовательно проверяет правила до тех пор, пока не очередное правило не выполнится (т.е., пока не будет запрещено или разрешено). Следующие за ним правила не проверяются. Если ни одно из правил не выполняется, то считается, что доступ к ресурсу разрешен.

Например:

```
<configuration>
   <system.web>
      <authorization>
         <deny users="?" />
         <allow roles="admin" />
         <deny users="*" />
      </authorization>
   </system.web>
</configuration>
```

Данный набор правил задает следующее. Первое правило: если пользователь не аутентифицирован, то в доступе к ресурсу подкаталога отказать и перенаправить на форму для выполнения аутентификации. Второе правило: если аутентифицированный пользователь имеет роль admin, то работу с ресурсами подкаталога разрешить. Третье (последнее правило): всем аутентифицированным пользователям, которые не относятся к роли admin в работе с ресурсами отказать.

При разработке web-приложения можно создавать учетные записи пользователей и роли с помощью специального web-приложения WAT, которое вызывается с помощью команды «Website=>ASP.NET Configuration». Используя данный инструмент можно создать учетные записи пользователе и их роли (рис. 4.33) и выполнять их корректировку.

Рис. 4.33. Создание учетной записи пользователя с помощью программы WAT

Серверные ЭУ для работы с учетными записями пользователей

В составе технологии ASP.Net Web Forms имеются следующие ЭУ для создания и использования учетных записей пользователей:

- Login – ЭУ для подключения пользователя (ввод имени и пароля) и проверки его соответствия данным, которые содержатся в БД. Если данные совпадают (т.е. пользователь прошел аутентификацию, то выполняется переход к запрашиваемой странице).
- LoginView – позволяет показывать разную информацию для подключенных пользователей. Например, можно использовать эту страницу для отображения информации, которая доступна только аутентифицированным пользователям.
- LoginStatus – ЭУ, показывающий ссылку на страницу подключения для пользователей, которые не были аутентифицированы (Logout) и ссылку на страницу отключения для подключенных пользователей (Login).
- LoginName – ЭУ, показывающий текущее имя пользователя, если он подключен к системе.
- PasswordRecovery – ЭУ для выполнения восстановления пароля пользователей, путем отправки e-mail сообщения или при ответе пользователя на секретный вопрос.
- CreateUserWizard – ЭУ, который собирает информацию о новом пользователе и создает в БД новую учетную запись.
- ChangePassword – ЭУ, позволяющий подключенному пользователю сменить пароль.

Элемент управления Login предоставляет готовый к использованию интерфейс, который запрашивает имя и пароль пользователя. Он включает кнопку с атрибутом CommandName="Login" для подключения пользователя. При нажатии пользователем на данную кнопки ЭУ автомати-

чески выполняет проверку соответствия введенного имени и пароля пользователя с данными, содержащимися в БД, а затем вызывает переход к запрашиваемой web-форме приложения. Данный ЭУ является полностью расширяемым и позволяет переопределить его разметку, стиль и свойства, а также самому обрабатывать события, чтобы изменить стандартное поведение. Пример настройки описания ЭУ Login показан ниже:

```
<%@ Page Language="C#" AutoEventWireup="true"
      CodeFile="LogPage.aspx.cs" Inherits="LogPage" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server"><title></title></head>
<body>
   <form id="form1" runat="server">
      <div>
      <asp:Login ID="Login1" runat="server" BackColor="aliceblue"
            BorderColor="Black" BorderStyle="double">
         <LayoutTemplate>
            <h4 align="center">Подключение к системе</h4>
            <table>
               <tr> <td> Имя пользователя: </td>
               <td> <asp:TextBox ID="UserName" runat="server" />
               <asp:RequiredFieldValidator ID="UserNameRequired" runat="server"
                     ControlToValidate="UserName" ErrorMessage="*" />
               </td> </tr> <tr> <td> Пароль:</td>
               <td>
               <asp:TextBox ID="Password" runat="server" TextMode="Password" />
               <asp:RequiredFieldValidator ID="PasswordRequired" runat="server"
                     ControlToValidate="Password" ErrorMessage="*" />
               </td> </tr>
               <tr> <td> <asp:CheckBox ID="RememberMe" runat="server"
                     Text="Запомни меня" /> </td>
               <td align="center">
                  <asp:Button ID="Login" CommandName="Login"
                        runat="server" Text="Войти" />
               </td> </tr>
            <table>
         </LayoutTemplate>
      </asp:Login>
      </div>
   </form>
</body>
</html>
```

Отображение настроенного выше ЭУ Login показано на рис. 4.34.

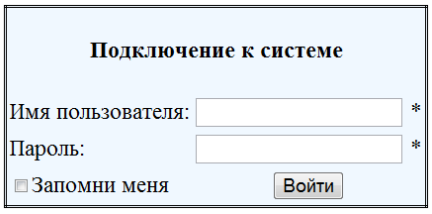


Рис. 4.34. Отображение настроенного ЭУ Login

ЭУ LoginView является достаточно простым и мощным. Он позволяет показывать различные наборы ЭУ для анонимный и аутентифицированных пользователей. Кроме этого он также позволяет показать разное содержание с учетом роли, к которой относится подключенный пользователь. ЭУ LoginView является шаблонным ЭУ с разными типами шаблонов – один для анонимных пользователей, другой для аутентифицированных пользователей, а третий для поддержки шаблонов учитывающих роли. В этих шаблонах требуется просто добавить ЭУ для отображения соответствующей ситуации, как это показано ниже. В данном ЭУ показывается простой текст для анонимных пользователей и некоторый текст для зарегистрированных пользователей:

```
<asp:LoginView ID="LoginViewCtrl" runat="server">
   <AnonymousTemplate>
      <h2>Вы являетесь анонимным пользователем</h2>
   </AnonymousTemplate>
   <LoggedInTemplate>
      <h2>Вы подключились к web-приложению</h2>
   </LoggedInTemplate>
   <RoleGroups>
      <asp:RoleGroup Roles="Admin">
         <ContentTemplate>
            <h2>Только администраторы видят это содержание</h2>
         </ContentTemplate>
      </asp:RoleGroup>
      <asp:RoleGroup Roles="Reader, Designer">
         <ContentTemplate>
            <h2>Это содержание для web дизайнеров и читателей </h2>
         </ContentTemplate>
      </asp:RoleGroup>
   </RoleGroups>
</asp:LoginView>
```

Примеры использования ЭУ для обеспечения безопасности web-приложения можно посмотреть в папке Account стандартного каркаса web-сайта ASP.Net Web Site.

4.9. Создание и использование ASMX Web сервисов

Помимо создания web-приложений технология ASP.Net позволяет создавать и web-сервисы (web-сервисы описаны в разделе 2.4). Однако следует отметить, что в настоящее время более совершенной является технология создания сервисов Windows Communication Foundation (WCF). Для понимания web-сервисов, учитывая ограниченный объем пособия, в данном разделе поясняется, только создание ASMX Web сервисов.

Для создания web-сервиса используется шаблон проекта ASP.Net Web Service, который имеется в платформе «.Net Framework 3.5» (рис. 4.35).

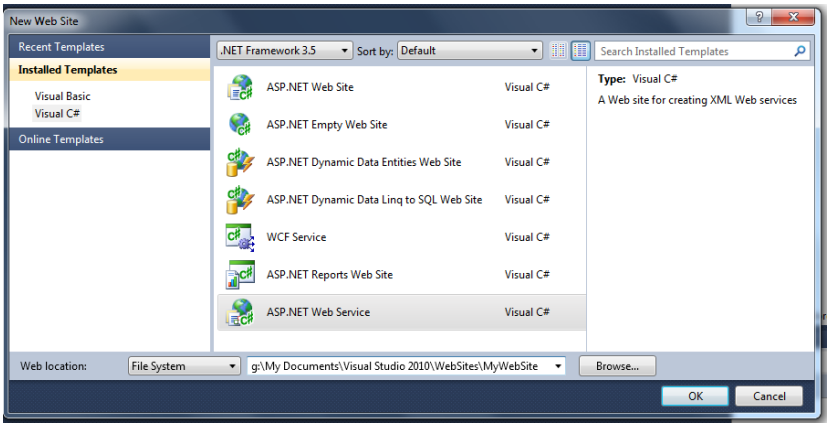


Рис. 4.35. Создание шаблона web-сервиса

В результате выбора данного шаблона будет создан набор файлов для создания web-сервиса с именем Service. Для того чтобы создать сервис с другим именем, например, MathService, лучше имеющийся сервис удалить (файлы Service.asmx и Service.cs в подкаталоге App_Code), а затем добавить к проекту файлы с другими именами. Для этого нужно выполнить команду «Website=>Add New Item», выбрать в диалоговом окне шаблон файла «Web Service» и задать ему новое имя, например, MathService. В результате этого будут созданы два файла MathService (\в корневом каталоге) и MathService.cs (в подкаталоге App_Code).

ASMX Web сервисы это файлы с расширением *.asmx, в которых содержится директива не страницы Page, а web-сервиса WebService. На пример:

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/MathService.cs" Class="MathService" %>
```

Программный код сервиса содержится в связанном файле, заданном в свойстве CodeBehind. В приведенном примере программный код сервиса хранится в файле MathService.cs (в подкаталоге App_Code).

Web-сервис представляет собой обычный класс на языке C#, (в данном случае MathService), производный от базового класса WebService (из пространства имен System.Web.Services). При описании данного класса используются специальные атрибуты WebService и WebServiceBinding, указывающие общей среде выполнения, как выполнять работу с объектами данного класса. Методы класса сервиса, которые будут доступны по протоколу SOAP должны иметь атрибут WebMethod.

В примере показанном на рис. 4.36 показан сервис, содержащий два метода: Add (суммирование двух чисел) и Subtract (вычитание двух чисел).

```
using System;
using System.Collections.Generic;
using System.Web;
using System.Web.Services;

[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class MathService : WebService {
    [ WebMethod ]
    public int Add(int a, int b) {
        return a + b;
    }
    [ WebMethod ]
    public void Subtract(int a, int b, out int c) {
        c = a – b;
        return;
    }
}
```

Рис. 4.36. Программный код web-сервиса.

Если созданный web-сервис запустить на выполнение (нажать кнопку F5), то в браузере будет показана web-страница с описанием данного сервиса (рис. 4.37).

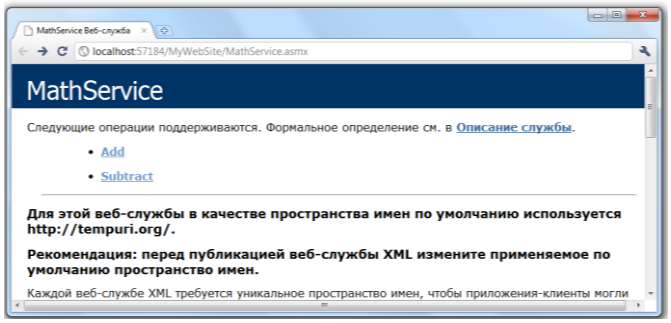


Рис. 4.37. Web-страница описанием разработанного web-сервиса

При нажатии на ссылки «Описание службы» в браузере будет показано WSDL описание данного web-сервиса, которое будет применять-

ся для создании прокси классов в программах использующих данный сервис (рис. 4.38).

Рис. 4.38. Web страница с WSDL описанием web-сервиса

При нажатии на названия одного из методов, включенного с данный сервис, будут показана форма, позволяющая ввести исходные данные для выбранного метода и проверить его работу (рис. 4.39).

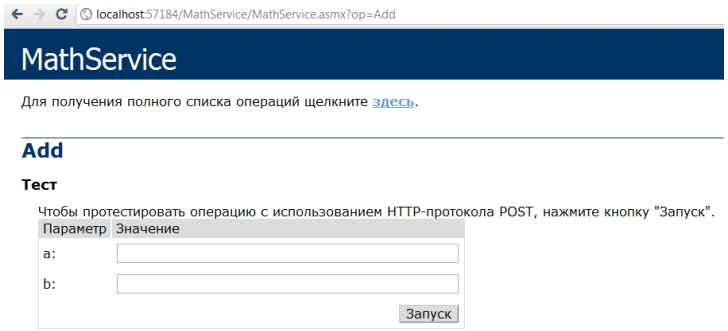
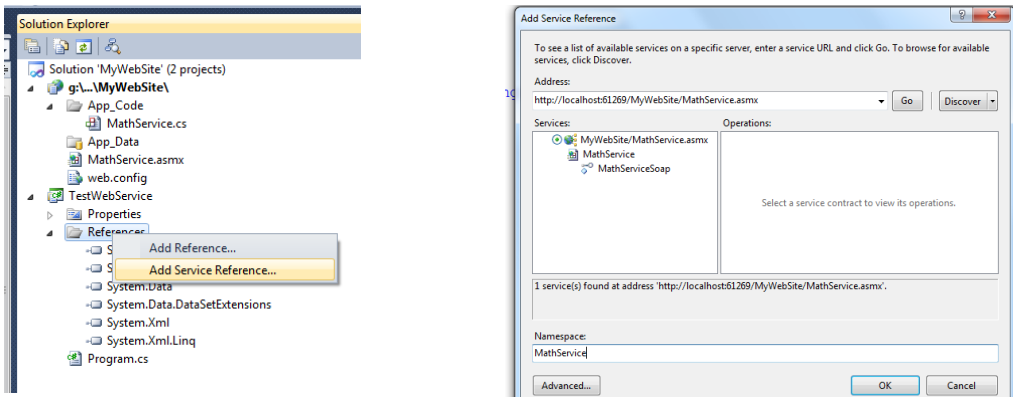


Рис. 4.39. Web страница для тестирования метода Add web-сервиса

Созданный web-метод может быть использован в web-формах, в скриптах JavaScript и даже в локальных программах. Рассмотрим пример использования созданного web-сервиса в консольной программе. Для этого нужно в проекте разрабатываемой программы создать прокси-класс (proxy class), который будет выполнять взаимодействие с web-сервисом. Такой прокси класс можно создать с помощью среды разработки Visual Studio, выбрав контекстную команду «Add Service Reference» для папки проекта Reference (рис. 4.40 a). В результате выполнения данной команды будет показано диалоговое окно, в котором нужно указать URL адрес web-сервиса (в данном случае это http://localhost:61269/MyWebSite/MathService.asmx) и задать имя пространства, в котором будет создаваться прокси класс (рис. 4.40 b).

В результате выполнения ссылки на web-сервисе в проекте создается прокси класс MathServiceSoapClient (рис. 4.30).



(а) (б)

Рис. 4.40. Создание прокси класса для web-сервиса
(а) контекстная команда добавления в проект ссылки на web-сервис
(b) диалоговое окно добавления в проект ссылки на web-сервис

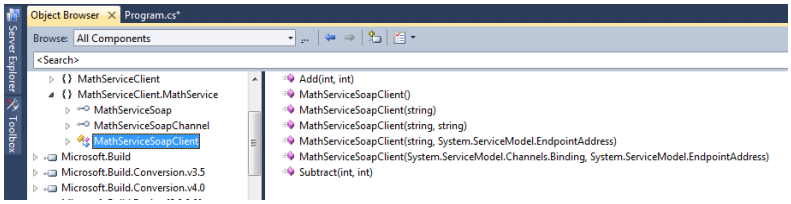


Рис. 4.41. Класс MathServiceSoapClient, созданный с помощью ссылки на web-сервис

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace TestWebService {
    class Program{
        static void Main(string[] args) {
            MathService.MathServiceSoapClient math =
                    new MathService.MathServiceSoapClient();
            int a = 7, b = 3, c;
            c = math.Subtract(a, b);
            Console.WriteLine("Вычитаем {0} – {1} = {2}", a, b, c);
            Console.ReadLine();
} } }
```

Рис. 4.42. Использование web-сервиса с помощью прокси-класса (класса MathServiceSoapClient

Теперь, чтобы обратиться к web-сервису в коде приложения нужно создать объект прокси-класса MathServiceSoapClient и вызвать один из его методов (например, Subtract) (рис. 4.42).

5. Проектирование web-приложений

Web технологии позволяют разрабатывать как простые web-приложения, так и сложные информационные web-системы. Для успешной разработки таких приложений требуется понимать общий процесс разработки программного обеспечения (ПО): основные действия по разработке, которые должны быть выполнены; как эти действия взаимосвязаны между собой; какой порядок их выполнения; какие специалисты, должны принимать участвие в этом процессе.

Разработка реальных web-приложений возможна только в результате коллективной работы. В зависимости от размера приложения и участвующих в его создании специалистов, разработка может быть сложным делом, подверженным разным рискам, которые могут влиять на успех конечного результата.

5.1. Организация разработки web-приложений

Разработка программного обеспечения это творческий процесс, который ведут к созданию инновационных программных продуктов и систем. Обычно процесс разработки программного обеспечения не является единым монолитным блоком работ, который принимает на входе некоторые идеи о новом приложении, а в качестве результата выдает решение (программный продукт), полностью соответствующее данной идеи.

Процесс разработки может быть разделен на набор базовых работ с хорошо определенными границами и смыслом:

- понимание проблемы;
- планирование решения;
- выполнение плана;
- проверка точности полученного результата;
- доработка с целью удаления возможных ошибок или неточностей.

Основные вида работ по разработке web-приложений

В процессе разработки любого ПО можно выделить следующие основные виды деятельности:

- ***Определение требований к ПО*** (инжиниринг требований): предназначено для понимания решаемой проблемы.
- ***Проектирование***: предназначено для планирования решения проблемы.
- ***Реализация***: преобразование плана в работающий программный код.

- ***Проверка и оценка качества***: предназначено для выявления ошибок кодирования (программного кода) или не соответствий между определенными требованиями и их реализацией.
- ***Развертывание***: предоставление пользователям возможности работать с созданным ПО.
- ***Поддержка***: предназначено для отслеживания использования работающей системы и сохранение ее работоспособности.
- ***Развитие***: предназначено для улучшения со временем разработанного решения, предоставление новых входных данных для процесса разработки в форме новых требований.

Инжиниринг требований предназначен для понимания требуемых возможностей и характеристик создаваемого ПО. Данный анализ направлен на определение функциональных требований (какие функции система должна выполнять) и не функциональных требований (качество предлагаемого решения). Инжиниринг требований также предполагает выявление общей идеи, которая стоит за разрабатываемой системой, основных заинтересованных лиц, которым требуется новая система и условия, в которых будет использоваться система. Выявленные требования обрабатываются с целью создания высокоуровневых моделей данной системы, которая абстрагируется от не нужных подробностей рассматриваемой проблемной области

Проектирование предназначено для описания решения, которое должно соответствовать функциональным требованиям и требованиям эффективности, а также ограничениям той среды, в которой она будет работать. Ранее собранные требования уточняются и улучшаются, чтобы удовлетворять возможным технологическим ограничениям. Проектирование включает такие действия, как:

- Проектирование схемы данных и классов.
- Проектирование компонент.
- Проектирование графического интерфейса.
- Проектирование архитектуры системы.

Помогает лучше сформировать специфические особенности системы, такие, как: структура, поведение, взаимодействие, данные и поток управления. Позволяет разделить области ответственности, основной принцип программной инженерии: решение проблемы путем разделения на разные подзадачи может помочь справится со сложностью и достичь требуемых технических качеств, таких, как адаптируемость, простоту поддержки, расширяемость и многократная используемость.

На этапе ***реализации проекта*** разработанные проектные решения преобразуются в соответствующий программный код (вручную или с помощью инструментов автоматизации программирования). Могут потребоваться библиотеки программ, разные языки программирования, разные коммуникационные протоколы и технические устройства.

Проверка и оценка качества обычно проводится параллельно с реализацией, так как правильность и надежность промежуточных результатов, а не только конечного продукта является очень важным для гарантирования качества всего приложения. Качество в значительной степени связано со следующими критериями:

- оценкой функциональности, т.е. правильности поведения приложения относительно заданных функциональных требований.
- оценкой производительности (т.е. время ожидания отклика приложения в обычных условиях и при пиковых нагрузках).
- оценкой удобства использования (usability), т.е. легкость использования, коммуникационная эффективность и соответствие стандартам использования.

Развертывание приложения предоставляет пользователям возможность использования данного приложения. В зависимости от типа приложения, процесс развертывания может включать:

- установка ПО на компьютерах клиентов;
- установка центрального приложения и баз данных на сервере;
- конфигурирование промежуточного коммуникационного ПО;
- инструктирование и обучение будущих пользователей, в особенности, если устанавливается совсем новое приложение, а не новая версия уже существующего приложения.

Поддержка развернутого и работающего приложения означает обеспечение его рабочего состояния, которое состоит в гарантировании его доступности и уменьшении сбоев. Может включать: периодическую проверку файлов журнала; отчетов об ошибках и очистке временных файлов; исправление ошибок; установку исправлений.

Развитие приложения. Приложение предназначено для решения реальных задач пользователей. Однако, в связи с развитием организации, усложнением решаемых задач, улучшением понимания пользователем возможностей данного ПО, неизбежно потребуется развивать созданное ПО. Новые потребности пользователей появляются только после того, как они некоторое время поработают с созданным ПО. После этого они начинают давать свои предложения и комментарии. Появление новых требований может вызвать необходимость запустить весь

процесс разработки заново. Несмотря на строгое следование рекомендациям правильной организации процесса разработки ПО, часто только после развертывания и накопления некоторого опыта работы пользователей с ПО становится ясно, что некоторые требования не были полностью выполнены и приложение должно быть доработано.

Особенности разработки web-приложений

Web-приложения имеют архитектурные, технологические и пользовательские особенности:

- web-приложения должны развертываться очень быстро и требования к ним, вероятно, меняются в ходе этапа разработки;
- все более частой практикой в web разработке становится замена документов на реальные прототипы приложения;
- очень раннее включение конечных пользователей к тестированию и развитию web приложения;
- в то время, как при разработке обычных приложений создается только одна версия приложения, которая удовлетворяет всем заданным требованиям, для web-приложений все более частой (и желательным) становится публикация в web-сети приложения, которое еще не удовлетворяют всем заданным требованиям;
- получение ранних отзывов пользователей становятся все более важным для разработки web-приложения, а цикл развития действительно начинает рассматриваться в качестве реальной возможности для улучшения web-приложения, а не как дополнительная трата времени на доведения приложения до рабочего состояния;
- работа по проектированию web-приложений делится на проектирование данных и проектирование навигации.

Разработка web-приложений включает специфические процессы:

- web-приложения являются специальным видом обычных программных приложений и поэтому web-инженерия может рассматриваться в качестве специальной разновидности программной инженерии;
- разработка приложений для web сети предполагает использование нескольких хорошо определенных правил и соглашений, которые предоставляют стабильно работающую, устойчивую и масштабируемую среду разработки и выполнения;
- существуют специальные процессы разработки, которые учитывают специфические возможности web-приложений.

Модель жизненного цикла современных web-приложений показана на рис. 5.1.

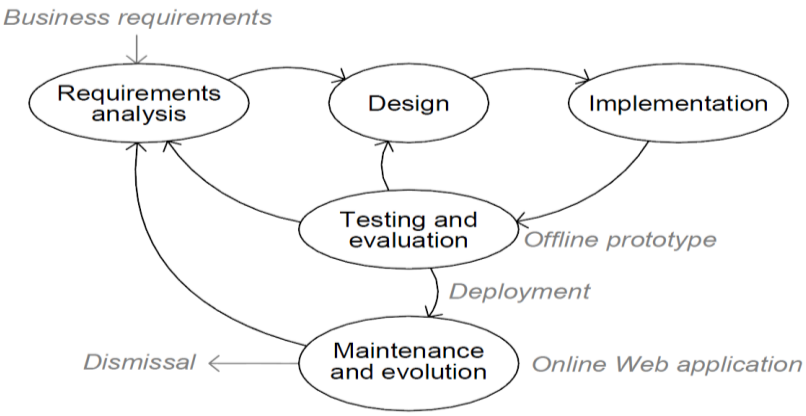


Рис. 5.1. Модель жизненного цикла современных web-приложений

Данная модель состоит из 5 основных видов деятельности и семи переходам между ними. К основным видам деятельности по разработке web-приложений относятся:

1. анализ требований;
2. проектирование;
3. реализация;
4. тестирование и оценка;
5. поддержка и развитие.

Основное различие данной модели от модели разработки обычного ПО заключается в понимании “развертывания приложения” не как вида деятельности (работы), а как перехода между работами.

В web-сети развертывание приложений для его пользователей действительно не является трудоемкой работой, так как в связи с централизованной архитектурой обычных web приложений; отсутствием специального кода приложения на стороне пользователей; использованием браузера в качестве среды выполнения, работа по развертыванию web приложения в значительной степени облегчается и ускоряется.

Данная модель предлагает явное соединение работы по «поддержке и развитию» с «анализом требований». Именно этот переход характеризует модель, он создает второй цикл модели, который включает «анализ требований». Данный цикл называется «циклом развития». Первый цикл связывает «проектирование», «реализацию» и «тестирование и развитие». Данный цикл называется «циклом построения и проверки». Такие два цикла соответствуют двум этапам, которые характерны для современных web-приложений: офф-лайн (off-line) разработка и он-лайн (on-line) разработка. Off-line разработка заключается в создании web-приложения в среде Visual Studio без использования реального web-сервера. Разработка в режиме on-line выполняется после перенос web-приложения под управление реального web-сервера: создание вир-

туального каталога; файлов настройки; управление web-сервером. Используя Visual Studio можно работать и с web-приложением, которое установлено в web-сервере.

5.2. Основные участники разработки web-приложений

В выполнении разработке web-приложений могут принимать участие следующие виды специалистов: аналитик приложения, архитектор приложения, администратор приложения, программисты, web-дизайнеры, авторы контента, менеджер контента, web-мастер.

Аналитик приложения, на этапе определения требований, собирает информацию о причинах (мотивации), вызвавших разработку данного приложения и преобразует их в спецификацию (детальное описание) требований к приложениям. При этом он оценивает долговременные стратегические бизнес-цели и ограничения, а также преобразует их в кратковременные, конкретные требования к приложению.

На этапе проектирования приложения, ***архитектор данных*** анализирует требования, которые связаны с контентом и данными прикладной области. На основе этого он разрабатывает концептуальную модель данных, которая организует данные в структуру и представление, которое может быть доступно и может использоваться в приложении.

Архитектор приложения анализирует требования к функциям и сервисам, которые должны быть реализованы приложением. На основе этого он разрабатывает концептуальное решение логики работы приложения (описанной с помощью моделей, рисунков и языков описания спецификаций), которые разрабатываются на основе модели данных.

На основе созданных спецификаций, ***программист*** (или разработчик), реализуют решения, которые схематично описаны архитекторами данных и приложения, выполняют тестирование и отладку реализованных решений. В большинстве случаев, программисты также управляют развертыванием приложения.

Администратор приложения является основным участником развертывания и развития приложения. Он отвечает за поддержку приложения; периодическое выполнение резервного копирования; управление сообществом пользователей; сбор отзывов пользователей.

Непосредственно в разработке web-приложений, помимо программистов и web-дизайнеров, также принимают участие следующие специалисты: web-мастер; автор контента; менеджер контента.

Программисты являются ответственными за программную реализацию логики работы приложения и работы с контентом (содержани-

ем баз данных, наборами документов, изображениями, мультимедиа и т.п.). Они должны уметь описывать сложную логику работы приложений и сложные запросы для получения требуемых данных. Программисты также выполняют тестирование и отладку реализованных решений.

Web дизайнеры отвечают за разработку внешнего вида (дизайна) страниц web-приложения, который является очень важным, как для удобства использования, так и для привлечения пользователей. На основе нефункциональных требований, связанных с корпоративными графическими особенностями заказчика и известных стандартов коммуникации: определяется графический внешний вид приложения; структурируется контент и графика в схемы (структуры) страницы; выбираются подходящие свойства стилей (например, шрифты, цвета, и размеры изображений). ***Web-дизайнеры*** обычно знают языки форматирования (HTML, CSS) web-страниц и умеют работать с программными системами проектирования и оформления страниц, такими, как Macromedia Dreamweaver или Microsoft Expression Studio. Кроме этого, они также могут иметь некоторые знания таких языков, как XML и XSLT. Но они не являются программистами и не имеют опыт в описании логики работы приложения и составлении программ.

Для дизайнера не требуется знания программирования. Они должны уметь работать с программными инструментами оформления и разработки графических интерфейсов прототипов страниц.

Проблема разработки web приложений во многом связана с тем, что программисты и дизайнеры совместно работают с одними и теми же web-страницами. Технологии разработки должны обеспечить максимальную независимость работы программистов и дизайнеров друг от друга.

Оба набора таких знаний и умений являются очень специализированными; они требуют большого опыта и критически важны при разработке web-приложений. Однако, также, как нельзя требовать от дизайнера страниц создания хранимых SQL процедур, также нельзя требовать и от программистов проектирования и оформления web-страниц. Обычно приложения, в которых программисты выполняли проектирование пользовательского интерфейса, имеют проблемы с удобством его использования. Аналогично, приложения, которые были задуманы и построены web-дизайнерами имеют проблемы с масштабированием и их трудно поддерживать.

Хотя некоторые программисты полагают, что они обладают умением достаточным для выполнения ***всех*** задач, связанных с формированием динамических страниц, опыт показывает, что разработка сложных web приложений является результатом совместных усилий дизайнеров

и программистов. Для достижения максимальной производительности, специалисты с разными умениями должны иметь возможность работать независимо над компонентами, которые больше всего подходят к их опыту.

Автор контента создает новый контент (например, новые статьи, документацию, фотографии, сообщения блогов и т.п.), который добавляется и публикуется web приложением.

Менеджер контента отвечает за сбор контента, оценку контента, обеспечение качества и конечную публикацию.

Web-мастер отвечает за поддержку и частично развитие web-приложений. Обычно, каждое web-приложение, доступно в web-сети, предлагает где-то (например, на странице контактов или в нижней части страниц) возможность связаться с web-мастером, чтобы например, сообщить о не работающих ссылках и других проблемах работы web-приложения.

Кроме этого весь процесс разработки web-приложений также включает реальных пользователей приложения, в особенности на этапе оценки удобства использования приложения и его развития во времени.

Понятно, что не все перечисленные специалисты участвую в разработке любых web-приложений. Некоторые функции могут выполняться одновременно одними и теми же специалистами.

5.3. Современные методологии разработки web-приложений

Наиболее известными современными методологиями проектирования и разработки web-приложений являются:

- WebML – метод разработки и язык Web Modeling Language;
- WSDM – один из первых методов разработки web приложений Web Site Design Method.

Методология WebML

Методологии WebML является подходом к разработке web-приложений на основе модели. Основной вклад WebML состоит в разработке набора понятий, обозначений и методик для создания web-приложений активно использующих данные, которые могут применяться командами разработчиков для поддержки всех видов работ жизненного цикла приложений, от анализа до развертывания и развития. Объединяет традиционные приемы хорошо известные разработчикам, такие, как: сценарии использования на языке UML и концептуальное проектирование данных с помощью модели Entity-Relationship, с новыми поня-

тиями и методами для проектирования гипертекстов, которые являются важными для web-приложений.

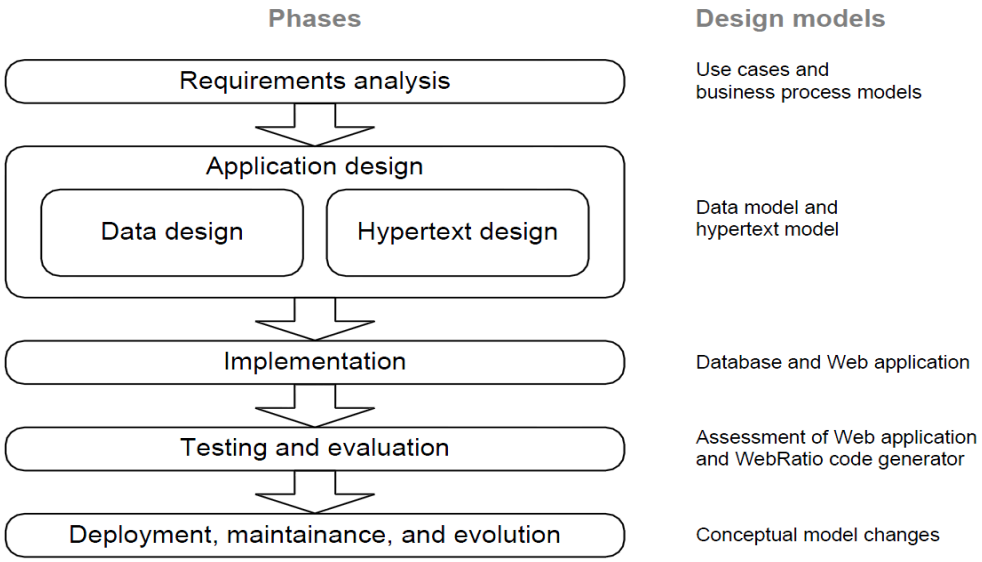


Рис. 5.2. Этапы модели разработки WebML

Процесс WebML является итеративным и инкрементальным. В данном процессе разные этапы повторяются и совершенствуются до получения результатов, которые полностью соответствуют требованиям к приложению. В связи с этим жизненный цикл продукта проходит несколько итераций, на каждом из которых создаются прототипы или частичные версии приложения. На каждой итерации, текущая версия приложения проверяется и оценивается, а затем расширяется или модифицируется для того, чтобы удовлетворять, как уже собранным требованиям, так и вновь появившимся.

Анализ требований WebML состоит в выполнении следующих работ:

- Выявление *групп пользователей*, для которых разрабатывается приложение. Каждая группа описывает пользователей, имеющих один и тот же профиль, или выполняющих одинаковые виды деятельности с одинаковыми правами доступа к одним и тем же классам информации.
- Спецификация *функциональных требований*, которые связаны с функциями, предоставляемыми пользователям. Для каждой группы пользователей выявляются и описываются релевантные виды работ, которые должны выполняться приложением; каждый вид работ является согласованным набором элементарных задач.
- Выявление *базовых информационных объектов*, т.е. основных информационных активов, к которым может быть предоставлен доступ пользователям и которыми он может манипулировать.

- Декомпозиция web приложения на представления сайта, т.е. разных гипертекстов, спроектированных таким образом, чтобы удовлетворять хорошо описанному набору функциональных требований и требований пользователей. Для каждой группы пользователей создается, по крайней мере, одно представление сайта, поддерживающее функции, выявленные для данной группы.

Проектирование приложения в WebML выполняется с помощью специальных концептуальных схем WebML, которые описывают структуру прикладной области и компоненты навигации на высоком уровне абстрактности, независимо от особенностей реализации. Проектирование web-приложения включает два вида работ:

- Проектирование данных: соответствует преобразованию базовых информационных объектов, выявленных в ходе анализа требований, в полную и согласованную схему данных.
- Проектирование гипертекста: создаются схемы представлений на основе ранее описанной схемы данных. Отличительной особенностью подхода WebML является упор на концептуальное моделирование спецификаций гипертекста.

Реализация web-приложений, разработанных с помощью WebML выполняется с помощью специальных инструментов (например, WebRatio CASE tool), которые в значительной степени помогают проектировщикам создавать базы данных и web-приложения. Прежде всего, они предлагает визуальную среду для рисования концептуальных схем данных и гипертекстов. Затем такие визуальные описания сохраняются в виде XML документов, которые используются в качестве входных данных для WebML генераторов кода.

Тестирование и оценка.

- Систематическое тестирование разрабатываемого приложения, благодаря доступности концептуальной модели и возможности преобразования модели для генерирования кода.
- Основное внимание перенесено с проверки конкретных web приложений к оценке правильности работы генератора кода. Если можно гарантировать, что генератор кода создает правильную реализацию для всех правильно составленных и имеющих смысл концептуальных схем (т.е., комбинаций моделирующих конструкций), то тогда тестирование web приложений сводится к более решаемой задаче проверки правильности описания концептуальных схем.

- Включает инновационные методы оценки качества разработанных web приложений. Имеется среда для управляемого моделью и автоматического оценивания качества. Данная среда поддерживает:
  - ✓ *Статический* анализ концептуальных схем (выполняется во время компиляции). Основывается на обнаружении в концептуальной схеме шаблонов проектирования и на их автоматической оценке относительно атрибутов качества записанных в виде правил.
  - ✓ *Динамический* (во время выполнения) сбор данных об использовании приложения для автоматического анализа и сравнения с навигацией заданной концептуальной схемой. Заключается в автоматической проверке и анализе усовершенствованного web-журнала, который дополняет обычный HTTP журнал, следующими данными:
    - информация об элементах и переходах, использованных пользователями;
    - информация об объектах БД опубликованных в просматриваемых страницах.

Для выполнения поддержки и развитие также использует концептуальную модель приложения:

- запросы на изменения могут быть фактически преобразованы в изменения на концептуальном уровне (в модели данных или в модели гипертекста).
- изменения на концептуальном уровне затем преобразуются в реализацию.

Методология WSDM

Данная методология первоначально (в 1998 году) была разработана для создания небольших, связанных с данными, web-приложений (информационных киосков). Постепенно она развилась до полной методологии проектирования приложений для web-сети (в том числе и семантической web сети), поддерживающей, как основные, так и широкий набор дополнительных задач проектирования (таких, как локализация, доступность, семантическое аннотирование, адаптируемость и т.п.).

WSDM является многошаговым методом проектирования web-приложений, включающий: методологию; ориентированность на аудиторию (audience-driven) и технологии подхода Semantic Web.

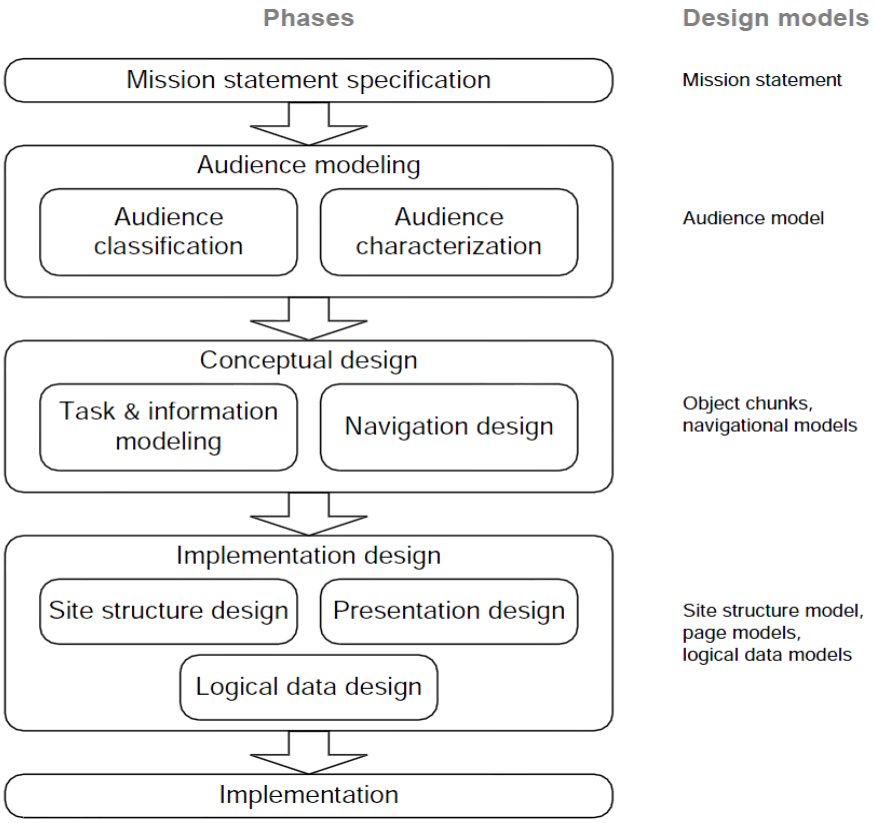


Рис. 5.3. Этапы модели разработки WSDM

Дополнительно к явно определенным принципам проектирования и моделям для описания web-приложений на разном уровне абстрактности, WSDM также предлагает проектировщикам помощь в том, как получить конкретные экземпляры настолько различающихся моделей, чтобы разработать хорошо структурированное, согласованное и удобное в использовании web-приложение. Таким образом, WSDM предлагает проектировщикам руководства и методики, тем самым предоставляя явный и систематизированный способ описания web приложений.

Методология WSDM признает важность пользователей и поэтому, в качестве начального действия использует анализ разных их типов (называемых – аудиториями), их характеристик и требований (различающихся), которые они предъявляют к разрабатываемому web-приложению. Результаты данного анализа далее используется для управления ходом проектирования.

В соответствии с методологией WSDM прежде всего требуется выполнить формальное ***описание миссии*** web-приложения. Целью описания миссии является задание границ для проектирования путем выявления цели web сайта, его тем (областей работы) и целевых пользователей. Описанная миссия используется в ходе проектирования для того, чтобы гарантировать, что вся требуемая информация и функциональ-

ность будет создана, все пользователи, для которых разрабатывается данное web приложение будут поддержаны. После завершения процесса проектирования миссия используется для проверки того, были ли достигнуты все сформулированные цели разработки web приложения. Миссия формулируется на естественном языке.

Следующим этапом методологией WSDM является ***моделирование аудитории***. В ходе этого этапа учитывается, что разные посетители web приложения могут иметь разные потребности и цели, и поэтому требуют специальной поддержки, более учитывающей их потребности. На первом подэтапе классификации аудитории все целевые пользователи, неформально описанные при определении миссии, уточняются и классифицируются на классы аудитории (пользователей). Класс аудитории это группа посетителей, которые имеют одинаковые требования к информации и функциям. Часть выявленного класса аудитории, которая имеет (одинаковые или) некоторые дополнительные требования в сравнении со всем данным классом, называется подклассом аудитории. Такое отношение частичного порядка приводит к описанию иерархической структуры, называемой иерархией классов аудитории. В ходе подэтапа описания аудиторий, для участников каждого класса аудиторий выявляются его характеристики, навигация и требования к удобству использования.

Этап ***концептуального проектирования*** делится на два подэтапа: моделирование задач и информации и моделирование навигации. На подэтапе моделирования задач и информации, проектировщик моделирует задачи, которые должны быть выполнены разными классами аудитории, совместно с используемым контентом и функциональностью. Для каждого требования, которое было сформулировано в ходе моделирования аудитории (на предыдущем шаге) описывается модель задач. Каждая из таких моделей задачи включает декомпозицию общей задачи элементарные задачи, требуемые для выполнения отдельных простых требований, а также временная взаимосвязь между ними (например, последовательная, независимая от порядка). Для выполнения такого анализа задач, WSDM использует немного модифицированную версию метода моделирования Concurrent Task Trees (CTT, согласованные деревья задач). В соответствии с данным методом, для каждой элементарной задачи создается, так называемый, объектный блок, который описывает точно, какая информация и/или функциональность требуется для выполнения данной элементарной задачи. Для формального описания таких объектных блоков WSDM использует язык OWL.

В ходе проектирования навигации выполняется моделирование (концептуальной) навигационной структуры, независимым от реализа-

ции способом. Такая навигационная структура показывает общую организацию структуры web приложения, т.е., как разные посетители могут выполнять переходы (навигацию) между страницами web-сайта.

В WSDM, базовая навигационная структура основывается на иерархии классов аудиторий: для каждого класса аудиторий создается путь навигации. Такой путь может рассматриваться в виде под-сайта, содержащего только такую информацию и функциональность, которая требуется для данного класса аудитории.

Внутренняя структура навигации в рамках навигационного пути основывается на (временных отношениях между) моделями задач. Навигационная модель состоит из трех основных моделирующих элементов: концептуальные вершины (nodes) навигации, соответствующие элементам навигации; связи между этими вершинами навигации; объектные блоки, которые связаны с вершинами навигации.

В ходе ***проектирования реализации*** концептуальные модели дополняются всеми необходимыми деталями, чтобы подготовить реальную реализацию, которая может быть генерироваться автоматически. На первом подэтапе, ***проектирование структуры сайта***, концептуальная навигационная структура отображается на реальные web страницы. Возможно наличие нескольких структур сайта, в зависимости от устройства, контекста и платформы (например, разные размеры экрана могут потребовать создание разных структур сайта). В ходе ***проектирования представления*** определяется общий вид и впечатление данного web сайта. Для разных типов страниц (например, домашней страницы) могут быть описаны шаблоны, которые в качестве основы при проектировании реальных страниц. В ходе проектирования страницы, разработчик должен:

- выбрать для каждой страницы конкретные элементы интерфейса (например, выпадающие списки или радио-кнопки для представления набора альтернатив);
- определить их расположение;
- определить информацию и функциональность объектных блоков;
- оценить общий вид и впечатление от данной страницы.

В результате таких работ получаются так называемые модели страниц. Для активно использующих данные web сайтов для хранения данных могут использоваться базы данных или системы управления контентом (CMS). В этом случае на подэтапе ***проектирования данных*** описываются реальные источники данных и соответствие между концептуальными моделями данных (т.е., объектными блоками) и такими источниками данных.

После того, как будут разработаны подходящие модели проектирования (т.е. объектные блоки, навигационные структуры, модели структуры сайта и модели страниц), источники данных и (логический) проект данных, может быть автоматически сгенерирован реальный web сайт. Известны два прототипа, реализующих процесс генерации кода: система, основанная на XSLT трансформации и сервлет на основе Java.

5.4. Общие рекомендации по разработке web-приложений

Основной целью архитектора программного обеспечения при проектировании web-приложений является максимальное упрощение их структуры путем разделение задач на функциональные области, обеспечивая при этом безопасность и высокую производительность. Данные рекомендации помогут выполнить все требования и создать условия для эффективной работы web-приложений в обычных для них сценариях:

- Выполните логическое разделение функциональности приложения. Используйте многослойную структуру для логического разделения приложения на слой представления, бизнес-слой и слой доступа к данным. Это поможет создать удобный в обслуживании код и позволит отслеживать и оптимизировать производительность каждого слоя в отдельности. Четкое логическое разделение также обеспечивает более широкие возможности масштабирования приложения.
- Используйте абстракцию для реализации слабого связывания между слоями. Этот подход можно реализовать с помощью интерфейсных типов или абстрактных базовых классов можно определить совместно используемую абстракцию, которая должна быть реализована интерфейсными компонентами.
- Определитесь с тем, как будет реализовано взаимодействие компонентов друг с другом. Для этого необходимо понимать сценарии развертывания, которые должно поддерживать приложение. Выясните, должно ли поддерживаться взаимодействие через физические границы или границы процесса, либо все компоненты будут выполняться в рамках одного процесса.
- Используйте кэширование для сокращения количества сетевых вызовов и обращений к базе данных. При проектировании web-приложения используйте такие техники, как кэширование и буферизация вывода, для сокращения сетевых вызовов между браузером и web-сервером и web-сервером и нижестоящими серверами. Правильно спроектированная стратегия кэширования, вероятно, единственный наиболее важный с точки зрения производительности аспект дизайна. ASP.Net

предоставляет следующие возможности кэширования: кэширование вывода страницы, частичное кэширование страниц и Cache API.

- Используйте протоколирование и инструментирование. Необходимо выполнять аудит и протоколирование действий в слоях и уровнях приложения. Журналы регистрации событий могут использоваться для выявления подозрительных действий, что часто обеспечивает раннее обнаружение атак на систему. Не забывайте, что могут возникнуть сложности с регистрацией проблем, возникающих в коде сценариев, выполняющихся в браузере.
- Продумайте аспекты аутентификации пользователей на границах доверия. При проектировании приложения необходимо предусмотреть аутентификацию пользователей при пересечении границ доверия, например, при доступе к удаленному бизнес-слою из слоя представления.
- Не следует передавать важные конфиденциальные данные по сети в виде открытого текста. Если требуется передавать по сети конфиденциальные данные, такие как пароль или куки аутентификации, используйте для этого шифрование и подписи данных либо шифрование с использованием протокола Secure Sockets Layer (SSL).
- Проектируйте web-приложение для выполнения с менее привилегированной учетной записью. Процесс должен иметь ограниченный доступ к файловой системе и другим ресурсам системы. Это позволит максимально сократить возможные негативные последствия на случай, если злоумышленник попытается взять процесс под свой контроль.

Существует ряд общих вопросов, на которые следует обратить внимание при проектировании. Эти вопросы можно сгруппировать по определенным областям проектирования. В следующих разделах представлены рекомендации по областям проектирования, которые чаще всего вызывают затруднения:

- Обработка запросов приложения.
- Аутентификация.
- Авторизация.
- Кэширование.
- Управление исключениями.
- Протоколирование и инструментирование.
- Навигация.
- Компоновка страницы.
- Формирование визуального отображения страницы.
- Управление сеансами.

- Проверка введенных данных (валидация).

Обработка запросов приложения

Существует два основных способа обработки запросов web-приложением. При использовании подхода с обратной отправкой браузер, преимущественно, взаимодействует с сервером посредством синхронных обращений к web-формам (Web Forms). Популярным альтернативным подходом является применение вызовов сервисов по протоколу REST между браузером и сервером. Оба подхода имеют определенные преимущества и недостатки.

При выборе стратегии обработки запросов необходимо учесть требуемую степень контроля над пользовательским интерфейсом (UI) в приложении, подход к разработке и тестированию, а также требования к производительности и масштабированию.

В подходе с обратной отправкой обычно допускается разработка с использованием форм и применяются сложные серверные элементы управления, формирующие для браузера визуальное отображение при помощи соответствующего HTML кода, связанного состояния представления и логики взаимодействия. Данный подход используется если необходимо быстро создать web-приложение на базе форм.

Независимо от выбранной стратегии обработки запросов необходимо обеспечить разделение функциональных областей путем реализации логики обработки запросов и логики приложения отдельно от UI. Это обеспечивают несколько шаблонов. Четкое разделение функциональных областей при использовании подхода с обратной отправкой Web Forms обеспечит шаблон Model-View-Presenter (MVP) или аналогичные ему шаблоны. Шаблон Model-View-Controller (MVC) обычно используется при обработке REST-запросов.

Также при проектировании стратегии обработки запросов следует руководствоваться следующими рекомендациями:

- Рассмотрите возможность централизации общих этапов пред- и пост-обработки запросов web-страницы для повышения возможности повторного использования логики на страницах. Например, разместите общую логику пред- и пост-обработки в специально предусмотренном для этого HTTP-модуле или базовом классе, наследуемом от ASP.Net-класса Page (Страница).
- Выберите соответствующий подход или шаблон для обработки UI. С помощью шаблонов MVC, MVP или аналогичных разделите обработку UI на три роли: модель, представление и контрол-

лер/презентатор. Избегайте смешения в компонентах логики обработки и формирования визуального отображения.

- При проектировании представления для обработки больших объемов данных предоставьте ему доступ к модели через использование шаблона Supervising Presenter (или Supervising Controller), который является разновидностью шаблона MVP. Если в приложении нет необходимости поддерживать состояние представления, и имеется лишь ограниченное число событий управления, используйте шаблон MVC.
- По необходимости применяйте шаблон Intercepting Filter для реализации этапов обработки в виде подключаемых фильтров.
- Обеспечьте защиту всех конфиденциальных данных, передаваемых по сети, особенно по Интернету. Используйте протоколы безопасных каналов, такие как SSL, и предусмотрите шифрование и цифровые подписи для всех особо конфиденциальных данных, передаваемых как по внутренним, так и по внешним сетям.

Аутентификация

Проектирование эффективной стратегии аутентификации имеет большое значение с точки зрения обеспечения безопасности и надежности приложения, в противном случае, оно будет уязвимым для атак с подделкой пакетов, атак перебором по словарю, перехватом сеансов и других типов атак. При проектировании стратегии аутентификации следует использовать следующие рекомендации:

- Определите границы доверия в рамках слоев web-приложения. Это поможет понять, где требуется проводить аутентификацию пользователей.
- Используйте такие способы управления учетными данными, как блокировки учетных записей и сроки действия паролей, а также строгие политики паролей, определяющие минимальную длину и сложность пароля.
- По возможности используйте поддерживаемые платформой механизмы аутентификации, такие как аутентификация Windows. Если решено применять аутентификацию с помощью форм, используйте преимущества встроенной поддержки в ASP.Net, не реализуйте собственный механизм аутентификации.
- Используйте интегрированную аутентификацию или единую регистрацию (SSO), если хотите предоставить пользователям возможность регистрироваться на нескольких сайтах с помощью одного набора учетных данных.

- Если приходится хранить пароли в базе данных, не храните их в виде открытого текста; храните хэш пароля.

Авторизация

Авторизация определяет, какие задачи может осуществлять аутентифицированный пользователь и к каким ресурсам он имеет доступ. Проектирование эффективной стратегии авторизации имеет большое значение с точки зрения обеспечения безопасности и надежности приложения, в противном случае, оно будет уязвимым для разглашения сведений, повреждения или подделки данных и несанкционированного получения прав. Глубокая защита – основной принцип безопасности стратегии авторизации приложения. При проектировании стратегии авторизации руководствуйтесь следующими рекомендациями:

- Выполняйте авторизацию пользователей при пересечении ими границ доверия. Используйте авторизацию URL для управления доступом к страницам и каталогам.
- Продумайте, насколько детализированными будут используемые настройки авторизации. Слишком детальная авторизация обусловит слишком высокие издержки на управление; однако меньшая детализация может негативно сказаться на гибкости.
- Применяйте олицетворение и делегирование, чтобы использовать преимущества аудита действий пользователя и тонкого контроля доступа, предоставляемые платформой, но не забывайте о влиянии на производительность и масштабируемость.

Кэширование

Кэширование способствует повышению производительности и сокращению времени отклика приложения. Оно позволяет оптимизировать поиск данных, сократить количество повторных обращений к сети и избежать ненужной повторной обработки. При реализации кэширования, прежде всего, необходимо принять решение о том, когда будут загружаться данные в кэш. Чтобы избежать задержек на клиенте, кэш должен загружаться асинхронно или с помощью пакетной передачи. При проектировании стратегии кэширования руководствуйтесь следующими рекомендациями:

- По возможности кэшируйте данные в готовом к использованию виде и избегайте кэширования часто меняющихся данных. Кэшируйте конфиденциальные данные только в шифрованном виде.
- Используйте кэширование вывода для кэширования относительно статических страниц. Это значительно улучшает производитель-

ность, обеспечивая при этом поддержку изменения на основании передаваемых значений. Если относительно статична только часть страницы, реализуйте частичное кэширование страницы через кэширование пользовательских элементов управления.

- Не кэшируйте, а используйте пул для совместно используемых ресурсов, создание которых требует множества ресурсов, таких как сетевые подключения.

Управление исключительными ситуациями

Проектирование эффективной стратегии управления исключениями (ошибками) имеет большое значение с точки зрения обеспечения безопасности и надежности приложения. Правильная обработка исключений в web-страницах предотвращает разглашение конфиденциальных данных об исключениях, повышает надежность приложения и обеспечивает сохранение согласованного состояния приложения в случае возникновения ошибки. При проектировании стратегии управления исключениями руководствуйтесь следующими рекомендациями:

- Обеспечьте пользователям понятные сообщения об ошибках, возникающих в приложении, но убедитесь, что не предоставляете конфиденциальные данные на страницах ошибок, в сообщениях об ошибках, файлах журналов и аудита.
- Перехватывайте необрабатываемые исключения и освобождайте ресурсы после возникновения исключения. Проектируйте общий обработчик исключений, отображающий общую страницу ошибок или сообщение об ошибке для всех необрабатываемых исключений. Избегайте использования собственных исключений без крайней необходимости.
- Перехватывайте только те исключения, которые должны обрабатывать; например, чтобы удалить конфиденциальные данные или ввести дополнительные сведения в исключение. Не используйте исключения для управления потоком выполнения приложения.

Протоколирование и инструментирование

Проектирование эффективной стратегии протоколирования и инструментирования имеет большое значение с точки зрения обеспечения безопасности и надежности приложения. Необходимо выполнять аудит и протоколирование действий всех уровней приложения. Журналы могут использоваться для выявления подозрительных действий, что часто обеспечивает раннее обнаружение атак на систему. Файлы журналов и аудита могут пригодиться для доказательства правонарушений в случае судебного разбирательства, и пользователи уже не смогут безнаказанно

отказываться от своих действий. Аудит считается наиболее достоверным, если данные журнала формируются непосредственно в момент доступа к ресурсу той же процедурой, которая выполняет этот доступ. При проектировании стратегии протоколирования и инструментирования руководствуйтесь следующими рекомендациями:

- Выполняйте аудит событий управления пользователями, критических событий системы, критических бизнес-операций и необычных действий.
- Создавайте политики безопасного управления файлами журналов, такие как ограничение доступа к файлам журналов, и предоставляйте пользователям доступ только для записи. Обеспечивайте настраиваемые в ходе разработки и производственной эксплуатации механизмы протоколирования и инструментирования.
- Не храните конфиденциальные данные в файлах журналов или аудита.

Навигация

Разделяйте стратегию навигации и логики обработки. Стратегия навигации должна позволять пользователям без труда перемещаться по экранам или страницам приложения. Обеспечьте единообразное представление навигационных ссылок и элементов управления во всем приложении, чтобы не запутывать пользователя и скрыть сложность приложения. При проектировании стратегии навигации руководствуйтесь следующими рекомендациями:

- Включите навигацию в главную страницу web-приложения (имеются в виду шаблоны в виде Master Page, а не только начальная страница сайта), чтобы обеспечить ее единообразие по всему приложению. Однако избегайте явного задания путей перехода в коде приложения. Также обеспечьте пользователям возможность перехода только к тем страницам, для просмотра которых они авторизованы.
- Предусмотрите карту сайта, она поможет пользователям находить необходимые страницы, а поисковым механизмам просматривать сайт в случае необходимости. Используйте в UI визуальные элементы, такие как встроенные ссылки, навигационные меню и отображение текущего положения пользователя в иерархии страниц web-приложения (например, ЭУ SiteMapPath), чтобы пользователи понимали, где они находятся, что предлагается на сайте и как быстро перемещаться по сайту.

Задание схемы страницы

Проектирование приложения должно обеспечивать отделение задания схемы (компоновку) страниц от конкретных компонентов пользовательского интерфейса и логики его обработки. При выборе способа задания схемы страниц необходимо учитывать, кто будет ее создавать: дизайнеры или программисты. Если этим будут заниматься дизайнеры, выбирайте такой подход к разработке схемы страниц, который не требует написания кода или использования инструментов разработки. При проектировании схемы страниц руководствуйтесь следующими рекомендациями:

- По возможности максимально используйте для компоновки каскадные таблицы стилей CSS, а не выполняйте компоновку на основе таблиц. Однако если требуется отображать сетку или если данные представлены в виде таблицы, то может использоваться и компоновка на основе таблицы.
- Постарайтесь максимально унифицировать компоновку страниц, чтобы обеспечить понятность и простоту работы с ними. Не создавайте большие страницы, выполняющие множество задач, особенно если с каждым запросом обычно выполняется лишь несколько задач. Максимально сокращайте размер страницы, чтобы обеспечить наилучшую производительность и снизить требования по пропускной способности.
- В ASP.Net-приложениях используйте мастер страницы для обеспечения общего внешнего вида и поведения для всех страниц и обновления сайта с минимальными трудозатратами. Общие разделы страниц вынесите в отдельные пользовательские элементы управления, это поможет упростить компоновку в целом и обеспечит возможность повторного использования этих элементов управления.
- Использование серверных элементов управления ASP.Net AJAX и библиотеки на стороне клиента ASP.Net AJAX упростит перенос клиентского сценария между разными браузерами. Также не смешивайте клиентский сценарий с HTML-кодом, это усложняет обслуживание страницы. Размещайте его в отдельных файлах сценария, чтобы обеспечить возможность их кэширования браузером.

Формирование визуального отображения страницы

При проектировании стратегии формирования визуального отображения страницы необходимо обеспечить эффективность этого процесса и максимальное удобство использования интерфейса. При опре-

делении формирования визуального отображения руководствуйтесь следующими рекомендациями:

- Использование сценария на стороне клиента или ASP.Net AJAX обеспечит лучшее взаимодействие с пользователем и меньшее время отклика за счет снижения числа необходимых обратных отправок. Применение собственного клиентского сценария может усложнить тестирование приложений, поскольку разные браузеры и версии реализуют разную поддержку сценариев.
- Лучше используйте ASP.Net AJAX, который поддерживает большинство популярных браузеров. Не забывайте, что использование любого кода на стороне клиента (включая сценарии, порождаемые встроенными элементами управления ASP.Net) может неблагоприятно сказываться на доступности.
- Обеспечьте соответствующую поддержку специальных возможностей для специализированных агентов (например, роботы поисковых систем) и пользователей с ограниченными возможностями.
- Рассмотрите возможности привязки данных. Например, можно выполнять привязку коллекций, объектов DataReader, таблиц DataSet и собственных объектов ко многим элементам управления ASP.Net. Используйте технологии разбиения данных на страницы для сокращения проблем с масштабируемостью, обусловленных необходимостью обработки больших объемов данных, и для улучшения производительности и времени отклика.
- Отделите компоненты пользовательского процесса от формирования визуального отображения данных и функций запроса.

Управление сеансами работы

При проектировании web-приложения важно выработать эффективный и безопасный подход к управлению сеансами работы, это будет иметь большое значение для производительности и надежности. При этом следует учесть такие аспекты, как, что нужно сохранять, где будет выполняться хранение и как долго будут храниться данные. При проектировании стратегии управления сеансами руководствуйтесь следующими рекомендациями:

- Проанализируйте, есть ли необходимость сохранять состояние сеанса. Использование состояния сеанса обусловливает дополнительные издержки при каждом запросе страницы. Сохраняйте данные сеанса, только если это действительно необходимо, для улучшения производительности используйте сеансы только для чтения или полностью отключайте состояние сеанса.

- Если используется один web-сервер, требуется обеспечить оптимальную производительность обработки состояния сеанса и число параллельных сеансов невелико, храните состояние внутри процесса. Однако если воссоздание данных сеанса чрезвычайно ресурсоемко и требуется обеспечить длительное хранение данных в случае перезапуска ASP.Net, используйте сервис состояния сеанса, выполняющийся на локальном web-сервере. Для сценариев с использованием множества серверов (web-фермы), где требуется централизованное хранение данных сеансов серверов, используйте хранилище состояния SQL Server.
- Если данные состояния хранятся на отдельном сервере, защитите канал связи для передачи состояния сеанса с помощью таких техник, как SSL или IPSec.
- Для снижения затрат на сериализацию отдавайте предпочтение базовым типам при хранении данных сеансов.

Проверка введенных данных (валидация)

Эффективный способ проверки данных вводимых пользователями (валидации) имеет большое значение с точки зрения обеспечения безопасности и надежности приложения. При несоответствующей или недостаточно полной валидации приложение может оказаться уязвимым к таким угрозам безопасности, как межсайтовые атаки внедрением сценариев (cross-site scripting, XSS), атаки типа внедрение SQL-кода, переполнение буфера и другие типы атак посредством входных данных. При проектировании стратегии валидации руководствуйтесь следующими рекомендациями:

- Проверяйте все данные, пересекающие границы доверия приложения. Исходите из предположения, что все управляемые клиентом данные являются злонамеренными и подлежат валидации.
- Стратегия валидации должна обеспечивать ограничение, отклонение и очистку злонамеренного ввода. Проводите проверку длины, диапазона, формата и типа вводимых данных.
- Используйте проверку на стороне клиента для оптимизации взаимодействия с пользователем и сокращения числа обращений к сети, но для обеспечения безопасности всегда выполняйте проверку на сервере.
- Изучите решения, шаблоны проектирования и библиотеки сторонних производителей, которые помогут реализовать централизованное управление и повторное использование правил и кода валидации.

Вопросы выбора технологий

В ASP.Net модель ASP.Net Web Forms можно сочетать с большим набором других технологий, включая ASP.Net AJAX, ASP.Net MVC и Silverlight. При этом можно использовать следующие рекомендации:

- Используйте ASP.Net для создания приложений, доступ к которым осуществляется через web-браузер, или специализированного агента пользователя.
- Используйте ASP.Net и AJAX для создания приложений, обеспечивающих повышенные возможности взаимодействия с пользователем и фоновую обработку с меньшим количеством перезагрузок страниц.
- Используйте ASP.Net и элементы управления Silverlight для создания приложений, которые включают насыщенное медиасодержимое и расширенные возможности взаимодействия с пользователем.
- При использовании ASP.Net позаботьтесь о предоставлении шаблонов страниц для реализации единообразного UI для всех страниц.

При использовании разработки через тестирование или в случае необходимости обеспечить детализированное управление UI применяйте шаблон MVC и ASP.Net MVC. Это позволит четко отделить логику приложения и навигации от UI приложения.

Список литературы

1. Комагоров В.П. Технологии сети Интернет: протоколы и сервисы Учебное пособие / Комагоров В.П. – Томск, изд-во ТПУ, 2008. – 112с.
2. Руководство компании Microsoft по проектированию архитектуры приложений (второе издание). 2009. – 560с. (электронный ресурс: http://download.microsoft.com/documents/rus/msdn/ры_приложений_полная_книга.pdf)
3. Столбовский Д.Н. Основы разработки Интернет приложений и Web сервисов на основе ASP.Net: Учебный курс / Столбовский Д.Н. – Владикавказ: Северо-Кавказский горно-металлургический институт (ГТУ), 2008. – 256с. (электронный ресурс: http://window.edu.ru/window/library?p_rid=57408)
4. Тузовский А.Ф. Высокоуровневые методы информатики и программирования. Учебное пособие / Тузовский А.Ф. – Томск, изд-во ТПУ, 2009. – 200с.
5. Эспозито Д. Microsoft ASP.Net 2.0. Базовый курс / Пер. с англ. – М.: Издательство «Русская редакция», 2007. – 688 с.
6. Berners-Lee T., Fichetti M. Weaving the Web (The original design and ultimate destiny of the World Wide Web by its inventor). – New York: HarperCollins Publisher, 1999. – 239p.
7. Casteleyn S., Daniel F., Dolog P., Matera M. Engineering Web Applications. – Berlin: Springer-Verlag, 2009. – 363p.
8. Eccher C. Professional Web Design: Techniques and Templates, Fourth Edition. – Boston, USA: Course Technology, 2011. – 927p.
9. MacDonald M., Freeman A., Szpuszta M. Pro ASP.Net 4 in C# 2010. – New-York: Apress, 2010. – 1617p.
10. Shklar L., Rosen R. Web Application Architecture: Principles, Protocols and Practices, 2nd Edition. – Atrium, England: John Wiley & Sons, 2009. – 440p.

Учебное издание

ТУЗОВСКИЙ Анатолий Фёдорович

ПРОЕКТИРОВАНИЕ ИНТЕРНЕТ ПРИЛОЖЕНИЙ

Учебно-методическое пособие

Издано в авторской редакции

Научный редактор *доктор технических наук*,
профессор А.Ф. Тузовский
Дизайн обложки И.О. Фамилия

Отпечатано в Издательстве ТПУ в полном соответствии с качеством представленного оригинал-макета

Подписано к печати ??.??.2010. Формат 60x84/16. Бумага «Снегурочка».
Печать XEROX. Усл.печ.л. 11,63. Уч.-изд.л. 10,53.
Заказ xxxxx Тираж xxx 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.
Тел./факс 8(3822)56-35-35, www.tpu.ru