**Тема 1 Взаимодействие клиентов с Web-сервером. Технологии стороны клиента.**
*(Понятие о всемирной паутине (WWW). Протокол HTTP. Язык разметки HTML. Сценарии, выполняемые на стороне клиента.)*

**План:**

1. Понятие о WWW.
2. Взаимодействие клиентских программ с Web-сервером.
3. Краткое описание протокола HTTP.
4. Краткое описание языка разметки HTML.
5. HTML - формы
6. Динамический HTML. Сценарии на стороне клиента. JavaScript. VBScript. (ASP (?!))

## 1. Понятие о WWW

Одним из крупнейших достижений Internet стала "всемирная паутина" – WWW (World Wide Web или просто Web). WWW представляет собой множество независимых, но взаимосвязанных серверов. Работая с Web, пользователь "перемещается" между серверами, то есть последовательно соединяется с ними и получает информацию, как правило, в виде *гипертекста*. В современном Internet WWW играет настолько важную роль, что именно ее часто имеют в виду, говоря об Internet, что, вообще говоря, неверно.

Согласно REC-html40-971218 – стандарту языка HTML 4.0 (RFC – Resource for Comments, так называются основные документы консорциума W3, специфицирующие технологии Internet), Web – это сеть информационных ресурсов, в которой для доступности этих ресурсов наиболее широкой аудитории используется три механизма:

1. Единая схема именования ресурсов для поиска последних в Web – URI.
2. Протокол для доступа к ресурсам через Web – HTTP.
3. Гипертекст для перемещения по ресурсам – HTML.

Под Web-технологиями будем понимать всю совокупность средств для организации WWW. Поскольку в каждом сеансе взаимодействуют две стороны – сервер и клиент, Web-технологии разделяются на две группы – технологии стороны сервера (server-side) и технологии стороны клиента (client-side).

Более подробную информацию о рассматриваемых здесь вопросах модно получить в пособии О.И. Стрельникова “Основы Web технологий”, сокращенный материал которого и представлен в данных лекциях, а также в других источниках, в том числе – в оригинальных документах W3 консорциума и иных стандартах :

RFC 1866 "Hypertext Markup Language – HTML 2.0"

RFC 1945 "Hypertext Transfer Protocol – HTTP/1.0"

RFC 2616 "Hypertext Transfer Protocol – HTTP/1.1"

REC-html4-971218 "Hypertext Markup Language – HTML 4.0"

ISO 8879 "Standard Generalized Markup Language – SGML".

## 2. Взаимодействие клиентов с Web сервером

Взаимодействие между клиентом и сервером осуществляется по протоколу **HTTP** - протокол передачи гипертекста. То есть в своей основе протокол обмена между клиентами и сервером Web является текстовым (несмотря на то, что по нему возможна передача и бинарных данных). Это делает его достаточно легким для понимания, программной поддержки, отладки программ, а также делает его удобным для межплатформенного взаимодействия, то есть для совместной работы клиентов и серверов, реализованных на разных платформах. (Такое взаимодействие можно реализовать и в бинарном протоколе, однако текстовый протокол более прозрачен). Протокол HTTP является одним из протоколов прикладного уровня в стеке протоколов TCP/IP и при этом одним из самых востребованных. Уже давно как весь стек протоколов TCP/IP, так и протокол HTTP используется не только в сети Internet, но и в локальных и корпоративных Intranet сетях, осуществляя взаимодействие клиентов и серверов в распределенных корпоративных приложениях, которые при необходимости легко переносятся и в Интернет.

Рассмотрим чуть подробнее протокол HTTP.

## 3. Протокол HTTP

Протокол передачи гипертекста **HTTP (Hypertext Transfer Protocol, RFC 1945, 2068)** предназначен для передачи гипертекстовых документов от сервера к клиенту. Протокол HTTP относится к протоколам прикладного уровня. На практике в подавляющем большинстве случаев транспортным протоколом для HTTP является протокол TCP, причем сервер HTTP (сервер Web) находится в состоянии ожидания соединения со стороны клиента стандартно по порту 80 TCP, а клиент HTTP (браузер Web) является инициатором соединения.

В терминах Web все, к чему может получить доступ пользователь, – документы, изображения, программы, – называется ресурсами. Каждый ресурс имеет уникальный для Web адрес, называемый универсальным идентификатором ресурса (URI – Universal Resource Identifier). В самом общем случае URI выглядит следующим образом:

**protocol://user:password@host:port/path/file?parameters#fragment**

Отдельные поля URI имеют следующий смысл:

protocol - прикладной протокол, посредством которого получают доступ к ресурсу;

user - пользователь, от имени которого получают доступ к ресурсу либо сам пользователь в качестве ресурса;

password - пароль пользователя для аутентификации при доступе к ресурсу;

host - IP-адрес или имя сервера, на котором расположен ресурс;

port - номер порта, на котором работает сервер, предоставляющий доступ к ресурсу;

path - путь к файлу, содержащему ресурс;

file - файл, содержащий ресурс;

parameters - параметры для обработки ресурсом-программой;

fragment - точка в файле, начиная с которой следует отображать ресурс.

Взаимодействие между клиентом и сервером Web осуществляется путем **обмена сообщениями. Сообщения HTTP** делятся на **запросы клиента серверу** и **ответы сервера клиенту**.

Сообщения запроса и ответа имеют общий формат. Оба типа сообщений выглядят следующим образом: сначала идет начальная строка (start-line), затем, возможно, одно или несколько полей заголовка, называемых, также, просто заголовками, затем пустая строка (то есть строка, состоящая из символов CR и LF), указывающая конец полей заголовка, а затем, возможно, тело сообщения:

начальная строка

поле заголовка 1

поле заголовка 2

...

поле заголовка N

CR LF

тело сообщения

Формат начальной строки клиента и сервера различаются и будут рассмотрены далее. **Заголовки** бывают четырех видов:

общие заголовки (general-headers), могут присутствовать как в запросе, так и в ответе;

заголовки запросов (request-headers), могут присутствовать только в запросе;

заголовки ответов (response-headers), могут присутствовать только в ответе;

заголовки объекта (entity-headers), относятся к телу сообщения и описывают его содержимое.

Каждый заголовок состоит из названия, символа двоеточия ":" и значения. Некоторые важные заголовки приведены в табл. 1.

Таблица 1

Примеры заголовков протокола HTTP

| Заголовок | Назначение |
|---|---|
| Заголовки объекта | |
| Content-Encoding | Способ, которым закодировано тело сообщения, например, с целью уменьшения размера |
| Content-Length | Длина сообщения в байтах |
| Content-Type | Тип содержимого и, возможно, некоторые параметры |

| Заголовки ответа | |
|---|---|
| Location | URI ресурса, к которому нужно обратиться для получения содержимого |
| Retry-After | Дата и время или число секунд, через которое нужно повторить запрос, чтобы получить успешный ответ |
| Server | Название программного обеспечения сервера, приславшего ответ |
| Заголовки запроса | |
| Accept | Типы содержимого, которое "понимает" клиент и может воспроизвести |
| Accept-Encoding | Способ, которым сервер может закодировать сообщение |
| If-Modified-Since, If-Match,If-Range... | Заголовки запроса для условного обращения к ресурсу |
| Range | Запрос части документа |
| User-Agent | Название программного обеспечения клиента |
| Общие заголовки | |
| Connection | Указывает серверу на завершение (close) или продолжение (keep-alive) сеанса |
| Transfer-Encoding | Способ кодирования сообщения при передаче |

Подробное описание заголовков HTTP/1.0 можно найти в RFC 2068.

В теле сообщения содержится собственно передаваемая информация – полезная нагрузка сообщения. Тело сообщения представляет собой последовательность байтов. Тело сообщения может быть закодировано, например, для уменьшения объема передаваемой информации, при этом способ кодирования указывается в заголовке объекта Content-Encoding.

**Сообщение запроса от клиента к серверу** состоит из строки запроса (request-line), заголовков (общих, запросов, объекта) и, возможно, тела сообщения.

**Строка запроса** начинается с метода, затем следует идентификатор запрашиваемого ресурса, версия протокола и завершающие символы конца строки:

**<Метод> <Идентификатор> <Версия HTTP> <CR><LF>**

Метод указывает команду протокола HTTP, которую нужно применить к запрашиваемому ресурсу. Идентификатор определяет запрашиваемый ресурс. Версия HTTP обозначается строкой следующего вида:

HTTP/<версия>.<подверсия> (В RFC 2068 представлен протокол HTTP/1.1.)

Рассмотрим некоторые **методы протокола HTTP**.

OPTIONS

Метод OPTIONS выполняет запрос информации об опциях соединения (например, методах, типах документов, кодировках), которые поддерживает сервер для запрашиваемого ресурса. Этот метод позволяет клиенту определять опции и/или требования, связанные с ресурсом, или возможности сервера, не производя никаких действий над ресурсом и не инициируя его загрузку.

GET

Метод GET позволяет получать любую информацию, связанную с запрашиваемым ресурсом. В большинстве случаев, если идентификатор запрашиваемого ресурса указывает на документ (например, документ HTML, текстовый документ, графическое изображение, видеоролик), то сервер возвращает содержимое этого документа. Если запрашиваемый ресурс является приложением, формирующим в процессе своей работы некоторые данные, то в теле сообщения ответа возвращаются эти данные. Это используется, например, при создании приложений CGI. Если идентификатор запрашиваемого ресурса указывает на директорию (каталог, папку), то, в зависимости от настроек сервера, может быть возвращено либо содержимое директории (список файлов), либо содержимое одного из файлов, находящегося в этой директории (как правило, index.html или Default.htm).

Разновидностями метода GET является "условный GET" ("conditional GET"), при котором сообщение запроса включает заголовки запроса If-Modified-Since, If-Unmodified-Since, If-Match, If-None-Match, или If-Range. Условный метод GET запрашивает передачу объекта, только если он удовлетворяет условиям, описанным в приведенных заголовках. Различают также "частичный GET" ("partial GET"), который запрашивает передачу только части объекта с помощью заголовка Range. Сообщение ответа в случае запроса с частичным методом GET должно содержать заголовок Content-Range, в котором указывается передаваемый диапазон.

HEAD

Метод HEAD идентичен GET, за исключением того, что сервер не возвращает в ответе тело сообщения. Этот метод может использоваться для получения информации об объекте запроса без непосредственной пересылки тела объекта, например с целью тестирования связей и пр.

POST

Метод POST используется для запроса, при котором адресуемый сервер принимает данные, включенные в тело сообщения запроса, и отправляет их на обработку приложению, указанному как запрашиваемый ресурс. Наряду с методом GET метод POST используется при создании серверных приложений. Отличие от GET состоит в способе передачи данных запроса (в GET параметры передаются в идентификаторе запрашиваемого ресурса, а в POST – в теле сообщения запроса), а также в том, что GET может вернуть содержимое ресурса, а POST всегда запускает приложение.

PUT

Тело сообщения, которое передается в запросе с методом PUT, сохраняется на сервере,

причем идентификатор запрашиваемого ресурса будет идентификатором сохраненного документа. Если идентификатор запрашиваемого ресурса указывает на уже существующий ресурс, то включенный в тело сообщения объект заменяет существующий.

Различие между методами POST и PUT заключается в различном назначении идентификатора ресурса. URI в запросе POST идентифицирует ресурс, который обрабатывает включенный в тело сообщения объект (приложение). Напротив, URI в запросе PUT назначается для создаваемого ресурса (объекта, включенного в запрос в виде тела сообщения).

DELETE

Метод DELETE запрашивает сервер об удалении ресурса, имеющего запрашиваемый идентификатор.

TRACE

Метод TRACE используется для возврата переданного запроса на уровне протокола HTTP. Получатель запроса (сервер Web) отправляет полученное сообщение обратно клиенту как тело сообщения ответа с кодом состояния 200 (OK).

Подробную информацию о методах протокола HTTP/1.1 можно найти в RFC 2068.

После получения и интерпретации сообщения запроса сервер отвечает **сообщением HTTP ответа**.

Первая строка ответа – это **строка состояния (Status-Line)**. Она состоит из версии протокола, числового кода состояния, поясняющей фразы, разделенных пробелами и завершающих символов конца строки:

**<Версия HTTP> <Код состояния> <Поясняющая фраза><CR><LF>**

Версия протокола имеет тот же смысл, что и в запросе.

Элемент **код состояния (Status-Code)** – это целочисленный трехразрядный (трехзначный) код результата понимания и удовлетворения запроса. Поясняющая фраза (Reason-Phrase) представляет собой короткое текстовое описание кода состояния. Код состояния предназначен для обработки программным обеспечением, а поясняющая фраза предназначена для пользователей.

Первая цифра кода состояния определяет класс ответа. Последние две цифры не имеют определенной роли в классификации. Имеется 5 значений первой цифры:

1xx: Информационные коды – запрос получен, продолжается обработка.

2xx: Успешные коды – действие было успешно получено, понято и обработано.

3xx: Коды перенаправления – для выполнения запроса должны быть предприняты дальнейшие действия.

4xx: Коды ошибок клиента – запрос имеет ошибку синтаксиса или не может быть

выполнен.

5xx: Коды ошибок сервера – сервер не в состоянии выполнить допустимый запрос.

Поясняющие фразы для каждого кода состояния перечислены в RFC 2068 и являются рекомендуемыми, но могут быть заменены на эквивалентные без ограничений со стороны протокола. Например, в локализованных русскоязычных версиях HTTP серверов эти фразы заменены русскими. В табл. 2 приведены коды ответов сервера HTTP.

Таблица 2 Некоторые коды ответов сервера HTTP

| Код | Поясняющая фраза согласно RFC 2068 | Эквивалентная поясняющая фраза на русском языке |
|---|---|---|
| 1xx: Информационные коды | | |
| 100 | Continue | Продолжать |
| 2xx: Успешные коды | | |
| 200 | OK | OK |
| 201 | Created | Создан |
| 204 | No Content | Нет содержимого |
| 4xx: Коды ошибок клиента | | |
| 400 | Bad Request | Испорченный запрос |
| 401 | Unauthorized | Несанкционированно |
| 404 | Not Found | Не найден |
| 405 | Method Not Allowed | Метод не дозволен |
| 408 | Request Timeout | Истекло время ожидания запроса |
| 5xx: Коды ошибок сервера | | |
| 500 | Internal Server Error | Внутренняя ошибка сервера |
| 503 | Service Unavailable | Сервис недоступен |

Подробную информацию о кодах ответа и заголовках, сопровождающих данные ответы, можно получить в RFC 2068.

За строкой состояния следуют заголовки (общие, ответа и объекта) и, возможно, тело сообщения.

Одной из важнейших функций сервера Web является предоставление доступа к части локальной файловой системы. Для этого в настройках сервера указывается некоторая директория, которая является корневой для данного сервера Web. Чтобы опубликовать документ, то есть сделать его доступным пользователям, "посещающим" данный сервер нужно скопировать этот документ в корневую директорию Web-сервера или в одну из ее поддиректорий. При соединении по протоколу HTTP на сервере создается процесс с правами пользователя, специально созданного для просмотра ресурсов сервера. Настраивая права и разрешения данного пользователя, можно управлять доступом к ресурсам Web.

## 4. Язык разметки HTML

Язык **HTML** (HyperText Markup Language - язык разметки гипертекста) является основным средством представления информации в виде гипертекста в Web. Важнейшим понятием языка HTML является понятие **элемента**. Под элементами понимаются структуры,

из которых строится документ на языке HTML. Некоторые элементы могут включать в себя другие. Каждый элемент в документе HTML может включать три части: **начальный тэг**, **содержимое и конечный тэг**. Любая из трех частей может отсутствовать. Содержимое может быть другим элементом либо обычным текстом. Для некоторых элементов конечный тэг может быть запрещен. Начальный тэг представляет собой название элемента, заключенное в угловые скобки. В конечном тэге внутри угловых скобок названию элемента предшествует символ "/". Важно различать понятия элемента и тэга: некоторые элементы присутствуют в документе HTML даже при отсутствии в этом документе соответствующих тэгов. Сведения об основных элементах языка HTML 4.0 приведены в табл. 3.

Таблица 3 Основные элементы языка HTML

| Элемент | Начальный тэг | Конечный тэг | Описание |
|---|---|---|---|
| Общая структура документа | | | |
| HTML | Не обязателен | Не обязателен | Корневой элемент документа |
| HEAD | Не обязателен | Не обязателен | Заголовок документа |
| BODY | Не обязателен | Не обязателен | Тело документа |
| FRAMESET | Обязателен | Обязателен | Разделение окна |
| FRAME | Обязателен | Запрещен | Вложенное окно |
| Заголовочные элементы | | | |
| TITLE | Обязателен | Обязателен | Название документа |
| BASE | Обязателен | Запрещен | Базовый URI документа |
| Блоковые элементы | | | |
| H1–H6 | Обязателен | Обязателен | Заголовок |
| P | Обязателен | Не обязателен | Абзац |
| DIV | Обязателен | Обязателен | Общий контейнер языка/стиля |
| CENTER | Обязателен | Обязателен | Сокращение для  |
| HR | Обязателен | Запрещен | Горизонтальный разделитель |
| TABLE | Обязателен | Обязателен | Таблица |
| FORM | Обязателен | Обязателен | Интерактивная форма |
| Шрифтовое выделение | | | |
| I | Обязателен | Обязателен | Курсив |
| B | Обязателен | Обязателен | Полужирный текст |
| U | Обязателен | Обязателен | Подчеркнутый текст |
| STRIKE, S | Обязателен | Обязателен | Перечеркнутый текст |
| BIG (SMALL) | Обязателен | Обязателен | Большой текст (Мелкий текст) |
| SUB (SUP) | Обязателен | Обязателен | Нижний индекс (Верхний индекс) |
| Специальные элементы | | | |
| A | Обязателен | Обязателен | Ссылка |
| IMG | Обязателен | Запрещен | Внедренное изображение |
| APPLET | Обязателен | Обязателен | Апплет Java |
| FONT | Обязателен | Обязателен | Локальное изменение шрифта |
| BR | Обязателен | Запрещен | Жесткий перевод строки |
| SCRIPT | Обязателен | Обязателен | Выражения скрипта |
| MAP | Обязателен | Обязателен | Клиентское изображение-карта |
| IFRAME | Обязателен | Обязателен | Встроенное окно |
| OBJECT | Обязателен | Обязателен | Общий внедренный объект |
| Органы управления формы | | | |
| INPUT | Обязателен | Запрещен | Управляющий элемент формы |
| SELECT | Обязателен | Обязателен | Выбор варианта |

| | | | |
|---|---|---|---|
| TEXTAREA | Обязателен | Обязателен | Текстовое поле из нескольких строк |
| OPTION | Обязателен | Не обязателен | Выбираемый элемент |
| BUTTON | Обязателен | Обязателен | Кнопка |
| LABEL | Обязателен | Обязателен | Текст метки поля формы |
| OPTGROUP | Обязателен | Обязателен | Группа опций |
| FIELDSET | Обязателен | Обязателен | Группа управляющих элементов формы |
| Содержимое таблиц | | | |
| CAPTION | Обязателен | Обязателен | Заголовок таблицы |
| TH | Обязателен | Не обязателен | Заголовок ячейки таблицы |
| TR | Обязателен | Не обязателен | Строка таблицы |
| TD | Обязателен | Не обязателен | Ячейка данных таблицы |
| COL | Обязателен | Запрещен | Столбец таблицы |

Подробное описание элементов можно найти в стандарте REC-html40.

Документ HTML всегда содержит элемент HTML, в свою очередь, содержащий элементы HEAD и BODY. Разделенный на фреймы документ, вместо элемента BODY, содержит элемент FRAMESET. Остальные элементы можно разделить на следующие группы:

заголовочные элементы;

блоковые элементы;

текстовые элементы.

Заголовочные элементы могут содержаться только внутри элемента HEAD. Содержимым блоковых элементов могут быть другие блоковые элементы, текстовые элементы и обычный текст. Каждый блоковый элемент начинает новый абзац. Содержимым текстовых элементов могут быть только другие текстовые элементы и текст.

С элементом могут быть связаны некоторые **свойства**, называемые **атрибутами**. Значения атрибутов задаются при объявлении элемента или присваиваются по умолчанию. Атрибуты элемента указываются в начальном тэге. Значение атрибута заключается в двойные кавычки, между атрибутом и его значением ставится символ "=". Если атрибуту присваивается значение, совпадающее с названием атрибута, то это значение с символом "=" можно опустить. Некоторые основные атрибуты элементов приведены в табл. 4.

Таблица 4

Некоторые атрибуты элементов

| Атрибут | Элементы | Значение |
|---|---|---|
| background | BODY | URI фонового изображения |
| bgcolor | BODY | Цвет фона |
| alink | BODY | Цвет выбранных ссылок |
| vlink | BODY | Цвет просмотренных ссылок |
| link | BODY | Цвет ссылок |
| text | BODY | Цвет текста |
| cols | FRAMESET | Список колонок в наборе фреймов |
| rows | FRAMESET | Список строк в наборе фреймов |
| align | P, H1–H6, DIV | Выравнивание текста (left, right, center, justify) |

| | | |
|---|---|---|
| href | A, AREA, LINK, BASE | URI, на который указывает элемент |
| coords | A, AREA | Список координат |
| name | A, APPLET, FRAME, IFRAME, OBJECT, MAP, PARAM, органы управления | Имя объекта |
| shape | A, AREA | Тип поверхности (rect, circle, poly, default) |
| target | A, AREA, BASE, FORM, LINK | Имя фрейма |
| align | IMG, APPLET, IFRAME, INPUT, OBJECT | Горизонтальное и/или вертикальное выравнивание (top, middle, bottom, left, right) |
| border | IMG, OBJECT | Толщина границы (размер) |
| height | IMG, IFRAME, OBJECT, APPLET | Высота |
| hspace | IMG, IMG, OBJECT | Горизонтальный отступ |
| vspace | IMG, APPLET, OBJECT | Вертикальный отступ |
| width | IMG, IFRAME, OBJECT, APPLET, HR, PRE | Ширина |
| frameborder | FRAME, IFRAME | Границы фреймов (0, 1) |
| scrolling | FRAME, IFRAME | Полоса прокрутки (yes, no, auto) |
| code | APPLET | Класс аплета |
| codebase | APPLET | Базовый URI |
| object | APPLET | Потоковый (сериализованный) класс, содержащий объект |
| data | OBJECT | URI объекта |
| action | FORM | URI обработчика формы |
| method | FORM | Метод доступа к обрабатывающему ресурсу сервера (GET, POST) |
| disabled | Органы управления | Не активен (disabled) |
| checked | INPUT | Отмеченный элемент (checked) |
| cols | TEXTAREA | Число столбцов |
| for | LABEL | Идентификатор связанного управляющего элемента |
| maxlength | INPUT | Максимальное число вводимых символов |
| readonly | TEXTAREA, INPUT | Не изменяемый (readonly) |
| selecting | OPTION | Выбран (selected) |
| rows | TEXTAREA | Число строк |
| multiple | SELECT | Разрешен множественный выбор (multiple) |
| type | INPUT | Тип ввода (text, password, checkbox, radio, submit, reset, file, hidden, image, button) |
| type | BUTTON | Тип кнопки (submit, reset, button) |
| size | HR, FONT, BASEFONT, INPUT, SELECT | Размер |
| value | OPTION, PARAM, INPUT, BUTTON, LI | Значение |
| language | SCRIPT | Язык скрипта |
| defer | SCRIPT | Отложить выполнение (defer) |
| clear | BR | Управление разбиением текста (left, all, right, none) |
| color | FONT | Цвет текста |
| nohref | AREA | Нет ссылки (nohref) |
| noshade | HR | Нет тени (noshade) |
| valuetype | PARAM | Тип параметра (data, ref, object) |

Подробное описание каждого атрибута можно найти в стандарте REC-html40.

Средством разработки и описания языков разметки, подобных HTML, является язык SGML (Standard Generalized Markup Language – обобщенный язык разметки). Каждая конструкция данного языка заключается между специальными скобками: "<!" и ">". В языке HTML могут присутствовать конструкции языка SGML, которые браузер должен пропускать. Для указания версии языка HTML, на котором написан документ, следует поместить в первую строку этого документа **конструкцию DOCTYPE языка SGML** с указанием типа документа и URI реализации версии HTML. Для HTML версии 4.0 эта строка выглядит следующим образом:

<!DOCTYPE HTML PUBLIC "–//W3C//DTD HTML 4.0//EN" "http://www.w3.org/TR/REC-html40/strict.dtd">

Внутри конструкций языка SGML можно помещать **комментарии**, заключенные между символами "--". Данные комментарии можно использовать и в языке HTML, поскольку последний пропускает конструкции SGML. Для HTML комментарии будут выглядеть следующим образом:

<!-- Текст комментария (возможно многострочный) -->

Подробную информацию о конструкциях языка SGML можно найти в ISO 8879.

Начиная с версии 4.0, в HTML поддерживаются **фреймы**. Фреймы позволяют отображать в одном окне навигатора несколько независимых документов. Будем считать фреймом рамку вокруг содержимого документа – окна. Окно может быть разбито на несколько частей. При этом каждая часть будет иметь свою собственную рамку – фрейм, имеющий собственное содержимое – окно. Таким образом, в любом фреймовом документе имеется иерархия фреймов и окон.

### 5. Формы HTML

**Форма** – это часть документа HTML, содержащая органы управления, информация о состоянии которых может быть передана пользователем серверу Web. Для размещения формы внутри документа HTML используется элемент FORM:

```
<FORM action=uri method=m enctype=mime>...</FORM>
```

Атрибут action задает идентификатор программы, которой будут отправлены данные формы для обработки. Чаще всего это программа на стороне сервера.

Атрибут method задает метод, применяемый к ресурсу. Может принимать значения GET и POST. Атрибут enctype задает способ кодирования содержимого формы (значений в строках ввода, состояний переключателей). Если данные из формы будут переданы серверу Web в теле сообщения запроса, то способ кодирования означает тип содержимого и

передается в заголовке объекта Content-Type. По умолчанию применяется единственный возможный для метода GET способ кодирования application/x-www-form-urlencoded. Для метода POST применим также способ text/plain.

Внутри формы, помимо других элементов, могут находиться **элементы органов управления**. Элементы органов управления могут располагаться и вне элементов FORM. В этом случае данные из органов управления для обработки передаются отображаемому в настоящий момент документу.

Отправка формы может осуществляться при нажатии кнопки класса submit, при нажатии клавиши ENTER в строке ввода, при нажатии мышью внутри органа управления класса image. При отправке формы клиент производит следующие действия:

1. Кодирование содержимого элементов. У каждого органа управления (элемента) формы имеются атрибуты name и value, обозначающие соответственно имя элемента и значение. Имя задается в документе HTML. Значение зависит от типа элемента. Кодирование заключается в замене некоторых недопустимых символов последовательностями допустимых. Пробелы в значениях кодируются символом '+'. Символы с кодом больше или равным 127, а также некоторые символы с кодом меньше 127, например, '+', '%' и '&', кодируются тройкой символов '%hh', где hh – две шестнадцатеричные цифры кода символа. Код символа зависит от кодировки, в которой создан содержащий форму документ. Таким образом, для каждого элемента получается строка вида name=value, где name – имя элемента органа управления, value – строка с закодированным значением элемента органа управления.
2. Кодирование содержимого формы в соответствии с атрибутом enctype элемента FORM. В случае способа кодирования application/x-www-form-urlencoded полученные строки для каждого элемента соединяются друг с другом символом '&':

```
name1=value1&name2=value2&...&nameN=valueN
```

В случае способа кодирования text/plain строки разделяются символами перевода строки и возврата каретки:

```
name1=value1
name2=value2
...
nameN=valueN
```

3. Формирование HTTP-сообщения запроса. В случае метода GET кодированное содержимое формы присоединяется к идентификатору запрашиваемого ресурса (программы CGI) в начальной строке запроса при помощи символа '?':

```
URI?name1=value1&name2=value2&...&nameN=valueN
```

В случае метода POST кодированное содержимое формы помещается в тело

сообщения. В сообщение запроса включается заголовок Content-Type, в котором содержится способ кодирования содержимого, и заголовок Content-Length, указывающий на длину кодированного содержимого.

4. Отправка сообщения серверу.

### 6. Сценарии на стороне клиента

Для обеспечения алгоритмической обработки содержимого документа и выполнения иных действий на стороне клиента в язык HTML введена поддержка **сценариев** – программ, загружаемых с сервера вместе с документом HTML и выполняемых браузером при просмотре этого документа. Существует несколько языков, на которых может быть написан сценарий. Наиболее распространен язык **JavaScript** компании Netscape, первой реализовавшей поддержку сценариев стороны клиента в своем браузере (Не тождествен языку Java !!!).

JavaScript, как и другие языки для встраивания сценариев в HTML-документы, имеет программный интерфейс для доступа к браузеру. Браузер, как правило, имеет панель инструментов, строку ввода адреса, рабочую область и строку состояния. Панель инструментов, как минимум, содержит кнопки "Back", "Forward", "Stop", "Reload" и "Home". Используя язык JavaScript, можно управлять содержимым рабочей области и строки состояния, а также открывать новые окна браузера, возможно, без панели инструментов и строки ввода адреса. Кроме того, можно вызывать простейшие диалоговые панели с кнопками и полями ввода.

Другое назначение языков сценариев – обработка различных **событий**, инициируемых элементами документа (прежде всего – органами управления и пр.) - технология **Dynamic HTML (DHTML)**. Реакции на события описываются в соответствующих атрибутах элементов (см. табл. 6).

Для доступа к компонентам в языке JavaScript используются объекты. В языке JavaScript между объектами не устанавливается отношения наследования. Таким образом, JavaScript не является объектно-ориентированным языком.

В языке JavaScript имеется возможность создавать собственные новые классы, однако данная возможность используется редко из-за малой гибкости (по сравнению с объектно-ориентированными языками). Можно определить JavaScript как язык для доступа к некоторой существующей объектной библиотеке. С точки зрения лексики и синтаксиса, язык JavaScript похож на языки C++ и Java.

Основные классы объектной библиотеки JavaScript:

Таблица 5 Классы JavaScript

| Свойства и методы | Описание |
|---|---|

| `string`: Строка (с информацией о выводе на экран) | |
|---|---|
| `number length` | Длина строки |
| `string big()` | Большой шрифт |
| `string small()` | Маленький шрифт |
| `string bold(), ...` | Полужирный шрифт |
| `string fontsize(number size)` | Установить размер символов |
| `string fontcolor(string color)` | Установить цвет символов |
| `string link(string href)` | Превращает строку в гиперссылку href |
| `string charAt(number i)` | i-й символ строки |
| `string substring(number begin, number end)` и т.д. ... | Подстрока |
| `window`: Окно | |
| `string name` | Название окна |
| `string status` | Строка состояния |
| `number length` | Число фреймов в окне |
| `frame[] frames` | Массив фреймов |
| `window self/window` | Ссылка на себя |
| `window parent` | Родительское окно |
| `window top` | Ссылка на верхнее окно в иерархии |
| `document document` | Документ |
| `frame frame` | Фрейм |
| `location location` | Местоположение отображаемого содержимого |
| `window open(string URL, string Name, string features)` | Открыть новое окно. Параметр features представляет собой список параметров и значений, перечисленных через запятую. Название параметра и значение разделяются символом "=". Возможны следующие параметры: toolbar, location, directories, status, menubar, scrollbars, resizeable, width, height. Все параметры, кроме width и height принимают логические значения (yes, no, 1 или 0). Параметры width и height принимают целочисленные значения. |
| `close()` | Закрыть окно |
| `timeout setTimeout(string expr, number msec)` | Установить таймаут |
| `clearTimeout(timeout t)` | Удалить таймаут |
| `alert(string message)` | Диалоговая панель с сообщением message и кнопкой Ok |
| `bool confirm(string message)` | Диалоговая панель с сообщением message и кнопками Ok и Cancel |
| `string prompt(string message, string default)` | Диалоговая панель с сообщением и полем ввода |
| `frame`: Фрейм | |
| `string name` | Название |
| `number length` | Число внутренних фреймов |
| `frame[] frames` | Набор фреймов внутри данного |
| `frame self/window` | Ссылка на себя |
| `frame parent` | Ссылка на родительский фрейм |
| `window parent` | Ссылка на родительское окно |
| `timeout setTimeout(string expr, number msec)` и `clearTimeout...` | Установить таймаут |
| `location`: Идентификатор ресурса | |
| `string hash` | Поле fragment в URI |
| `string host` | Поля host и port в URI |
| ... | |
| `string search` | Параметры в URI для программы CGI |
| `document`: Документ (содержимое окна) | |
| `string title` | Заголовок |
| `string URL` | URI |

| | |
|---|---|
| `Date lastModified` | Дата и время последней модификации документа |
| `string fgColor, ...` | Цвет переднего плана |
| `link link` | Ссылка |
| `history history` | Набор ссылок |
| `form form` | Форма |
| `link[] links` | Массив ссылок |
| `form[] forms` | Массив форм |
| `string referrer` | URI документа, из которого вызван текущий |
| `open([string mime])` | Начать вывод содержимого документа |
| `close()` | Завершить вывод содержимого документа |
| `write(...)` | Вывести в качестве содержимого документа |
| `writeln(...)` | Вывести в качестве содержимого документа и перевести строку |
| `history`: Набор ссылок | |
| `number length` | Число ссылок в памяти |
| `back()` | Вернуться на предыдущую страницу |
| `forward()` | Загрузить следующую страницу |
| `go(number delta/string location)` | Перейти |
| `static Math`: Математические константы и функции | |
| `number E` | E |
| `number LN2 ...` | ln(2) |
| `number abs(number a), ...` | Абсолютное значение |
| `Date`: Дата и время | |
| `Date()` | Инициализация объекта текущей датой и временем |
| `number getDate()` | Число |
| `number getDay()` | День недели |
| `number getHours(), ...` | Часы |
| `setDate(number Date)` | Установка даты |
| `setHours(number Hours), ...` | Установка часов |
| `Array`: Массив | |
| `Array()` | Пустой массив |
| `Array(number size)` | Массив с size элементами |
| `Array(a0, a1, ...)` | Массив с элементами, инициализированными a0, a1, ... |
| `number length` | Число элементов массива |
| `string join()` | Объединение элементов массива в строку |
| `reverse()` | Изменение порядка элементов массива на обратный |
| `sort()` | Сортировка элементов в массиве |
| `Image`: Изображение | |
| `Image()` | Пустое изображение |
| `Image(number w, number h)` | Пустое изображение с заданными размерами |
| `string src` | URI ресурса, содержащего изображение |

В языке JavaScript имеется возможность определять собственные **функции**. Определение функции в JavaScript имеет следующий вид:

function <идентификатор>(<параметры>) { ... }

Идентификатор функции – это имя, по которому функция будет вызываться в сценариях JavaScript. Параметры указываются без типов и разделяются символом ",". Пользовательские функции в JavaScript не могут возвращать значения, в отличие от встроенных функций.

Для размещения сценариев JavaScript внутри документов HTML используется элемент SCRIPT:

<SCRIPT language=l src=s defer></SCRIPT>

Атрибут language указывает язык, на котором написан сценарий. Для сценариев на языке JavaScript этот атрибут должен принимать значение "JavaScript". В атрибуте src указывается URI файла со сценарием. Атрибут defer указывается, если сценарий не осуществляет вывод на экран при помощи методов write и writeln объекта document для ускорения формирования содержимого страницы.

Как уже упоминалось, языки сценариев могут использоваться для обработки **событий.** В табл. 6 перечислены основные события Dynamic HTML :

Таблица 6 События Dynamic HTML

| Атрибут | Элемент | Событие |
|---|---|---|
| onfocus | A, AREA, BUTTON, INPUT, LABEL, SELECT, TEXTAREA | Попадание в фокус |
| onblur | | Потеря фокуса |
| onchange | INPUT, SELECT, TEXTAREA | Изменение |
| onselect | INPUT, TEXTAREA | Выбор элемента списка |
| onclick | Кроме APPLET, BASE, BASEFONT, BDO, BR, FONT, FRAME, FRAMESET, HEAD, HTML, IFRAME, ISINDEX, META, PARAM, SCRIPT, STYLE, TITLE | Щелчок |
| ondblclick | | Двойной щелчок |
| onkeydown | | Клавиша нажата |
| onkeypress | | Клавиша нажата и отпущена |
| onkeyup | | Клавиша отпущена |
| onmousedown | | Кнопка мыши нажата |
| onmousemove | | Мышь перемещена |
| onmouseout | | Курсор мыши покинул область элемента |
| onmouseover | | Курсор мыши перемещен в область элемента |
| onmouseup | | Кнопка мыши отпущена |
| onload | BODY, FRAMESET | Документ загружен |
| onunload | | Документ выгружен |
| onreset | FORM | Форма сброшена |
| onsubmit | | Форма отправляется |

7. Дополнительно :

7.1 Примеры : ?!... - см. ниже + пример из лабы ...

7.2 Другие языки сценариев ... (VBScript, asp?!!!)

Таблица 7

| ***Пример с JavaScript*** | ***Пример с VBScript*** |
|---|---|
| <HTML><HEAD><TITLE>A Simple Page With JavaScript</TITLE><br><SCRIPT LANGUAGE="JavaScript"><br><!--<br>function OnClick() {<br>window.alert("Hello !");<br>}<br>--><br></SCRIPT></HEAD><br><BODY><br><H3>A Simple JavaScript</H3><HR><br><FORM><br><INPUT NAME="Button1" TYPE="BUTTON" VALUE="Click Here" onclick="OnClick)"><br></FORM><br></BODY></HTML> | <HTML><HEAD><TITLE>A Simple Page With VBScript</TITLE><br><SCRIPT LANGUAGE="VBScript"><br><!--<br>Sub Button1_OnClick<br>MsgBox "Hello !"<br>End Sub<br>--><br></SCRIPT></HEAD><br><BODY><br><H3>A Simple VBScript</H3><HR><br><FORM><INPUT NAME="Button1" TYPE="BUTTON" VALUE="Click Here"></FORM><br></BODY></HTML> |

Отличия VBScript :

## Тема 2 Технологии создания корпоративных Интернет-решений (Наталия Елманова, Компьютер Пресс, 2'2005)

Когда-то, на заре развития Интернета, создатели Всемирной сети представляли ее себе как общее пространство для обмена информацией. Изначально Интернет существовал как документоориентированная сеть, а первыми веб-сайтами были примитивные файловые серверы, которые возвращали статические HTML-страницы запросившим их клиентам. Подобные сайты и сегодня по-прежнему не редкость — именно таким способом выполнены небольшие персональные веб-страницы, а также сайты некрупных компаний, не претендующих ни на что, кроме собственно факта присутствия в Интернете, где обычно размещен относительно небольшой объем не слишком часто меняющейся информации.

В процессе развития из набора информационных ресурсов Интернет постепенно превратился в инструмент, способствующий повышению эффективности деятельности компаний, а затем и в одно из основных средств ведения бизнеса. Аналогичным образом развивались и технологии создания корпоративных веб-сайтов — постепенно в их числе появились средства реализации интерактивности, персонализации информационного наполнения, взаимодействия с клиентами, а также инструменты для осуществления интеграции с корпоративными информационными системами и средствами управления предприятиями. Возникли и специализированные средства для создания инфраструктуры корпоративных веб-приложений, внедрение которых в общем случае не требует программирования (о них можно прочесть в статье «Инфраструктура корпоративных Интернет-решений» в данном спецвыпуске). Однако в основе подавляющего большинства как средств для создания инфраструктуры корпоративных веб-приложений, так и индивидуальных заказных решений лежит относительно небольшое количество технологий разработки веб-приложений, о которых мы и собираемся поговорить в этой статье.

### Клиентские технологии

Технологии создания веб-приложений условно можно разделить на клиентские (то есть используемые веб-браузерами и другими веб-клиентами, например офисными приложениями или клиентами средств мгновенной передачи сообщений) и серверные (то есть использующиеся на веб-серверах).

Клиентские технологии применяются главным образом для повышения интерактивности приложений, например для проверки корректности вводимых данных без дополнительного обращения к серверу, и для создания удобного пользовательского интерфейса. Так, современные веб-браузеры и некоторые почтовые клиенты способны интерпретировать код на скриптовых языках, выполнять Java-аплеты и элементы управления ActiveX, использовать другие дополнения, такие как Macromedia Flash Player, средства просмотра презентаций QuickTime, средства воспроизведения мультимедиаданных.

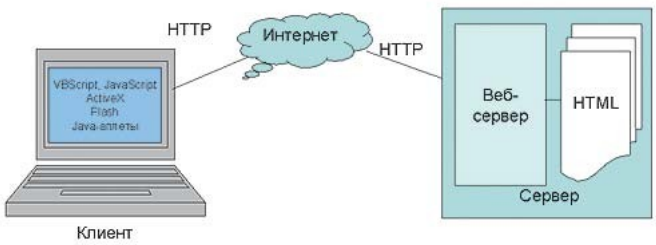


Рис. 1

**Код, интерпретируемый браузером**

Большинство современных веб-браузеров, созданных для различных платформ и устройств, способно интерпретировать внедренный в HTML-страницу код на скриптовых языках, таких как VBScript и JavaScript. Типичные примеры применения внедренного клиентского кода — проверка корректности введенных пользователем данных без обращения к веб-серверу, создание некоторых элементов дизайна — наподобие всплывающих кнопок и меню, а также управление другими объектами, внедренными в HTML-страницу.

Код, интерпретируемый браузером, выполняется в его собственном адресном пространстве. Такой код обладает довольно ограниченным набором средств (в частности, он не может обращаться к файловой системе компьютера-клиента или запускать на нем другие приложения). Тем не менее большинство браузеров позволяет запретить исполнение кода на скриптовых языках, поскольку любое выполнение кода на компьютере-клиенте может оказаться небезопасным (например, из-за каких-либо ошибок реализации интерпретатора этого кода в браузере).

**Java-аплеты**

Практически все современные браузеры способны отображать и выполнять Java-аплеты — специальные Java-приложения, ссылка на которые внедряется в веб-страницу. Аплеты могут выполняться на всех платформах, для которых существуют виртуальные Java-машины. Способы взаимодействия аплетов с компьютером-клиентом также ограничены — например им недоступны его файловая система и приложения, а сетевой доступ из аплета возможен только к тому компьютеру, с которого он был загружен. Однако аплетом можно управлять, задавая в тексте содержащей его HTML-страницы или в коде на скриптовых языках той же страницы его параметры (например, цвет, шрифт и т.д.).

Большинство браузеров обладает доступными пользователю средствами ограничения возможностей выполнения аплетов, поскольку они, как и интерпретаторы скриптовых языков, реализуют выполнение кода на компьютере-клиенте и никто не может дать стопроцентной гарантии отсутствия ошибок в реализации Java-машины, выполняющей аплет.

**Элементы управления ActiveX**

Некоторые из современных браузеров (в частности, Microsoft Internet Explorer) могут служить контейнерами для элементов управления ActiveX — специальных COM-серверов, выполняющихся в адресном пространстве браузера. Ссылки на такие элементы управления могут содержаться в составе веб-страницы. Сами элементы управления ActiveX представляют собой динамически загружаемые библиотеки, выполняющиеся в адресном пространстве браузера.

С помощью элементов управления ActiveX, как и посредством Java-аплетов, можно реализовать любую функциональность; при этом, в отличие от Java-аплетов, выполнение элементов управления ActiveX в общем случае не подвергается никаким ограничениям на доступ к файлам и иным ресурсам операционной системы и сети, а код, содержащийся в них, выполняется от имени загрузившего их пользователя. Как и Java-аплеты, элементы управления ActiveX могут считывать свои свойства с содержащей их страницы; кроме того, свойства элемента управления ActiveX можно менять динамически из кода на скриптовых языках, содержащихся в составе той же страницы; в том же коде можно обрабатывать события, возникающие в таких элементах управления (данная технология носит название ActiveX scripting).

Естественно, браузеры, поддерживающие выполнение элементов управления ActiveX, содержат средства ограничения их функциональности — от запрета управления ими из кода на скриптовых языках до полного запрещения их загрузки из Интернета. К тому же внутрь элемента управления ActiveX можно поместить электронную цифровую подпись, и если после добавления этой подписи файл с элементом управления ActiveX будет изменен, то

перед запуском такого элемента управления об этом будет сообщено пользователю. Однако наличие электронной подписи не гарантирует отсутствия потенциально опасного содержимого — в лучшем случае оно позволяет установить его источник.

Отметим, что сегодня элементы управления ActiveX применяются главным образом в интрасетях, а не на общедоступных веб-сайтах.

### Приложения Macromedia Flash

Еще одна очень популярная веб-технология, основанная на выполнении кода в клиентском приложении, — приложения Macromedia Flash. Macromedia Flash Player, как и виртуальная Java-машина, обладает ограниченными возможностями с точки зрения доступа к ресурсам клиентского компьютера. Так, приложения Flash не имеют доступа к файловой системе, за исключением служебного каталога Macromedia Flash Player, а доступ к внешним устройствам ограничивается микрофонами и видеокамерами. Доступ к сетевым ресурсам ограничивается доменом, с которого было получено данное приложение. Отметим, что, так же как и Java-аплеты и элементы управления ActiveX, приложения Flash могут управляться с помощью кода JavaScript, присутствующего на той же странице. Поскольку Macromedia Flash Player для Microsoft Internet Explorer сам по себе является элементом управления ActiveX, он использует некоторые возможности элементов управления ActiveX для доступа к свойствам приложений Flash из скриптовых языков.

Имеется и ряд других средств, реализованных обычно в виде так называемых модулей расширения (plug-in), представляющих собой исполняемый код. При этом современные браузеры обладают средствами ограничения возможностей, связанных с их загрузкой и выполнением.

Все перечисленные средства расширения функциональности HTML-страниц могут быть использованы и в динамических страницах, генерируемых серверными веб-приложениями, — подобные страницы могут содержать ссылки на элементы управления ActiveX, приложения Flash, аплеты. Но наиболее широкое распространение получили средства создания веб-приложений, которые выполняются под управлением веб-серверов и генерируют динамические HTML-страницы с внедренным в них кодом на скриптовых языках, предназначенным для интерпретации браузером.

## Серверные технологии

Элементарные требования безопасности требуют, чтобы возможности, связанные с выполнением кода в веб-клиентах, могли быть существенно ограничены как минимум посредством средств администрирования клиентских приложений. В этом случае любое более или менее серьезное приложение, основанное исключительно на применении клиентских технологий, в известной степени обречено на провал в силу того, что условия, необходимые для его функционирования, могут не соответствовать требованиям корпоративной информационной безопасности, принятым в той или иной компании. В этом одна из причин массового развития и широкого применения технологий, связанных с выполнением кода приложений на самих веб-серверах.

### Скрипты и исполняемые файлы

Одной из первых технологий создания веб-приложений, выполняющихся на серверах, была Common Gateway Interface (CGI). Она позволяла создавать и выполнять серверные приложения, обращение к которым происходит посредством указания их имени (а иногда — и параметров) в URL. Входной информацией для таких приложений служит содержимое HTTP-заголовка либо тело запроса, в зависимости от применяемого протокола. CGI-приложения — это консольные приложения, которые генерируют HTML-код, передаваемый браузеру. Подобные приложения могут представлять собой код на скриптовых языках, интерпретируемый на сервере, либо исполняемый файл, который можно создать с помощью

практически любого средства разработки, генерирующего консольные приложения для операционной системы, под управлением которой функционирует веб-сервер.

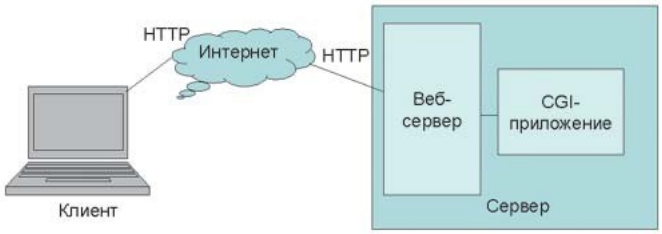


Рис. 2 Веб-приложения, основанные на применении технологии CGI

Основная проблема всех CGI-приложений заключается в том, что при каждом клиентском запросе сервер загружает это приложение с жесткого диска в отдельный процесс, а затем инициирует его выполнение и выгрузку. Эта особенность ограничивает производительность приложений и возможность одновременной обработки большого количества клиентских запросов.

**Библиотеки, загружаемые в адресное пространство веб-сервера**

Проблему ограниченной производительности веб-приложений, которые выполняются в отдельном адресном пространстве, можно решить, создав приложение в виде библиотеки, загружающейся в адресное пространство веб-сервера и при необходимости остающейся там для обработки последующих запросов от других клиентов (понятно, что в этом случае веб-сервер должен поддерживать загрузку таких библиотек). Подобные приложения для Microsoft Internet Information Servise носят название ISAPI (Internet Server Application Program Interface), а такие библиотеки для весьма популярного веб-сервера Apache называются Apache DSO (Dynamic Shared Objects). Означенные технологии существуют уже довольно продолжительное время, но все еще весьма популярны.

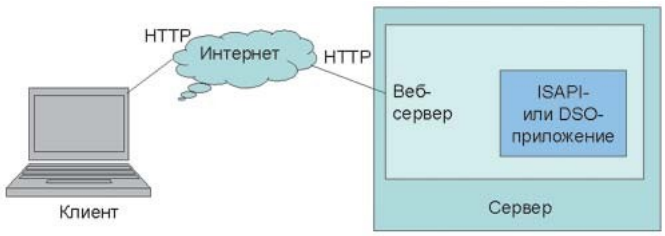


Рис. 3 Веб-приложения, основанные на применении библиотек, загружаемых в адресное пространство веб-сервера

Заметим, что при создании как CGI-, так и ISAPI-приложений довольно сложно отделить задачи дизайна от задач, связанных с реализацией функциональности и логики приложений, — подобные приложения генерируют веб-страницы целиком.

**Веб-страницы с фрагментами серверного кода**

**ASP и ASP .NET**

Очередным шагом в развитии технологий создания Интернет-приложений стало появление средств, позволяющих отделить задачи веб-дизайна от задач, связанных с реализацией функциональности приложений. Первой подобной технологией стала Active

Server Pages (ASP). Основная идея ASP заключается в создании веб-страниц с внедренными в них фрагментами кода на скриптовых языках. Однако, в отличие от рассмотренных выше средств применения скриптовых языков для расширения функциональности браузеров, указанные фрагменты кода интерпретируются не браузером, а предназначенной для этого ISAPI-библиотекой, входящей в состав Internet Information Server. Внедренный фрагмент кода замещается результатом его выполнения, а полученная таким образом динамическая страница передается в пользовательский браузер.

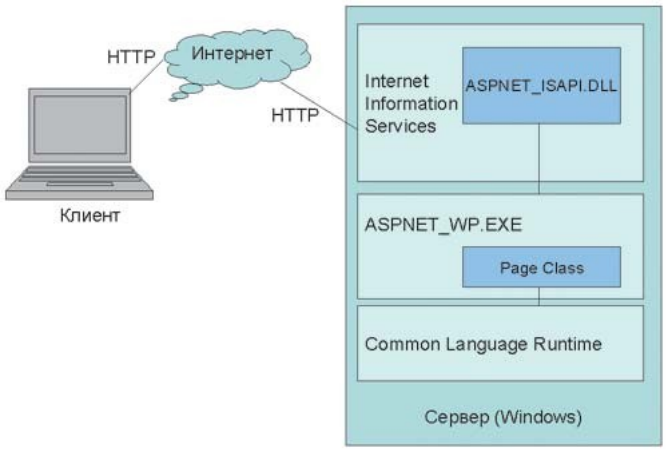


Рис. 4 Веб-приложения, основанные на применении ASP .NET

Одной из наиболее популярных сегодня технологий, реализующих идею создания веб-страниц с фрагментами кода, является ASP .NET — ключевая в архитектуре Microsoft .NET Framework. Основное отличие этой технологии от ASP в плане архитектуры приложений заключается в том, что код, присутствующий на веб-странице, не интерпретируется, а компилируется и кэшируется, что способствует повышению производительности приложений. Кроме того, указанная технология позволяет создавать так называемые серверные компоненты, возвращающие в браузер HTML-код с интерпретируемыми браузером фрагментами кода на скриптовых языках и способные предоставить более удобный пользовательский интерфейс, нежели обычный HTML-код. Важными особенностями серверных компонентов ASP .NET являются возможность обработки на сервере событий, возникающих в клиентском приложении, и возможность генерировать HTML-, WML- и CHTML-код в зависимости от типа клиента и поддерживаемых им языков разметки и протоколов передачи данных.

Технология ASP .NET 2.0, представляющая собой дальнейшее развитие технологии ASP .NET, станет доступной разработчикам в течение этого года. С ее помощью разработчики получат доступ к расширенному набору готовых блоков, шаблонов страниц, прикладных интерфейсов, средств упрощения настройки внешнего вида приложений, а также к средствам персонализации, к инструментам для поддержки локализации приложений, к утилитам развертывания, позволяющим распространять приложения без предоставления исходного кода, и к компонентам для создания порталов, доступа к защищенной информации, для удобного отображения данных, что позволит создавать веб-приложения, близкие в плане удобства и пользовательского интерфейса к Windows-приложениям.

**Java Server Pages**

Наряду с ASP и ASP .NET существуют и другие технологии, реализующие идею размещения внутри веб-страницы кода, выполняемого веб-сервером. Наиболее известной из них сегодня является технология JSP (Java Server Pages), основная идея которой — однократная компиляция Java-кода (сервлета) при первом обращении к нему, выполнение методов этого сервлета и помещение результатов выполнения этих методов в набор данных, отправляемых в браузер.

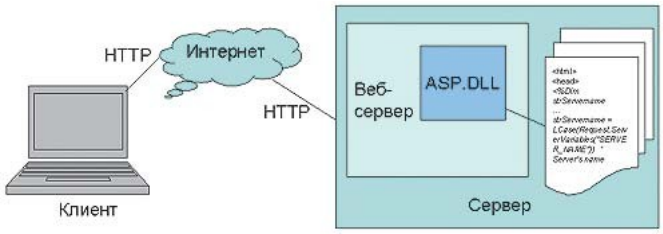


Рис. 5 Веб-приложения, основанные на применении веб-страниц с внедренными в них фрагментами серверного кода

Говоря о технологии JSP, нельзя не отметить относительно новую спецификацию Sun под названием Java Server Faces. Эта спецификация описывает правила создания веб-приложений с удобным пользовательским интерфейсом (схожим по функциональности с интерфейсом Windows-приложений) и разработки серверных компонентов, реализующих подобный интерфейс. Средства разработки Java-приложений, поддерживающие указанную спецификацию, в идеале должны позволить создавать веб-приложения, основанные на J2EE, примерно с той же скоростью и степенью удобства, что и средства разработки .NET-приложений.

Из других популярных технологий, реализующих создание веб-страниц с фрагментами кода, выполняемого на сервере, отметим PHP (Personal Home Pages). Данная технология основана на применении CGI-приложений, интерпретирующих внедренный в HTML-страницу код на скриптовом языке. Несмотря на наличие недостатков, присущих всем CGI-приложениям, PHP пользуется немалой популярностью благодаря простоте разработки и доступности для различных платформ, особенно при создании приложений, не отличающихся высокими требованиями к масштабируемости и надежности.

**Применение инфраструктурного ПО общего назначения**

Чем выше посещаемость сайта и больше объем обрабатываемых им данных, тем более строгими будут требования к производительности и масштабируемости веб-приложений. Чем более серьезные задачи решаются с помощью веб-приложения, тем выше требования к его надежности и безопасности. Чаще всего для удовлетворения этих требований бизнес-логика, реализованная в веб-приложении, а также службы обработки данных и реализации транзакций отделяются от пользовательского интерфейса приложений и реализуются в виде отдельных приложений, библиотек, сервлетов, которые в общем случае называются бизнес-объектами. Сегодня в подавляющем большинстве современных корпоративных решений применяются либо серверы, поддерживающие спецификацию Java2 Enterprise Edition, либо серверы, базирующиеся на применении служб серверных версий Windows, технологиях COM и Microsoft .NET.

Бизнес-объекты могут обладать различной функциональностью. Как правило, они предоставляют доступ к данным, управляемым какой-либо серверной СУБД, нередко — доступ к данным корпоративных информационных систем. Довольно часто бизнес-объекты реализуют какую-либо часть корпоративной информационной системы, создание которой изначально предполагало включение внешнего веб-сервера в качестве составной части корпоративной информационной системы (например, в качестве одного из источников данных для CRM-приложения). Готовые CRM- и ERP-системы ведущих производителей, таких как SAP, PeopleSoft, Siebel, обычно содержат в своем составе подобные бизнес-объекты, а зачастую — и обращающиеся к ним готовые веб-приложения, например порталы для клиентов и удаленных пользователей, приложения для осуществления электронной коммерции и иные приложения.

**Веб-службы**

Информационным системам многих предприятий уже не один десяток лет. А поскольку предприятия нередко начинали свое развитие со стихийной автоматизации отдельных видов деятельности, то сегодня многие из них сталкиваются с проблемой интеграции приложений, создававшихся в разные годы для разных платформ. Одним из средств подобной интеграции является технология веб-служб, использующая для обмена данными стандартный протокол HTTP. Создавать веб-службы можно и в виде исполняемых файлов, и в виде библиотек, и в виде интерпретируемого кода; существуют также средства представления бизнес-объектов, основанных на различных технологиях, в виде веб-служб (эту технологию сегодня поддерживают все ведущие производители офисных продуктов), средств разработки, СУБД, серверов приложений и операционных систем. Методы веб-служб можно вызывать из обычных приложений, веб-приложений, других веб-служб. В последнее время наблюдается массовое появление приложений, использующих веб-службы, в том числе предназначенных для конечных пользователей (к таким приложениям, например, относятся приложения семейства Microsoft Office System, позволяющие при помощи веб-служб пользоваться данными словарей, энциклопедий, систем онлайнового перевода, служб онлайнового заказа товаров).

**Web-службы вчера, сегодня, завтра**

(Алексей Федоров, Компьютер-Пресс, 2' 2005)

За сравнительно недолгую историю своего существования Web-службы прошли путь от относительно простого механизма межплатформенного удаленного вызова процедур по протоколу HTTP до ядра новой архитектуры создания приложений — архитектуры, ориентированной на службы (Services Oriented Architecture, SOA).

Начав со стандартов, которые теперь принято относить к стандартам Web-служб первого поколения — SOAP (Simple Object Access Protocol), WSDL (Web Service Description Language), UDDI (Universal Description, Discovery and Integration Protocol), Web-службы пополнились и собственной архитектурой (Web Services Architecture, WSA) и обширным набором дополнительных стандартов и протоколов для решения таких задач, как гарантированная передача сообщений, обеспечение адресации, безопасность, передача вложенных данных и т.п. (краткое описание существующих сегодня стандартов см. в разделе «Основные стандарты Web-служб»).

Примитивные средства создания Web-служб превратились в мощные программные продукты, позволяющие визуально создавать и потреблять Web-службы, локально или удаленно управлять ими, выполнять мониторинг их активности и производительности.

Вопросы, связанные с обеспечением полной совместимости Web-служб, реализованных на различных платформах, выделились в особую категорию. Появилась даже специальная организация — Web Services-Interoperability Organization (WS-I), отвечающая за создание профилей, описывающих принципы совместимых Web-служб. В настоящее время уже разработан базовый профиль (WS-I Basic Profile) для SOAP 1.1 и WSDL 1.0.

Кратко рассмотрим базовые стандарты и технологии, используемые для создания Web-служб, и ознакомимся с основными стандартами Web-служб, разработанными в настоящее время.

**Основы Web-служб**

В основе Web-служб лежат несколько простых принципов. Возможные для вызова команды описываются на языке WSDL; непосредственная активизация команд происходит в виде посылки SOAP-сообщений по адресу, где располагается Web-служба (используется стандартный протокол HTTP); для поиска Web-служб существуют глобальные или локальные (внутренние) каталоги, поддерживающие стандартные службы обнаружения

UDDI. Не вдаваясь в технические подробности, можно отметить, что все современные средства разработки ведущих производителей поддерживают создание Web-служб, а программные платформы (будь то серверные операционные системы или серверы приложений) обеспечивают выполнение Web-служб.

В Web-службах везде используется язык XML. Он служит, в частности, для описания сообщений, которыми могут обмениваться Web-службы и их потребители. SOAP-сообщение — это XML-документ, состоящий из трех базовых элементов: <Envelope>, <Header> и <Body>. Язык WSDL базируется на языке XML и позволяет создавать XML-документы, описывающие методы Web-служб, параметры методов, способы их вызова и т.п. Для того чтобы воспользоваться специализированными Web-службами в рамках механизмов обнаружения UDDI, следует составить SOAP-сообщения и интерпретировать возвращаемые XML-документы.

Жизненный цикл Web-службы можно условно разделить на три фазы: первая — программирование и публикация, вторая — поиск в каталоге, третья — потребление из клиентского приложения (рис. 1).


Рис. 1. Жизненный цикл Web-службы

Поддержка на уровне средств разработки и программных платформ, относительная простота создания и использования Web-служб и практически повсеместная доступность Интернета привели к тому, что за короткий срок появилось множество Web-служб, позволяющих получать различные данные — от прогнозов погоды и транспортных расписаний рейсов до курсов валют, котировок акций и даже гороскопов.

Таблица 1 Примеры Веб-служб

| Назначение Web-службы | Адрес |
|---|---|
| Web-службы для добавления в базу данных заявок на перевозку груза и предложений транспорта. В виде Web-служб также реализован доступ к транспортным новостям сайта, тематика которых охватывает все виды перевозок и сопутствующие области — таможню, страхование и т.д. | http://www.perevozki.ru/ |
| Web-службы для получения оперативной информации об актуальном расписании, прилете/вылете самолетов, состоянии рейсов. Сервис «Справка о рейсе/Табло аэропортов» предоставляет следующий набор функций:<br>- AirportList — список аэропортов;<br>- AirportInfo — информация об аэропорте по его коду;<br>- DateList — список дат, по которым имеется информация;<br>- Arrival/Departure — информация о прибытии/отправлении рейсов;<br>- FlightSearch — поиск рейса по номеру;<br>- FlightInfo — подробная информация о текущем состоянии рейса (по ключу, полученному в результате вызова Arrival/Departure/FlightSearch).<br>Сервис «Расписание рейсов» предоставляет следующий набор функций:<br>- AirportAlphabetList — список букв, на которые начинаются названия аэропортов;<br>- AirportList — список аэропортов на указанную букву;<br>- AirportInfo — информация об аэропорте;<br>- Search — поиск рейсов;<br>- Calendar — информация о других датах, в которые летает указанный рейс | http://webservices.aeroflot.ru/ |
| Использование Web-служб для извлечения информации из поисковой системы Google позволяет разработчикам выполнять запросы более чем к 8 млрд. Web-страниц. (Справедливости ради отметим, что Yandex предоставляет похожую услугу, но базирующуюся на XML-запросах, посылаемых методом POST или методом GET (протокол HTTP) без использования SOAP-запроса.) | http://www.google.com/apis/ |
| Создание набора Web-служб — Amazon Web Services (AWS) — превратило онлайновый магазин Amazon в полноценную платформу для электронной коммерции. В настоящее время в рамках Amazon Web Services доступны следующие службы:<br>- Amazon E-Commerce Service (ECS);<br>- Amazon Simple Queue Service;<br>- Alexa Web InformationService | http://www.amazon.com/ |

Некоторые примеры Web-служб представлены в табл. 1.

Следует отметить, что, возможно, самой важной особенностью Web-служб является их независимость от платформы. Это означает, что Web-служба и ее потребитель могут быть реализованы практически на любой программной платформе, причем как сама служба, так и ее потребители могут быть реализованы на разных платформах — минимальным требованием к потребителям является поддержка протокола HTTP и возможность программной обработки XML-документов.

По мере возрастания интереса к Web-службам появлялось все больше реализаций Web-служб для решения корпоративных задач; прежде всего Web-службы применялись для интеграции приложений, для обеспечения создания функциональных модулей в гетерогенных средах, для реализации уровней абстракции бизнес-логики от клиентских приложений и т.п.

Применение Web-служб в качестве основной технологии для создания корпоративных приложений привело к тому, что базовые протоколы и стандарты, удовлетворявшие требованиям разработчиков публичных Web-служб, оказались не слишком пригодными для решения задач, стоящих перед корпоративными программистами. В результате появилась масса дополнений и расширений базовых протоколов, ориентированных на растущие потребности к повсеместному использованию Web-служб.

**Основные стандарты Web-служб**

В табл. 2 собрана информация об основных стандартах, связанных с Web-службами. Стандарты и протоколы сгруппированы по базовым функциональным группам — передача сообщений, безопасность, надежность, транзакции, метаданные и бизнес-процессы. Получить полное описание вышеприведенных спецификаций можно на сайте Microsoft по адресу http://msdn.microsoft.com/webservices/understanding/specs/default.aspx.


Рис. 2. Распределение стандартов Web-служб по уровням

Отметим, что различные компании по-разному обеспечивают поддержку стандартов в своих продуктах, средствах разработки и программных платформах. Например, компания Microsoft выпускает бесплатные программные компоненты под названием Web Services Enhancements, которые доступны для загрузки с Web-сайта компании, расположенного по адресу http://msdn.microsoft.com/webservices/downloads/default.aspx.

Таблица 2. Распределение стандартов Web-служб по уровням

| | |
|---|---|
| **Передача сообщений (Messaging)** | |
| SOAP | Протокол для обмена структурированной информацией в распределенных средах. Стандарт опубликован W3C |
| WS-Addressing | Описывает механизмы адресации Web-служб, нейтральные к транспортному протоколу. Разработан компаниями IBM, BEA и Microsoft |
| WS-Eventing | Описывает протокол, по которому одна Web-служба может проявить интерес к получению сообщений о возникающих в другой Web-службе событиях. Разработан компаниями BEA, Tibco и Microsoft |
| **Безопасность (Security)** | |
| WS-Security | Описывает расширения протокола SOAP для обеспечения целостности и конфиденциальности сообщений. Является частью стандарта OASIS Web Services Security 1.0 |
| WS-Trust | Расширяет стандарт WS-Security, добавляя к нему возможность запроса и получения security tokens. |
| WS-SecureConversion | Описывает язык Web Services Secure Conversation Language, который может использоваться поверх протоколов WS-Security и WS-Trust для обеспечения безопасных коммуникаций между службами. Разработан компаниями IBM, RSA, Verisign и Microsoft |
| WS-Federation | Язык Web Services Federation Language используется для обеспечения механизмов передачи аутентификационных и авторизационных данных между различными доверительными доменами. Разработан компаниями IBM, BEA, RSA, Verisign и Microsoft |
| **Надежность (Reliability)** | |
| WS-ReliableMessaging | Описывает протокол, позволяющий осуществлять надежную доставку сообщений между распределенными приложениями. Разработан компаниями IBM, BEA, TIBCO Software и Microsoft (альтернативный протокол был предложен компаниями Sun Microsystems, Oracle, Fujitsu, NEC и Hitachi под названием WS-Reliabiliy) |
| **Транзакции (Transactions)** | |
| WS-Coordination | Описывает протокол для координации действий распределенных приложений. Разработан компаниями IBM, BEA и Microsoft |
| WS-AtomicTransaction | Предоставляет определение координации атомарных транзакций в рамках протокола WS-Coordination. Разработан компаниями IBM, BEA и Microsoft |
| WS-BusinessActivity | Предоставляет определение координации бизнес-активностей в рамках протокола WS-Coordination. Разработан компаниями IBM, BEA и Microsoft |
| **Метаданные (Meta Data)** | |
| WSDL | Язык описания Web-служб, основанный на XML. Содержит абстрактное описание операций и сообщений для вызова этих операций, а также их конкретное описание для различных сетевых протоколов и форматов сообщений. Опубликован W3C |
| UDDI | Спецификация описывает основанные на SOAP Web-службы для программного обнаружения Web-служб. Опубликована OASIS |
| WS-Policy | Задает основанный на языке XML синтаксис для описания политик Web-служб: требований, возможностей и т.п. Разработан компаниями IBM, BEA, SAP и Microsoft. Расширения — WS-PolicyAttachment, WS-PolicyAssertions, WS-SecurityPolicy и др. |
| WS-SecurityPolicy | Расширяет спецификацию WS-Security, объясняя, каким образом описывать требования к безопасности Web-служб на уровне политик |
| WS-MetadataExchange | Определяет три пары сообщений, которые могут использоваться для получения трех основных типов метаданных: для получения политик уровня WS-Policy, для получения описания службы на языке WSDL и для получения XML-схемы для указанного пространства имен. Разработан компаниями IBM, BEA, SAP и Microsoft |
| WS-Discovery | Описывает протокол для обнаружения Web-служб. Расширяет функциональность UDDI за счет поддержки устройств и систем, которые не всегда подключены к сети. Разработан компаниями Microsoft, BEA, Canon и Intel |
| **Бизнес-процессы (Business Process)** | |
| BPEL4WS | BPEL4WS (Business Process Execution Language for Web Services) — это язык для описания того, как отдельные Web-службы могут быть объединены для создания комплексных, надежных бизнес-приложений. Служит для описания бизнес-процессов, способных потреблять и предоставлять Web-службы. Данная спецификация является дальнейшим развитием языков Microsoft XLANG и IBM WSFL. Разработана компаниями Microsoft, IBM, BEA при участии SAP и Siebel. Дальнейшая работа над стандартом проводится в рамках комитета OASIS WSBPEL Technical Commitee |

## Вместо заключения

Web-службы все активнее проникают в область, уже занятую объектно-ориентированными и компонентными технологиями, представляя собой альтернативу традиционным подходам к созданию корпоративных приложений. В ближайшие годы можно ожидать серьезных инвестиций в технологии, связанные с Web-службами, развертываемыми и потребляемыми внутри компаний (согласно исследованиям IDC, к 2008 году объем рынка Web-служб превысит 11 млрд. долл.).

К тому же следует ожидать дальнейшего развития отечественных Web-служб (как бесплатных, так и коммерческих) и появления комплекса других служб, связанных с данной технологией. Возможно, это произойдет в рамках проекта «Электронная Россия», а возможно — по инициативе некоторых крупных поставщиков услуг. Впрочем, каким образом это будет происходить, не так уж и важно — важно то, что Web-службы уже пришли в Россию, и технологии их создания и потребления готовы к использованию.