Лекция 1.
Принципы гипертекстовой разметки. Структура гипертекстовых документов. Идентификаторы UDI. Коды языков.

Отображение содержимого WEB-страницы осуществляется на компьютере пользователя с помощью специальной программы - браузера. В настоящее время самым распространённым браузером является Internet Explorer компании Microsoft.

Для того, чтобы браузер отобразил содержимое WEB- страницы в том виде, в каком его задумал веб-дизайнер , необходимо браузеру каким-то образом сообщить: как будет выглядеть отображаемая страница. Для этого был придуман специальный язык разметки текста. В упрощённом представлении, для понимания его действия - это набор инструкций, которые предписывают что и в каком виде в данном месте следует отобразить.

Пример:

<Отобразить жирный заголовок, помещённый в центр страницы>

Текст заголовка

<Закончить отображение жирного заголовка, помещённого в центр страницы>

<Отобразить абзац текста, отформатированного по левому краю>

Текст абзаца

<Закончить отображение абзаца текста, отформатированного по левому краю>

<отобразить изображение из файла img.jpg шириной 250 пикселов без рамки>

и так далее.

Таким языком разметки стал HTML который в настоящее время является базовым языком для отображения WEB-страниц в современных браузерах. Другими словами, чтобы отобразить в браузере какой-нибудь документ, необходимо его представить в формате HTML.

Начало истории HTML восходит к далёкому 1969 году, когда сотрудник компании IBM Чарльз Гольдфарб, создал прототип языка для разметки технической документации, названного в последствии GML, ставшего международным стандартом и переименованным в SGML(Standart Generalized Markup Language). Этот стандарт устанавливал синтаксис записи элементов разметки, а также правила определения новых элементов и указания структурных соотношений между ними. Для практической же разметки необходимо было придумать приложение SGML - набор определений элементов, представляющих собой формальное описание структуры документа. Принципы, на которых строится язык SGML, оказали влияние на многие компьютерные разработки, однако сам язык SGML не получил заметного распространения до тех пор, пока в 1991 году сотрудники Европейского института физики частиц (CERN), занятые созданием системы передачи гипертекстовой информации через Интернет не выбрали SGML в качестве основы для нового языка разметки гипертекстовых документов. Этот язык - самое известное из приложений SGML был назван HTML.

Язык HTML (HyperText Markup Language) не является языком программирования в том смысле, в каком мы привыкли представлять себе такие языки программирования, как C, C++, Java, Basic и другие. При помощи HTML невозможно написать полноценное приложение, которое бы работало в системе MS-DOS, Windows или какой-либо другой ОС. HTML предназначен для создания документов, размещаемых на Web-сервере, доступ к которому осуществляется по сети Интернет, а выполнение программ, написанных на HTML происходит непосредственно в браузере.

Написать сайт можно, используя простой текстовый редактор, например Блокнот. Достаточно написать простой текст и сохранить его с расширением .html или . htm. У вас получится WEB-страница, которую вы сможете просмотреть в браузере. Но для того, чтобы текст на странице смотрелся красиво, необходимо применить к нему HTML-форматирование.

Язык HTML состоит из так называемых тэгов.

Все тэги HTML начинаются с "<" (левой угловой скобки) и заканчиваются символом ">" (правой угловой скобки). Как правило, существует стартовый тэг и

завершающий тэг. Для примера приведем тэги заголовка, определяющие текст, находящийся внутри стартового и завершающего тэга и описывающий заголовок документа:

Завершающий тэг выглядит также, как стартовый, и отличается от него прямым слэшем перед текстом внутри угловых скобок. В данном примере тэг - о завершении текста заголовка.

Некоторые тэги, такие, как

(тэг, определяющий абзац), не требуют завершающего тэга, но его использование придает исходному тексту документа улучшенную читаемость и структурируемость.

В разных операционных системах имеются различные редакторы, которые можно использовать для создания документов HTML. Если вы используете:

- Microsoft Windows запустите редактор Notepad;
- Mac OS запустите редактор SimpleText;
- OSX запустите редактор TextEdit (обязательно измените настройки "Rich Text" или "Расширенный текст" на "Plain text" или "Простой текст" и отметьте пункт "Ignore rich text commands in HTML files" или "Игнорировать команды расширенного текстового формата в файлах HTML").

Файлы HTML можно создавать и в редакторе Microsoft Word, в котором имеется возможность сохранить документ как Web-страницу (в меню "Файл"), однако использовать эту возможность не рекомендуется. Во-первых, потому что HTML-код, генерируемый MS Word не оптимален и содержит множество ненужных элементов разметки, и, во-вторых, автоматическая генерация кода не будет способствовать изучению и правильному пониманию HTML.

Имеется также большое количество специализированных редакторов для создания файлов HTML, таких как FrontPage, Macromedia Dreamweaver или Adobe Web Bundle, которые обладают возможностью WYSIWYG (What You See Is What You Get - что видишь, то и получишь). С их помощью можно легко создавать документы HTML, при помощи кнопок и элементов меню, а не писать самому теги разметки. Однако, как уже отмечалось выше, тем, кто хочет стать технически грамотным разработчиком Web, настоятельно рекомендуется использовать простой текстовый редактор для начального изучения HTML.

```
<html                                                                    >
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
<title></title>
</head>
<body>
</body>
</html>
```

Именно так выглядит "пустой" WEB-документ, написанный на языке разметки HTML.

Напечатайте следующий текст:

```
<html>
<head>
<title>Это заголовок страницы</title>
</head>
<body>
<h1>Здравствуйте!</h1>
<p>
Это моя первая страница HTML.
<b>Этот текст выводится жирным шрифтом.</b>
</p>
</body>
</html>
```

Сохраните файл как "page1.htm".

При сохранении файла HTML можно использовать расширение .htm или .html. Расширение .htm было принято для старых версий операционных систем, которые допускали трехбуквенное расширение для файлов. В настоящее время практически все операционные системы не имеют подобного ограничения и можно использовать расширение .html.

Теперь посмотрите, как браузер отобразит вашу первую страницу. Запустите браузер Интернет. Выберите "Open" или "Open Page" ("Открыть" или "Открыть страницу") в меню File (Файл) браузера. Появится диалоговое окно. Выберите "Browse" или "Choose File" ("Просмотр" или "Выбрать файл") и найдите только что созданный файл HTML - "page1.htm" - выберите его и щелкните на кнопке "Open" ("Открыть"). В диалоговом окне должен появиться адрес, например "C:\MyDocuments\page1.htm". Щелкните на кнопке OK, и браузер выведет на экран вашу страницу.

За основу модели разметки документов в HTML принята теговая модель. *Теговая модель* описывает документ как совокупность контейнеров, каждый из которых начинается и заканчивается тегами. Т.е. документ HTML представляет собой не что иное, как обычный ASCII-файл, с добавленными в него управляющими HTML-кодами (тегами).

Теги HTML-документов в большинстве своем просты для понимания и использования, ибо они образованы с помощью общеупотребительных слов английского языка, понятных сокращений и обозначений. HTML-тег состоит из имени, за которым может следовать необязательный список атрибутов тега. Текст тега заключается в угловые скобки (< и >). Простейший вариант тега - имя, заключенное в угловые скобки, например <HEAD> или . Для более сложных тегов характерно различие атрибутов, которые могут иметь конкретные значения, определенные автором для видоизменения функции тега.

Атрибуты тега следуют за именем и отделяются друг от друга одним или несколькими знаками табуляции, пробелами или символами возврата к началу строки. Порядок записи атрибутов в теге значения не имеет. Значение атрибута, если таковое имеется, следует за знаком равенства, стоящим после имени атрибута. Если значение атрибута - одно слово или число, то его можно просто указать после знака равенства, не выделяя дополнительно. Все остальные значения необходимо заключать в одинарные или двойные кавычки, особенно если они содержат несколько разделенных пробелами слов. Длина значения атрибута ограничена 1024 символами. Регистр символов в именах тегов и атрибутов не учитывается, чего нельзя сказать о значениях атрибутов. Например, особенно важно использовать нужный регистр при вводе URL других документов в качестве значения атрибута HREF.

Чаще всего HTML-теги состоят из начального и конечного компонентов, между которыми размещаются текст и другие элементы документа. Имя конечного тега идентично имени начального, но перед именем конечного тега ставится косая черта (/) (например, для тега стиля шрифта - курсив закрывающая пара представляет собой , для тега заголовка <TITLE> закрывающей парой будет </TITLE>). Конечные теги никогда не содержат атрибутов. По своему значению теги близки к понятию скобок "begin/end" в универсальных языках программирования, которые задают области действия имен локальных переменных и т. п. Теги определяют область действия правил интерпретации текстовых тегов документа.

При использовании вложенных тегов в документе следует соблюдать особую аккуратность. Вложенные теги нужно закрывать, начиная с самого последнего и двигаясь к первому. Некоторые HTML-теги не имеют конечного компонента, поскольку они являются автономными элементами. Например, тег изображения <IMG>, который служит для вставки в документ графического изображения, конечного компонента не требует. К автономным тегам также относятся разрыв строки (
), горизонтальная линейка (<HR>) и теги, содержащие такую информацию о документе, которая не влияет на его отображаемое содержимое, например теги <META> и <BASE>.

В некоторых случаях конечные теги в документе можно опускать. Большинство браузеров реализованы так, что при обработке текста документа начальный тег воспринимается как конечный тег предыдущего. Самый распространенный тег такого

типа - тег абзаца <P>. Поскольку он используется в документе очень часто, то его обычно ставят только в начале каждого абзаца. Когда один абзац заканчивается, следующий тег <P> сигнализирует браузеру о том, что нужно завершить данный абзац и начать следующий. Большинство авторов тегом конца абзаца вообще не пользуются.

Есть и другие конечные теги, без которых браузеры отлично работают, например конечный тег </HTML>. Тем не менее, рекомендуется включать по возможности больше конечных тегов, чтобы избежать путаницы и ошибок при воспроизведении документа.

В Unix-подобных операционных системах, пользователи идентифицируются **идентификаторами пользователя**(англ. *User identifier, UID*).

Операционная система различает пользователей именно по UID (а не, например, по логину). Во многих системах существует возможность создать две записи пользователя с разными логинами, но одинаковыми UID; в результате оба логина будут иметь одинаковые права, так как с точки зрения системы они неотличимы (так как имеют одинаковый UID). Это может использоваться злоумышленниками: проникнув в систему и получив права root, взломщик может создать себе аккаунт с UID=0, чтобы потом возвращаться в систему под логином, не привлекающим внимания, но получать права root.

Множество допустимых значений UID зависит от системы; в общем случае UID допускает использование значений от 0 до 65535 с некоторыми оговорками:

- Суперпользователь всегда должен иметь UID, равный нулю (0).
- Пользователю nobody обычно присваивается или наибольший из возможных UID (в противоположность суперпользователю), или один из системных UID (см. ниже).
- UIDы с 1 по 100 по соглашению резервируются под системные нужды; некоторые руководства рекомендуют резервировать UIDы со 101 по 499 (в Red Hat) или даже 999 (в Debian).

Значение UID ставится в соответствие пользователю в файле */etc/passwd*. Файлы теневого пароля и Network Information Service также используют числовые UIDы. Идентификатор пользователя-владельца является необходимым атрибутом файла файловых систем Unix и процессов.

Некоторые операционные системы могут поддерживать 16-битные UIDы, что делает возможным создание 65536 уникальных идентификаторов, хотя современные системы с поддержкой 32-битных UIDов могут иметь 4,294,967,296 (2^{32}) различных значений идентификаторов.

Лекция 2.
Описание языка HTML. Теги языка HTML и их свойства. Создание HTML-документа. Структура и синтаксис документа.

Описание языка HTML. Теги языка HTML и их свойства.

1. Документ HTML.

Документ HTML представляет собой файл типа html или htm, находящийся на сервереИнтернета, в локальной сети или на жестком диске. Этот файл содержит обычный текст и текстовые команды разметки, называемые *тегами.* С помощью тегов можно решить две основные задачи:

- управлять содержимым документа, включая форматирование текста, разметку заголовков, создание списков и таблиц;
- управлять связями документа с другими ресурсами (изображениями, таблицами стилей, внешними программами, сторонними Web-страницами).

Теги HTML не задают определенные и точные атрибуты форматирования документа, как, например, Microsoft Word. Конкретный вид документа окончательно определяет только *программа-браузер* на компьютере. Необходимость такого подхода связана с разнородностью аппаратного и программного обеспечения устройств, подключенных к Интернету. HTML — *не язык программирования,* хотя Web-страницы могут как сами

являться результатом работы серверных программ, так и включать в себя специально подготовленные клиентские программы — скрипты и апплеты.

2. Теги языка HTML и их свойства.

Тег HTML имеет общий вид

<имя>содержимое</имя>

и действует на все, что расположено между парами треугольных скобок. Все теги, имеющие содержимое, должны *закрываться,* причем закрывающая часть </имя> отличается от открывающей <имя> только наличием символа «/». Теги могут вкладываться друг в друга *иерархически, но без пересечений,* т. е. допустимо вложение вида <тег1><тег2></тег2></тег1>, но не <тег1><тег2></тег1></тег2>. Тег вместе с содержимым часто называют *элементом* HTML.

Действие вложенных тегов может *объединяться,* т. е. если внутрь тега, создающего полужирное начертание шрифта, мы вложим тег курсива, то должен получиться полужирный курсив. Аналогично, внутрь тега, создающего ячейку таблицы, мы можем вставить тег подключения картинки — и картинка окажется внутри ячейки. HTML предоставляет большую свободу обращения с тегами, но существуют и ограничения — нельзя же вложить старшую матрешку внутрь младшей!

Теги делятся на *блочные* (block-level) и *текстовые* (inline). Первые могут содержать как текстовые, так и другие блочные теги. При отображении они всегда выводятся *с новой строки.* Вторые могут содержать только текст и другие текстовые теги, но не блочные. При отображении они выводятся *в текущей строке.*

Теги, имеющие содержимое и нуждающиеся в закрывающей части, называют *контейнерными,* а теги без содержимого и закрывающей части — *унарными.*

Внутри открывающей части большинство тегов содержит *атрибуты,* называемые также *параметрами* или *опциями.* Все эти названия просто означают команды, уточняющие действие тега. Какой ширины должна быть ячейка таблицы, формируемая тегом? Где хранится картинка, которую тег должен отобразить? На эти и многие другие вопросы отвечают атрибуты, всегда находящиеся в открывающей части тега. Атрибуты имеют общий вид имя=»значение» и разделяются между собой хотя бы одним пробелом, символом табуляции или перевода строки. Если значение атрибута состоит из одного слова, символы двойных кавычек можно не писать, хотя стандарт рекомендует указывать их всегда. Например, элемент вида

Hello

описывает контейнерный тег с именем p, имеющий один атрибут с именем align и значением »justify» Содержимое тега строка текста Hello.

В закрывающей части тега атрибуты не применяются.

Названия всех тегов и атрибутов *нечувствительны* к регистру символов. Тем не менее, лучше придерживаться единообразного их написания.

У каждого тега имеется набор допустимых для него атрибутов. Для многих атрибутов также заранее известен набор значений, которые они могут принимать. Чаще всего атрибуты можно пропускать, тогда браузер выводит документ, придерживаясь правил, принятых «по умолчанию». Существуют и обязательные атрибуты. Например, тег вставки картинки не сможет ее найти, если в атрибуте **src** не будет указано ее местоположение. Как правило, порядок следования атрибутов, допустимых для данного тега, может быть произвольным.

HTML довольно «демократичен», неправильный тег, лишний атрибут, недопустимое вложение тегов обычно просто игнорируются браузером и не приводят к «зависаниям» или сообщениям об ошибках. Разумеется, при этом может произойти неправильное форматирование документа.

3. Общая структура HTML-документа.

HTML-документ состоит из трех основных частей:

- строка декларации типа документа;
- заголовок документа, заключенный в тег <head>…</head>;
- тело документа, заключенное в тег <body>…</body> или <frameset>…</ frameset>.

Заголовок и тело документа заключаются в объединяющий их тег

<html >…< / html >.
Строка декларации является служебной и указывает на стандарт разработки документа. Для документа, строго соответствующего стандарту HTML 4.0, она представляется унарным тегом вида
<!DOCTYPE HTML PUBLIC «-//W3C//DTD HTML 4.0//EN»>.
На сегодняшний день поддержка деклараций рекомендуется стандартами, но редко реализуется Web-мастерами, в дальнейшем мы будем ее пропускать.
Заголовок в теге <head> содержит информацию об общих свойствах документа. Содержимое тега <head> непосредственно не отображается в окне браузера, однако используется браузером при работе с файлом. Заголовок может включать в себя следующие теги:
- титул окна документа (тег <title>);
- *метатеги* документа (теги <meta>);
- базовый адрес для ссылок (тег <base>);
- связь с другими документами (теги <link>);
- встроенные таблицы стилей для оформления (тег <style>);
- встроенные программы-сценарии клиента (теги <script>).

Из всех этих тегов нам необходим пока только <title>. <title> представляет собой контейнерный тег без атрибутов, его содержимым может быть любая строка текста, не содержащая других тегов. Именно эта строка будет выведена браузером в заголовке окна или вкладки. Содержимое тега <title> должно кратко отражать суть документа, его рекомендуемый размер — не более 60-80 символов. При добавлении ссылки на документ в папку **«Избранное»** браузера именем новой закладки становится именно содержимое тега <title>. Многие браузеры создают для каждой закладки отдельный файл, поэтому внутри тега <title> нежелательны символы двойных кавычек, <,>,&, !, ? и другие знаки, недопустимые в именах файлов. Составление «правильных» <title> и других заголовочных тегов — целое искусство.Пока же просто не будем забывать, что заголовок в теге <title> — «самый главный».
Тело документа содержит теги, предназначенные для отображения браузером, и ограничено контейнерным тегом <body>…</body> или <frame-set>…</frameset>, если документ разбит на несколько *окон-фреймов.*
Все тэги HTML начинаются с < (левой угловой скобки) и заканчиваются символом > (правой угловой скобки). Как правило, существует стартовый тэг и завершающий тэг. Для примера приведем тэги заголовка, определяющие текст, находящийся внутри стартового и завершающего тэга и описывающий заголовок документа:

<TITLE> Заголовок документа </TITLE>

Завершающий тэг выглядит также, как стартовый, и отличается от него прямым слэшем перед текстом внутри угловых скобок. В данном примере тэг <TITLE> говорит WEB-броузеру об использовании формата заголовка, а тэг </TITLE> - о завершении текста заголовка.
Некоторые тэги, такие, как <P> (тэг, определяющий абзац), не требуют завершающего тэга, но его использование придает исходному тексту документа улучшенную читаемость и структурируемость.
HTML не реагирует на регистр символов, описывающих тэг, и приведенный ранее пример может выглядеть следующим образом:

<TITLE> Заголовок документа </TITLE>

Структура документа
Когда WEB-броузер получает документ, он определяет, как документ должен быть интерпретирован. Самый первый тэг, который встречается в документе, должен быть тэгом <HTML>. Данный тэг сообщает WEB-броузеру, что ваш документ написан с использованием HTML. Минимальный HTML-документ будет выглядеть так:

<HTML> ...тело документа... </HTML>

Заголовочная часть документа <HEAD> Тэг заголовочной части документа должен быть использован сразу после тэга <HTML> и более нигде в теле документа. Данный тэг представляет из себя общее описание документа. Избегайте размещать какой-либо текст

внутри тэга <HEAD>. Стартовый тэг <HEAD> помещается непосредственно перед тэгом <TITLE> и другими тэгами, описывающими документ, а завершающий тэг </HEAD> размещается сразу после окончания описания документа.
Например:

<HTML>
<HEAD>
<TITLE> Список сотрудников </TITLE>
</HEAD>

Лекция 3

Служебные теги, теги форматирования текста и таблиц. Макетирование документа с фиксированных и динамических таблиц

Служебные теги

К служебным тегам относятся те, которые несут в себе соответственно всю служебную информацию. Это мета заголовки страницы, версия документа, информация об авторе, таблицы стилей и т.д.
Первый тэг с которым я вас познакомлю это:
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML+RDFa 1.0//EN" "http://www.w3.org/MarkUp/DTD/xhtml-rdfa-1.dtd">
С него начинается код наверное 97% всех сайтов. Что он означает? Он говорит о том что это html документ с определенной версией. Это нужно для того, чтобы браузер понимал как ему отображать сайт, так как разные версии имеют небольшие отличия в синтаксисе(правописании) некоторых тегов. Более подробно о том какие бывают версии можно почитать тут .
Дальше по коду у нас идет <html></html>. Тут все просто, внутри него мы будем размещать весь наш код html. Получается, если мы создаем страницу, то код ее должен выглядеть примерно так:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
Вот тут будет размещаться код всего нашего HTML документа
</html>
```

Теперь давайте пропишем <head></head> — внутри этого тега всегда прописываются все служебные теги:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
тут будут все служебные теги
</head>

</html>
```

Дальше правильно прописать мета данные html документа — заголовок, краткое описание и ключевые слова. Ни в коем случае нельзя пренебрегать этим, так как сео оптимизация страницы без этих служебных тегов невозможна. Вот пример правописания:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Тут пишем заголовок нашей страницы</title>
<meta name="description" content="Тут пишем краткое описание страницы" />
<meta name="keywords" http-equiv="keywords" content="Здесь указываем ключевые слова" />
</head>

</html>
```

Последний служебный тэг с которым я вас сегодня познакомлю отвечает за подключение к странице файла каскадных стилей css (он отвечает за оформление, внешний вид и т.д.) Выглядит он так:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Тут пишем заголовок нашей страницы</title>
<meta name="description" content="Тут пишем краткое описание страницы" />
<meta name="keywords" http-equiv="keywords" content="Здесь указываем ключевые слова" />
<link rel="stylesheet" href="css/theme.css" type="text/css" />
</head>

</html>
```

Заголовки

Со служебными тегами мы закончили, пришло время переходить к основной части кода нашей страницы, к телу документа, отсюда соответствующее название следующего тега <body> Выглядит он так:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Тут пишем заголовок нашей страницы</title>
<meta name="description" content="Тут пишем краткое описание страницы" />
<meta name="keywords" http-equiv="keywords" content="Здесь указываем ключевые слова" />
<link rel="stylesheet" href="css/theme.css" type="text/css" />
</head>
<body>
Здесь будет основная, видимая часть нашей страницы.
</body>
</html>
```

Для выделения и форматирования Важных моментов в тексте и на сайте в целом существуют заголовки — h1, h2-h6. Их использовать я крайне рекомендую, так как с ними информация воспринимается более форматировано и читабельно. Самый главный по значимости заголовок это h1. Пишется он следующим образом: <h1>Заголовок первого уровня</h1> Все остальные заголовки пишутся так же, только меняется само название: <h2>Заголовок второго уровня</h2> и т.д.

Тэги отвечающие за форматирование текста.

 — означает начало абзаца. Всегда выводится с новой строки. Этот тэг является блочным элементом и имеет отступ

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Тут пишем заголовок нашей страницы</title>
<meta name="description" content="Тут пишем краткое описание страницы" />
<meta name="keywords" http-equiv="keywords" content="Здесь указываем ключевые слова" />
<link rel="stylesheet" href="css/theme.css" type="text/css" />
</head>
<body>
<p>Здесь будет основная, видимая часть нашей страницы.</p>
<p>Тут будет написан второй абзац нашего текста.</p>
</body>
</html>
```

Дальше давайте рассмотрим два тега отвечающих за выделение (акцентирование) текстовых участков, это: — выделение **жирным шрифтом** и — *курсивный шрифт* (под наклоном):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Тут пишем заголовок нашей страницы</title>
 <meta name="description" content="Тут пишем краткое описание страницы" />
 <meta name="keywords" http-equiv="keywords" content="Здесь указываем ключевые слова" />
 <link rel="stylesheet" href="css/theme.css" type="text/css" />
</head>
<body>
 <p>Здесь будет основная, <strong>видимая часть</strong> нашей страницы.</p>
 <p>Тут будет написан <em>второй</em> абзац нашего текста.</p>
</body>
</html>
```

Как видно из примера я использовал теги выделения (em, strong) внутри тега . Теперь Вы знаете что так делать можно.

Теперь я вам покажу как делать ссылку. За это отвечает тэг <a>, у него есть обязательный атрибут href при помощи которого нужно указывать ссылку на страницу на которую Вы ссылаетесь. Вот пример:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Тут пишем заголовок нашей страницы</title>
 <meta name="description" content="Тут пишем краткое описание страницы" />
 <meta name="keywords" http-equiv="keywords" content="Здесь указываем ключевые слова" />
 <link rel="stylesheet" href="css/theme.css" type="text/css" />
</head>
<body>
 <p>Здесь будет основная, <strong>видимая часть</strong> нашей страницы.</p>
 <p>Поисковик <a href="http://yandex.ru">яндекс</a> лучшее решение для поиска ответов!</p>
 <p>Тут будет написан <em>второй</em> абзац нашего текста.</p>
</body>
</html>
```

 — Это самый популярный тег в сегодняшней верстке сайтов, так как это блочный элемент позволяющий собирать сайт что называется по кирпичикам.

И в конце я не мог не вспомнить про таблицу. Это старый тэг — лично я им стараюсь уже не пользоваться, мне гораздо удобнее верстать дивами (div), но не рассказать про таблицы я просто не мог. Итак…для начала напишу, а потом расшифрую что и зачем:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Тут пишем заголовок нашей страницы</title>
<meta name="description" content="Тут пишем краткое описание страницы" />
<meta name="keywords" http-equiv="keywords" content="Здесь указываем ключевые слова" />
<link rel="stylesheet" href="css/theme.css" type="text/css" />
</head>
<body>
<p>Здесь будет основная, <strong>видимая часть</strong> нашей страницы.</p>
<p>Поисковик <a href="http://yandex.ru">яндекс</a> лучшее решение для поиска ответов!</p>
<table>
 <tr>
  <td>Ячейка - 1</td>
  <td>Ячейка - 2</td>
 </tr>
</table>
<p>Тут будет написан <em>второй</em> абзац нашего текста.</p>
</body>
</html>
```

<td>-это ячейка

<tr>-это строка

В построении любой таблицы обязательно есть строка, а в ней ячейка. Как Вы понимаете за строку у нас отвечает тег <tr></tr>, а за содержание этой строки (т.е. ячейку) тэг <td></td> и все это помещается между тэгами <table></table>, что собственно говоря и свидетельствует о таблице. С первого раза может сразу и не понять этот принцип, хотя тут

нет ничего сложного. Именно поэтому мне больше нравится использовать — меньше кода и меньше заморочек.

Теги могут быть как одиночные, так и парные. Парные теги открываются <код тега> и закрываются </код тега>, все что расположено между открывающим и закрывающим тегами, попадет под его влияние. Теги могут иметь различные атрибуты, которые указываются с помощью специальных параметров. Теги могут быть вложенными один в другой и т.д.

Внутри html документа необходимо использовать так называемые теги. Тег - это определенный условный код, обычно заключенный в скобки типа <>. Этот код дает понять браузеру, какой тип данных размещен между тегами, как нужно отображать эти данные, например такую информацию предоставляют теги форматирования текста. Тегов существует довольно много и начинающему вебмастеру тяжело в них ориентироваться, поэтому рассмотрим основные теги, которые могут Вам пригодиться уже при составлении первого сайта. Ниже в таблице тегов приведены не все теги, но основные выделены жирным шрифтом.

Основные теги	
<html></html>	Определяет тип документа как web страницу.
<head></head>	Служебная часть документа, не отображаемая в браузере, тут размещаются заголовок, описание, ключевые слова и т.д.
<body></body>	Между этими тегами располагается непосредственно сама web сраница, которую Вы видите на экране, могут размещаться и другие типы данных.
<meta name="allow-search" content="?">	Определяет способ сканирования данных для роботов поисковых систем.
<meta name="robots" content="?">	Указывается информация для поисковых роботов.
<meta name="author" content="?">	В данном теге можно указать авторство.
<meta name="keywords" content="?">	Для обеспечения работы с поисковымисерверами , указываются ключевые слова, через запятую.
<base href="www.?">	Применяется для не полностью прописанных ссылок, указывает начальный адрес в интернете
<meta name="description" content="?">	Краткое описание web страницы, будет показано в результатах поиска поисковиками.
<title></title>	Тег для заголовка страницы
<body bgcolor="?">	Определяет цвет фона web страницы
<body text="?">	Определяет цвет текста web страницы
<body link="?">	Задается цвет гиперссылок в нормальном состоянии
<body vlink="?">	Задает цвет уже просмотренных гиперссылок.
<body alink="?">	Задает цвет выделенной гиперссылки.
** **	Тег заголовка, создает самый крупный заголовок, цифра 1 может меняться в пределах от 1 до 6. Шесть будет самый мелкий заголовок.
	Делает текст полужирным.

	Форматирует текст в наклонный.
<tt></tt>	Иммитация стиля печатной машинки.
<kbd></kbd>	Иммитация стиля печатной машинки.
<var></var>	Название переменных отображается курсивом
<cite></cite>	Выделение цитат курсивом
<address></address>	Создает абзац с текстом выделенным курсивом.
	Курсив (воспринимается поисковиками как выделение)
	Полужирное форматирование текста (воспринимается поисковиками, как особо сильное выделение)
	Устанавливает размер шрифта, значение от 1 до 7.
	Задает цвет текста.
Оформление гиперссылок в html	
****	Создает гиперссылку, имеется ввиду как внутренние так и внешние гиперссылки.
<atarget="?">	Указывает в каком окне открывать гиперссылку. ***параметры Значение*** _Blank Открыть в новом окне _Parent Открыть в текущем окне _Self Открыть в окне без использования действующих фреймов _Top Игнорировать фреймы
****	Гиперссылка для отправления почтового сообщения по электронной почте.
	Создает гиперссылку на метку в текущей странице.
	Ставит метку для гиперссылки на текущей странице.
	Гиперссылка на метку другой страницы.
Форматирование текста	
** **	Создает новый параграф
** **	Выравнивает параграф относительно одной из сторон документа, значения: left, right, justify или center
<nobr>	Запрещает перевод строки.
<wbr>	Указывает где разбивать строку для переноса при необходимости.
** **	Вставляет перевод строки.
<blockquote></blockquote>	Создает отступы с обеих сторон текста.
<dl></dl>	Создает список определений.
<dt>	Определяет каждый из терминов списка

<dd>	Описывает каждое определение
** **	Форматирует текст в пронумерованный список
	Определяет каждый элемент списка и присваивает порядковый номер
	Тег для создания ненумерованного списка
	Отмечает каждую новую позицию в списку гружком или квадратиком.
	Данный тег используется для форматирования больших блоков текста HTML документа, также используется в таблицах стилей
Графические элементы на странице	
<img src="name">	Вставляет изображение на страницу.
<img src="name" align="?">	Форматирует положение изображения в документе, может иметь значения: left, right, center; bottom, top, middle.
<img src="name" border="?">	Устанавливает толщину рамки вокруг изображения
<img src="name" vspase="?">	Устанавливает поля сверху и снизу
<img src="name" hspase="?">	Устанавливает поля слева и справа.
<img src="name" alt="?">	Создает всплывающую подсказку
<hr>	Добавляет горизонтальную линию.
<hr size="?">	Указывает толщину линии.
<hr width="?">	Указывает ширину линии в пикселах или процентах.
<hr noshade>	Линия без тени.
<hr color="?">	Определяет цвет линии.
Создание таблиц	
<table></table>	Тег создающий таблицу.
<tr></tr>	Задает строку в таблице.
<td></td>	Задает отдельную ячейку в таблице.
<th></th>	Задает заголовок таблицы (обычная ячейка с форматированием по центру и полужирным текстом)
Какие бывают атрибуты таблицы	
<caption></caption>	Назначает подпись таблицы
<table border="#">	Определяет толщину рамки.
<table cellspacing="#">	Определяет расстояние между ячейками.
<table cellpadding="#">	Поля для текстового содержимого ячейки.
<table width="#">	Устанавливает ширину таблицы. (Значение параметра может быть в пикселях или процентах)
<table height="#">	Устанавливает высоту таблицы. (Значение параметра может быть в пикселях или процентах)

<tr align="?">или <td align="?">	Определяет горизонтальное выравнивание положения ячеек в таблице, может иметь значения: left, center, right.
<tr valign="?"> или <td valign="?">	Определяет вертикальное выравнивание положения ячеек в таблице, может иметь значения: top, middle, или bottom.
<td colspan="#">	Указывает количество столбцов, объединенных в одной ячейке.
<td rowspan="#">	Указывает количество строк, объединенных в одной ячейке.
<td nowrap>	Не позволяет программе просмотра делать перевод строки в ячейке таблицы.
<td width="#">	Задает ширину ячейки таблицы в пикселях или процентах.
<td height="#">	Задает высоту ячейки таблицы в пикселях или процентах.
Фреймы в документах html	
<frameset></frameset>	Предшествует тегу <body> на фреймовой странице.
<frameset rows="value,value">	Определяет строки в таблице фреймов, значение высоты которых определяется количеством пикселей или в процентах.
<frameset cols="value,value">	Определяет столбцы в таблице фреймов, значение ширины которых определяется количеством пикселей или в процентах.
<frame>	Задает единичный фрейм или область в таблице фреймов.
<noframes></noframes>	Задает информацию, которую увидит посетитель в случаях, когда показ фрейма невозможен.
Используемые атрибуты фреймов html	
<frame src="URL">	Привязка web страницы к окну фрейма.
<frame name="name">	Назначает фрейму имя, для улучшения маршрутизации информации между фреймами и страницами.
<frame marginwidth="#">	Отступы по горизонтали внутри фреймового окна.
<frame marginheight="#">	Отступы по вертикали внутри фреймового окна.
<frame scrolling=VALUE>	Создает линейку прокрутки фрейма; значение value может быть "yes," "no," или "auto". По умолчанию - auto.
<frame noresize>	Запрещает изменение размеров фреймового окна посетителем сайта.
Ифрейм и его атрибуты	
<iframe></iframe>	Создает контейнер, который может содержать любые элементы. Другие элементы страницы

	будут обтекать данный контейнер.
<iframe src="URL">	Привязывает web страницу к окну ифрейма.
<iframe name="name">	Назначает ифрейму имя, для улучшения маршрутизации информации.
<iframe vspase="?">	Задает отступы по горизонтали снаружи ифрейма.
<iframe hspase="?">	Задает отступы по вертикали снаружи ифрейма.
<iframe marginwidth="#">	Задает поля по горизонтали внутри ифрейма.
<iframe marginheight="#">	Задает поля по вертикали внутри ифрейма.
<iframe scrolling=VALUE>	Создает линейку прокрутки фрейма; значение value может быть "yes," "no," или "auto". По умолчанию - auto.
<iframe width="#">	Задает ширину ифрейма
<iframe height="#">	Задает высоту ифрейма;
<iframe title="?">	Закрепляет текст контекстной помощи.
Теги относящиеся к формам	
<form></form>	Тег для создания формы.
<select multiple name="NAME" size="?"> </select>	Создает меню со скроллингом. Атрибут size определяет численность пунктов в меню видимых сразу без использования линейки прокрутки.
<option>	Задает каждый элемент меню по отдельности.
<select name="NAME"></select>	Создает ниспадающее меню
<textarea name="NAME" cols=40 rows=8></textarea>	Размещает редактируемое окно для ввода текста. Атрибуты columns - задает значение ширины окна, а rows соответственно высоту.
<input type="checkbox" name="NAME">	Размещает элемент checkbox.
<input type="radio" name="NAME" value="x">	Размещает элемент типа radio кнопки.
<input type=text name="foo" size=20>	Размещает строку для ввода текста. Атрибутом Size задается длина в символах.
<input type="submit" value="NAME">	Размещает на странице кнопку "Отправить"
<input type="image" border="0" name="NAME" src="name.gif">	Размещает на странице "Отправить" с использованием графического изображения.
<input type="reset">	Размещает на странице "Очистить".

Лекция 4.
Теги включения ссылок, изображений, мультимедийных объектов. Фреймы. Формы

Вставка рисунка.

Чтобы вставить в документ изображение, необходимо воспользоваться тегом <IMG>, который, например, может выглядеть следующим образом.

<IMG SRC="название_файла_с_рисунком.gif" WIDTH="ширина" HEIGHT="высота">

Здесь параметр SRC= задает имя файла с изображением, которое должно быть задано

обязательно, а WIDTH= и HEIGHT= -соответственно высоту и ширину изображения. Если они не совпадают с реальными размерами картинки, то браузер перемасшабирует изображение. Если эти параметры вообще не не заданы, то будут оспользованы реальные размеры изображения.

Чтобы изображение примыкало к границе страницы и обтекалось текстом документа, следует указать также атрибут ALIGN=LEFT ALIGN=RIGHT.Пример:

<IMG SRC="название_файла_с_рисунком.gif" WIDTH="ширина" HEIGHT="высота" ALIGN=LEFT>

Фоновые изображения.

Обычно текст в окне браузера выводится на стандартном фоне, белом (Internet Explorer) или сером (Netscape Navigator). Если web-страница насыщена рисунками, то этого бывает достаточно. Если же страница содержит мало вставных рисунков, имеет смысл рассмотреть возможность задания фонового изображения. В таком случае текст не изображается на одноцветном фоне, а накладывается на фоновое изображение. Чтобы это сделать, надо указать этот рисунок в теге <BODY>.

<BODY BACKGROUND="имя_файла_с_фоновым_изображением.gif">

Выбранное изображение накладывается на экран и "размножается" так, чтобы занять все имеющееся место. То есть, первый экземпляр изображения помещается в левый верхний угол окна браузера. Если при этом занята не вся ширина окна, то справа размещается еще одна копия изображения и так далее. Точно так же при необходимости под верхним рядом озображения помещается следующий ряд копий и так далее. Обычно в качестве фонового изображения рекомендуется использовать неяркое изображение пастельных тонов с узкой цветовой гаммой. Когда фоновый ривунок представляет собой элемент орнамента, повтряющиеся копии складываются в единое изображение. Цвет текста должен быть контрастным по отношению к фоновому изображению. Если фоновый рисунок светлый, то текст следует изображать темными буквами, а если фон темный (например "звезное небо"), то на таком фоне хорошо различим светлый текст.

Еще в место фоновых изображений могут использоваться однотонные цвета, которые задаются в HEX или RGB значениях, пример ниже:

<BODY BGCOLOR="#RGB значение">

Мультимедийные объекты.

Для того, чтобы все время, пока данная web-страница находится на экране, звучала фоновая музыка, вы можете воспользоваться тегом <BGSOUND>, который должен выглядеть следующим образом. <BGSOUND SRC="имя_звукового файла.wav" LOOP="число_повторений"> Эта строка должна входить в тело документа HTML, то есть находится между тегами <BODY> и </BODY>. Звуковой файл, заданный парметром SRC=, будет автоматически закружен, после чего начнется его воспроизведение. Параметр LOOP= указывает, сколькораз этот файл должен быть воспроизведен. Если Вы хотите, чтобы музыка звучала непрерывно, пока посетитель продолжает изучать открытую страницу, укажите атрибут LOOP=INFINITE.

<BGSOUND SRC="имя_звукового файла.wav" LOOP="INFINITE">

Разумеется, в качестве "фонового звука" можно использовать и речевой комментарий, относящийся к тому, что изображено на данной странице. Этот комментарий также активизируется автоматически. В этом случае, разумеется, нет необходимости задавать его многократное повторение, хотя лучше предусмотреть возможность снова включить его воспроизведение по желанию читателя. Как это сделать, будет описано ниже. Вставка на страницу видеоролика производится аналогично тому, как вставляется статическое изобращение. Соответствующий тег имеет следующий вид. <IMG DYNSRC="имя_файла_с_роликом.avi" START="способ запуса" LOOP="число повторов">

Здесь параметр DYNSRC= задает имя файла с видеороликом. Параметр START= задает способ запуска видеоролика. Если Вы укажите START="OPENFILE", то воспроизведение ролика начнется сразу после загрузки страницы, а если указать START="MOUSEOVER", то воспроизведение начнется после того как указатель мыши будет наведен (щелчок не нужен) на изображение, которое представляет собой превый кадр видеоролика. Параметр LOOP задает число повторений видеоролика.

Создание ссылок.

Гипертекстовые ссылки, представляющие собой основной инструмент прехода от одной страницы к другой, задаются при помощи специального парного тега. Соответствующий фрагмент документа должен выглядеть следующим образом.

<A HREF="имя_файла.html"> текст ссылки </A>

В качестве значения параметра HREF= используется имя файла, который должен быть открыт при щелчке на ссылки. Текст, находящийся на экране между открывающим тегом <A> и закрывающим </A>, выделяется другим цветом и подчеркивается, так что наличие ссылки легко заметить. Обычно этот текст как раз и объясняет, что содержится на той странице, куда Вы перейдете, если щелкните на ссылке.

Нет необходимости добиваться того, чтобы каждая ссылка непременно указывала на другой файл. В частности, допустимо создавать очень длинные страницы, занимающие много экранов. Для перемещения по такой странице, помимо использованиия полос прокрутки, могу использоваться и гипертекстовые ссылки. С их помощью можно очень быстро перейти к нужному месту страницы.

Для реализации такого перехода нужны два элемента: ссылка, указывающая, кда надо перейти, и якорь, фиксирующий место перехода. Ссылка выглядит почти так же, как это было указано выше.

<A HREF="#название_якоря"> текст ссылки </A>

Якорь также использует теги <A> и </A> и задается следующим образом.

<A NAME="название_якоря"> текст </A>

Обратите внимание на название якоря. Это произвольный текст, который никогда не появляется на экране и однозначно определяет место перехода. Если у Вас на одной странице имеется несколько якорей, все они должны иметь различные названия. Символ "#" при задании ссылки указывает, что дальше следует не имя файла, а название якоря. Текст, расположенный между тегами <A> и </A>, задающими якорями, появится на экране, если Вы щелкните на ссылке.

Имейте в виду также, что вы можете сочетать оба способы задания ссылки. Если пользователь щелкнет на ссылке, заданной как:

<A HREF="имя_файла#название_якоря"> текст ссылки </A>

то произойдет переход к указанному якорю внутри заданного файла. Все эти возможности позволяют сформировать систему гипертекстовых ссылок так, как это нужно в конкретных обстоятельствах.

Использование изображений ссылок, ссылки на другие элементы.

В качестве гипертекстовой ссылки может использоваться не только строка текста, но и изображение. Если поместить тег изображения <IMG> между тегами <A> и </A>, задающими ссылку, то все изображение преватится в гиперссылку.

<A HREF="имя_файла.html"><IMG SRC="имя_файла_с_изображением.gif"></A>

На экране такое изображение выделяется специальной рамкой, хотя в документе можно включить и вспомогательный текст, подсказывающий человеку, который в данный момент просматривает web-страницу, что данное изображение является ссылкой (с помощью атрибута ALT="вспомогательный текст" тега <IMG>)

Гипертекстовая ссылка может указывать также не на другой документ HTML, а на звуковой или видеоролик. Программы-браузеры либо воспроизводят файлы самостоятельно, либо вызывают вспомагательные приложения, предназначенные для воспроизведения этих файлов. Например, если Вы укажите следующий тег:

<A HREF="music.wav"> послушайте музыку </A>

то при щелчке на выделенном цветом (как ссылка) слове "послушайте музыку" начнется воспроизведение звукового файла "music.wav" (который, вероятно, содержит музыкальный фрагмент). Браузер распознает такие файлы по их расширениям. Это, в частности, позволяет Вам включить в состав одной web-страницы несколько мудьтимедийных фрагментов.

Использование подобных ссылок обычно предпочтительнее, чем применение фонового звука, по крайней мере, если речь идет о голосе. Этот метод дает читателю возможность самостоятельно выбрать, хочет ли он слушать то, что для него подготовили, и,

соответсвенно, готов ли он ждать, пока произойдет загрузка звукового файла, обычно не малого по размерам.

Фреймы

Фреймы разделяют окно браузера на отдельные области, расположенные рядом друг с другом. В каждую из таких областей загружается самостоятельная веб-страница. Поскольку вокруг фреймов существует много разговоров об их необходимости, далее приведем достоинства и недостатки фреймов, чтобы можно было самостоятельно решить стоит ли их использовать на своем сайте.

Достоинства фреймов

Простота

С помощью фреймов веб-страница разграничивается на две области, которые содержат навигацию по сайту и его контент. Механизм фреймов позволяет открывать документ в одном фрейме, по ссылке, нажатой в совершенно другом фрейме. Такое разделение веб-страницы на составляющие интуитивно понятно и логически обусловлено.

Быстрота

Для верстки без фреймов характерно размещение на одной странице и навигации и содержания. Это увеличивает объем каждой страницы и в сумме может существенно повлиять на объем загружаемой с сайта информации. А так как фреймы используют разделение информации на части, страницы с ними будут загружаться быстрее.

Размещение

Фреймы предоставляют уникальную возможность — размещение информации точно в нужном месте окна браузера. Так, можно поместить фрейм внизу браузера и независимо от прокручивания содержимого, эта область не изменит своего положения.

Изменение размеров областей

Можно изменять размеры фреймов «на лету», чего не позволяет сделать традиционная верстка HTML.

Загрузка

Загрузка веб-страницы происходит только в указанное окно, остальные остаются неизменными. С помощью языка JavaScript можно осуществить одновременную загрузку двух и более страниц во фреймы.

Недостатки фреймов

Навигация

Пользователь зачастую оказывается на сайте, совершенно не представляя, куда он попал, потому что всего лишь нажал на ссылку, полученную в поисковой системе. Чтобы посетителю сайта было проще разобраться, где он находится, на каждую страницу помещают название сайта, заголовок страницы и навигацию. Фреймы, как правило, нарушают данный принцип, отделяя заголовок сайта от содержания, а навигацию от контента. Представьте, что вы нашли подходящую ссылку в поисковой системе, нажимаете на нее, а в итоге открывается документ без названия и навигации. Чтобы понять, где мы находимся или посмотреть другие материалы, придется редактировать путь в адресной строке, что в любом случае доставляет неудобство.

Плохая индексация поисковыми системами

Поисковые системы плохо работают с фреймовой структурой, поскольку на страницах, которые содержат контент, нет ссылок на другие документы.

Внутренние страницы нельзя добавить в «Закладки»

Фреймы скрывают адрес страницы, на которой находится посетитель, и всегда показывают только адрес сайта. По этой причине понравившуюся страницу сложно поместить в закладки браузера.

Несовместимость с разными браузерами

Параметры фреймов обладают свойством совершенно по разному отображаться в различных браузерах. Причём противоречие между ними настолько явное, что одни и те же параметры интерпретируются браузерами совершенно по-своему.

Непрестижность

Весьма странный недостаток, который не имеет никакого отношения к техническим особенностям создания сайта, а носит скорее идеологический характер.

Сайты с фреймами считаются несолидными, а их авторы сразу выпадают из разряда профессионалов, которые никогда не используют фреймы в своих работах. Исключение составляют чаты, где без фреймов обойтись хотя можно, но достаточно хитрыми методами, а с помощью фреймов создавать чаты достаточно просто.

Надо отметить, что некоторые приведённые недостатки вполне обходятся. Так, с помощью скриптов можно сделать, что открытый в браузере отдельный документ формируется со всей фреймовой структурой. Поисковые системы также уже лучше индексируют фреймовые документы, чем это было несколько лет назад.

Лекция 5

Организация Web-страниц. Каскадные таблицы стилей. Способы определения стилей. Элементы стилей. Синтаксис стилей.

Для создания Web-страниц понадобится любой браузер - Internet Explorer или Netscape Communicator, а лучше оба, так как многие элементы HTML по-разному отображаются в разных программах просмотра и весьма желательно контролировать эту разницу. Кроме того, нужен любой текстовой редактор, например, Блокнот из Windows. Программа Блокнот нужна для подготовки HTML-файлов, а браузер - для просмотра и контроля сделанного. С помощью этих инструментов мы создадим сайт на своем локальномкомпьютере, после чего поместим его на один из WWW-серверов в Интернете, сделав, таким образом, ваши Web-страницы доступными всему миру.

В качестве примера создадим сайт лицея. Цель сайта - рассказать миру о лицее, сфере его деятельности, интересов, найти друзей, партнеров, спонсоров.

Для файлов нашего сайта нужна отдельная папка.

Создайте папку с именем Web на жестком диске вашего компьютера.

Теперь запустим программу Блокнот и приступим к работе.

Вы можете использовать любой другой текстовый редактор. Однако в этом случае следует сохранять файл как простой текст, чтобы избежать включения в Web-документ посторонних символов форматирования.

Сначала введем в окне программы Блокнот тэги, определяющие структуру любого HTML-документа. Напомним, что в HTML-коде допускается использовать любой регистр символов - верхний или нижний.

Введите с клавиатуры следующие основные тэги, поместив каждый из них, кроме закрывающего тэга </title>, в новой строке.

```
<html>
<head>
<title></title>
</head>
<body>
</body>
</html>
```

Для ввода парных тэгов вы можете использовать операции копирования и вставки через буфер обмена Windows с последующим добавлением символа слэш /.

Напомним, что первый <html> и последний </html> тэги означают соответственно начало и конец документа, элемент<head>...</head> определяет заголовок Web-страницы, элемент <body>...</body> - тело документа, а в элементе <title></title> мы сейчас укажем название Web-страницы.

Между открывающим <title> и закрывающим </title> тэгами вставьте название документа – Лицей № 1548. Этот элемент должен выглядеть следующим образом:

```
<title>Лицей № 1548</title>
```

Напомним, что название Web-страницы должно быть коротким и в максимальной степени отображать ее содержание.

Наша следующая задача - вставить в тело документа между тэгами <body>...</body> короткий текст-приветствие посетителям Web-страницы.

Вставьте пустую строку между тэгами <body> и </body> и введите в ней следующий текст:

Добро пожаловать на страничку лицея № 1548! Здесь Вы узнаете о нашей деятельности, о наших интересах и увлечениях, о наших успехах и достижениях

Для каждой Web-страницы вы можете определить цвет фона и цвет текста. Это выполняется с помощью атрибутов bgcolor itext открывающего тэга <body>. Для определения цвета как значения атрибутов существует два варианта:

• словесное указание имени цвета, например, white (белый). В языке HTML предусмотрено шестнадцать таких имен;

• цифровое обозначение в шестнадцатеричной записи, например, "#ffffff" - белый, которое указывает, каким образом цвет формируется из основных цветов - красного, зеленого, синего.

Конечно, словесное указание цвета более удобно и понятно. С другой стороны, численные обозначения дают больше возможностей, так как позволяют указать практически любой из 16777215 оттенков, тогда как словесные обозначения ограничены только шестнадцатью цветами.

Используем в качестве примера для фона нашей Web-страницы синий цвет (blue), а для текста - желтый (yellow).

Вставьте в открывающий тэг <body> атрибуты bgcolor=blue и text=yellow. Этот тэг должен принять вид:

<body bgcolor=blue text=yellow>

Ваш текстовый файл должен выглядеть примерно так:

<html>
<head>
<title>Лицей № 1548</title>
</head>
<body bgcolor=blue text=yellow>

Добро пожаловать на страничку лицея № 1548! Здесь Вы узнаете о нашей деятельности, о наших интересах и увлечениях, о наших успехах и достижениях

</body>
</html>

Теперь документ следует сохранить. Откройте ранее созданную папку Web, введите имя файла licey1548.html и нажмите кнопку Сохранить.

Обратите внимание: вы обязательно должны указать расширение имени файла .html или .htm, чтобы браузер смог его открыть.

Возможно, в вашем браузере размер шрифта текста будет крупнее или мельче, чем на рисунке. В таком случае выберите команду меню Вид • Размер текста • Средний в браузере Internet Explorer или Вид • Увеличить шрифт, Вид • Уменьшить шрифт в браузере Netscape Communicator, чтобы установить средний размер шрифта.

Следует иметь в виду, что разные браузеры могут по-разному отображать содержимое одного и того же HTML-файла. Поэтому при создании Web-страниц желательно всегда просматривать результат в обоих наиболее популярных браузерах - InternetExplorer и Netscape Communicator. В таком случае вы будете уверены, что посетитель вашего сайта увидит именно то, что вы хотите ему показать.

Так как в элементе <body></body> мы ввели текст без разбивки на абзацы, то в браузере он отображается в виде одного абзаца и выровнен влево. Теперь следует придать тексту более наглядный вид.

Каскадные (многоуровневые) таблицы стилей - cascading style sheets (CSS) - это мощный стандарт на основе текстового формата, определяющий представление данных в броузере. Если формат HTML предоставляет информацию о составе документа, то таблицы стилей сообщают как он должен выглядеть. Таким образом каскадные таблицы стилей дают возможность хранить содержимое отдельно от его представления. Стиль включает все типы элементов дизайна: шрифт, фон, текст, цвета ссылок, поля и расположение объектов на странице. CSS разрабатывались так, чтобы обеспечить

больший уровень контроля над размещением текста и графики. Каскадные таблицы стилей обеспечивают должный уровень единства оформления, организации и контроля во время разработки узла, который является недостижимым с помощью одного только HTML.

CSS предполагает 3 типа таблиц стилей - встроенные, внедренные (внутренние) и связанные (внешние).

Впервые идея форматирования HTML-документов с помощью CSS была рекомендована Консорциумом W3C в 1996 году. Эта рекомендация, которая была обновлена в 1998 году, используется Web-разработчиками и по сей день.

Что значит слово *"каскадный"?* Термин "каскадный" означает, что в одной странице HTML могут использоваться разные стили. Броузер, поддерживающий таблицы стилей, будет следовать их порядку (как по каскаду), интерпретируя информацию стилей. Это означает, что вы можете использовать все три типа стилей, и броузер будет интерпретировать сначала связанные, затем внедренные и, наконец, встроенные стили. Даже если ко всему узлу будут применены образцы стилей, можно будет управлять отдельными аспектами страниц с помощью внедренных стилей, а отдельными областями внутри этих страниц - с помощью встроенных стилей. Другой аспект каскадирования - *наследование (inheritance)*. Наследование означает, что если не указано иное, то конкретный стиль будет унаследован другими элементами страницы HTML. Например, если вы примените определенный цвет текста в теге <*p*>, то все теги внутри этого абзаца наследуют этот цвет, если не оговорено иное.

Методы

Существует три метода для применения таблицы стилей к документу HTML:

- *Встроенный (Inline)*. Этот метод позволяет взять любой тег HTML и добавить к нему стиль. Использование встроенного метода предоставляет максимальный контроль над всеми свойствами Web-страницы. Предположим, вы хотите задать внешний вид отдельного абзаца. Вы можете просто добавить атрибут style к тегу абзаца, и броузер отобразит этот абзац с помощью параметров стиля, добавленного в код.
- *Внедренный (Embedded)*. Внедрение позволяет контролировать всю страницу HTML. При использовании тега <style>, помещенного внутри раздела <head> страницы HTML, в код вставляются детализированные атрибуты стиля, которые будут применяться ко всей странице.
- *Связанный (Linked или External)*. Связанная таблица стилей - мощный инструмент, который позволяет создавать образцы стилей *(master styles)*, которые можно затем применять ко всему узлу. Основной документ таблицы стилей (расширение .css) создается Web-дизайнером. Этот документ содержит стили, которые будут едиными для всего Web-узла (неважно, содержит одну страницу или тысячи страниц). Любая страница, связанная с этим документом, будет использовать указанные стили.

Синтаксис таблицы стилей

Таблицы стилей строятся в соответствии с определенным порядком (синтаксисом), в противном случае они не могут нормально работать. Синтаксис всех методов, используемых для применения стилей к документа HTML, практически одинаков. Таблицы стилей составляются из определенных частей. Эти части включают следующие элементы:

- *Указатель (Selector)*. Указатель является элементом, к которому будут применяться назначаемые вами атрибуты. Это может быть просто тег типа заголовка (H1) или абзаца (P). Таблицы стилей позволяют использовать различные объекты, включая классы, которые будут кратко обсуждаться далее.
- *Свойство (Property)*. Свойство определяет указатель. Например, если в качестве указателя выбран абзац, вы можете включить свойства, определяющие этот указатель. В свойства входят такие элементы, как поля, шрифты и фоновые изображения. В таблицах стилей существует много свойств, которые можно использовать для того, чтобы определить указатель.
- *Значение (Value)*. Значения определяют свойства. Предположим, что у вас есть заголовок первого уровня H1(указатель) и вы включаете свойство type

family (семейство шрифта). Шрифт, который на самом деле будет применен к указанному фрагменту, задается значением этого свойства.

- *Описание (Declaration).* Свойства и значения объединяются, образуя описание.
- *Строка (Rule).* Указатель и описание образуют строку

Определение правил CSS

Итак, какскадная таблица стилей - это набор правил форматирования тегов HTML. Например, для того, чтобы цвет фона Web-страницы сделать черным, необходимо объявить следующее правило форматирования:

```
body{background:black}
```

Будьте внимательны! По умолчанию цвет шрифта - черный! Не следует претендовать на лавры Малевича! Ваш "Черный квадрат" Эрмитаж не купит!

В данном случае объявлено правило форматирования тега *body*, а именно - свойству стиля *background*присвоено значение *black* (черный). В результате применения этого правилацвет фона всего документа изменится на черный.

Обратите внимание: в таблице стилей теги HTML не заключаются в круглые скобки.

Свойства CSS должны находиться в фигурных скобках. Для каждого тега HTML можно указать не одно, а несколько свойств стиля.

Изменим с помощью CSS не только цвет фона Web-страницы, но и цвет шрифта (на белый).

```
body{background:black;color:white}
```

Формат самого правила не имеет значения, главное - правило должно читаться удобно и легко. Например, вышеприведенное правило можно записать и так:

```
body{
background:black;
color:white}
```

Одно и то же правило сстиля можно применить сразу к нескольким различным тегам HTML-страницы. Например:

```
body,td,h1{
background:black;
color:white}
```

Отдохнем - посмотрим как это выглядит на экране!

Возврат в начало страницы Возврат на главную страницу сайта

Встроенный стиль

Встроенный стиль применяется к любому тегу HTML с помощью атрибута *style* следующим образом:

```
<P style="font: 12pt Courier New"> The text in this line will
display as 12 point text using the Courier New font.
</P>
```

Или:

```
<p style="font: 12pt Arial">
The text in this line will display as 18 point text using the
 Arial font.
</p>
```

Посмотрим на результат:

The text in this line will display as 12 point text using the Courier New font.

The text in this line will display as 18 point text using the Arial font.

Можно добавлять встроенный стиль в любой тег HTML, в котором эта операция будет иметь смысл. Среди таких тегов можно назвать абзацы, заголовки, горизонтальные линии, якоря и ячейки таблиц. Ко всем этим элементам логично применять встроенные стили. Существуют два тега, которые помогают применять встроенные стили к разделам страницы. Это теги* (division - раздел)* и * (промежуток)*. Эти теги определяют диапазон текста, так что все, находящееся между ними, будет оформлено с помощью нужного стиля. Единственное различие

между <*div*> и <*span*> состоит в том, что <*div*> создает принудительный разрыв строки, а <*span*> - нет. Следовательно, нужно использовать <*span*> для изменения стиля любой части текста, меньшей абзаца.
Вот пример работы тега <*div*>:

<div style="font-family: Garamond; font-size: 18 pt;>"AII of the
text within this section is 18 point Garamond.
AII of the text within this section is 18 point Garamond.

и тега <*span*>:

 This text appears in the color red,
with no line break after the closing span tag and the rest of
the text.

Лекция 6.
Характеристика и возможности расширяемого языка разметки XML

XML (англ. *eXtensible Markup Language* — расширяемый язык разметки; произносится [*экс-эм-э́л*]) — рекомендованный Консорциумом Всемирной паутины язык разметки, фактически представляющий собой свод общих синтаксических правил. XML предназначен для хранения структурированных данных (взамен существующих файлов баз данных), для обмена информацией между программами, а также для создания на его основе более специализированных языков разметки (например, XHTML), иногда называемых *словарями*. XML является упрощённым подмножеством языка SGML.

Целью создания XML было обеспечение совместимости при передаче структурированных данных между разными системами обработкиинформации, особенно при передаче таких данных через Интернет. Словари, основанные на XML (например, RDF, RSS, MathML, XHTML,SVG), сами по себе формально описаны, что позволяет программно изменять и проверять документы на основе этих словарей, не зная ихсемантики, то есть не зная смыслового значения элементов. Важной особенностью XML также является применение так называемыхпространств имён (англ. *namespace*).

Достоинства

- XML(человеко-ориентированный) — это формат, одновременно понятный и человеку и компьютеру;
- XML поддерживает Юникод;
- в формате XML могут быть описаны основные структуры данных — такие как записи, списки и деревья;
- XML — это самодокументируемый формат, который описывает структуру и имена полей также как и значения полей;
- XML имеет строго определённый синтаксис и требования к парсингу, что позволяет ему оставаться простым, эффективным и непротиворечивым.
- XML также широко используется для хранения и обработки документов как он-лайн, так и офф-лайн:
- XML — формат, основанный на международных стандартах;
- иерархическая структура XML подходит для описания практически любых типов документов;
- XML представляет собой простой текст, свободный от лицензирования и каких-либо ограничений;
- XML не зависит от платформы;
- XML является подмножеством SGML (который используется с 1986 года). Уже накоплен большой опыт работы с языком и созданы специализированные приложения.
- XML не накладывает требований на расположение символов на строке [1]

Недостатки

- Синтаксис XML избыточен.

- Размер XML документа существенно больше бинарного представления тех же данных. В грубых оценках величину этого фактора принимают за 1 порядок (в 10 раз).
- Размер XML документа существенно больше, чем документа в альтернативных текстовых форматах передачи данных (напримерJSON [2]) и особенно в форматах данных оптимизированных для конкретного случая использования.
- Избыточность XML может повлиять на эффективность приложения. Возрастает стоимость хранения, обработки и передачи данных.
- Для большого количества задач не нужна вся мощь синтаксиса XML и можно использовать значительно более простые и производительные решения [3]

- XML не содержит встроенной в язык поддержки типов данных. В нём нет понятий «целых чисел», «строк», «дат», «булевых значений» итд
- Иерархическая модель данных, предлагаемая XML, ограничена по сравнению с реляционной моделью и объектно-ориентированными графами
  - Выражение не иерархических данных (например графов) требует дополнительных усилий
  - Кристофер Дейт отмечал, что «…XML является попыткой заново изобрести иерархические базы данных…» [4] (в 1980-е года иерархические базы данных были вытеснены реляционными базами данных).
- Пространства имён XML сложно использовать и их сложно реализовывать в XML парсерах
- Существуют другие, обладающие сходными с XML возможностями, текстовые форматы данных, которые обладают более высоким удобством чтения человеком (YAML [5] , JSON [6], SweetXML [7])

Использование:

- В первую очередь, эта технология может оказаться полезной для разработчиков сложных информационных систем, с большим количеством приложений, связанных потоками информации самой различной структурой. В этом случае XML - документывыполняют роль универсального формата для обмена информацией между отдельными компонентами большой программы.
- XML является базовым стандартом для нового языка описания ресурсов, RDF, позволяющего упростить многие проблемы в Web, связанные с поиском нужной информации, обеспечением контроля за содержимым сетевых ресурсов, создания электронных библиотек и т.д.
- Язык XML позволяет описывать данные произвольного типа и используется для представления специализированной информации, например химических, математических, физических формул, медицинских рецептов, нотных записей, и т.д. Это означает, что XML может служить мощным дополнением к HTML для распространения в Web "нестандартной" информации. Возможно, в самом ближайшем будущем XML полностью заменит собой HTML, по крайней мере, первые попытки интеграции этих двух языков уже делаются (спецификация XHTML).
- XML-документы могут использоваться в качестве промежуточного формата данных в трехзвенных системах. Обычно схема взаимодействия между серверами приложений и баз данных зависит от конкретной СУБД и диалекта SQL, используемого для доступа к данным. Если же результаты запроса будут представлены в некотором универсальном текстовом формате, то звено СУБД, как таковое, станет "прозрачным" для приложения. Кроме того, сегодня на рассмотрение W3C предложена спецификация нового языка запросов к базам данных XQL, который в будущем может стать альтернативой SQL.
- Информация, содержащаяся в XML-документах, может изменяться, передаваться на машину клиента и обновляться по частям. Разрабатываемые спецификации XLink и Xpointer поволят ссылаться на отдельные элементы документа, с учетом их вложенности и значений атрибутов.
- Использование стилевых таблиц (XSL) позволяет обеспечить независимое от конкретного устройства вывода отображение XML- документов.

- XML может использоваться в обычных приложениях для хранения и обработки структурированных данных в едином формате.

	GML	TML	ML

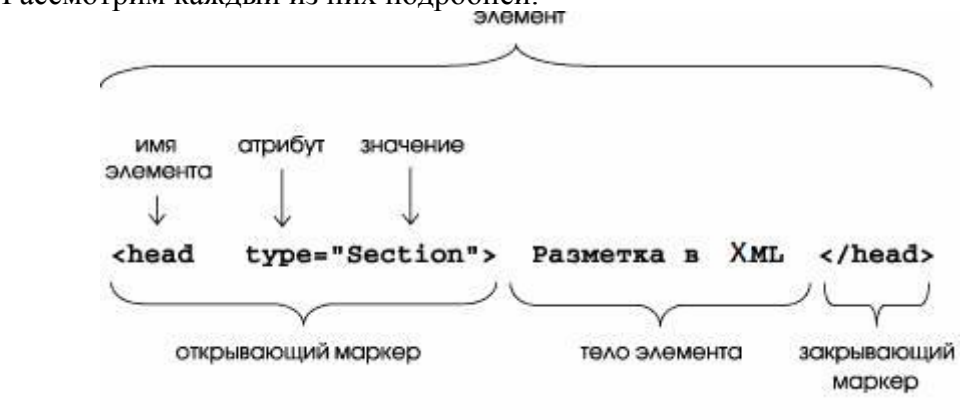
ринцип построения XML-документов. Общие синтаксические конструкции. Символьные данные и разметка. Разделы CDATA. Объявление типа документа.

В общем случае XML- документы должны удовлетворять следующим требованиям:

- В заголовке документа помещается объявление XML, в котором указывается язык разметки документа, номер его версии и дополнительная информация
- Каждый открывающий тэг, определяющий некоторую область данных в документе обязательно должен иметь своего закрывающего "напарника", т.е., в отличие от HTML, нельзя опускать закрывающие тэги
- В XML учитывается регистр символов
- Все значения атрибутов, используемых в определении тэгов, должны быть заключены в кавычки
- Вложенность тэгов в XML строго контролируется, поэтому необходимо следить за порядком следования открывающих и закрывающих тэгов
- Вся информация, располагающаяся между начальным и конечными тэгами, рассматривается в XML как данные и поэтому учитываются все символы форматирования (т.е. пробелы, переводы строк, табуляции не игнорируются, как в HTML)

Конструкции языка

Содержимое XML- документа представляет собой набор элементов, секций CDATA, директив анализатора, комментариев, спецсимволов, текстовых данных. Рассмотрим каждый из них подробней.



Пример XML-документа:

```
<conservatory>
    <flower>rose</flower>
    <flower>tulip</flower>
    <flower>cactus</flower>
</conservatory>
```

Элементы

Элемент - это структурная единица XML- документа. Заключая слово rose в в тэги <flower> </flower> , мы определяем непустой элемент, называемый <flower>, содержимым которого является *rose*. В общем случае в качестве содержимого элементов могут выступать как просто какой-то текст, так и другие, вложенные, элементы документа, секции CDATA, инструкции по обработке, комментарии, - т.е. практически любые части XML- документа.

Любой непустой элемент должен состоять из начального, конечного тэгов и данных, между ними заключенных. Например, следующие фрагменты будут являться элементами:

```
<flower>rose</flower>
<city>Novosibirsk</city>
```

Набором всех элементов, содержащихся в документе, задается его структура, и определяются все иерархическое соотношения. Плоская модель данных превращается с использованием элементов в сложную иерархическую систему с множеством возможных связей между элементами.

Производя в последствии поиск в этом документе, программа клиента будет опираться на информацию, заложенную в его структуру - используя элементы документа. То есть, если, например, требуется найти нужный университет в нужном городе, используя приведенный фрагмент документа, то необходимо будет просмотреть содержимое конкретного элемента <university>, находящегося внутри конкретного элемента <city>. Поиск при этом, естественно, будет гораздо более эффективен, чем нахождение нужной последовательности по всему документу.

В XML документе, как правило, определяется хотя бы один элемент, называемый корневым и с него программы-анализаторы начинают просмотр документа. В приведенном примере этим элементом является <country>

В некоторых случаях тэги могут изменять и уточнять семантику тех или иных фрагментов документа, по разному определяя одну и ту же информацию и тем самым предоставляя приложению-анализатору этого документа сведения о контексте использования описываемых данных. Например, прочитав фрагмент **<city>**Holliwood**</city>** мы можем догадаться, что речь в этой части документа идет о городе, а вот во фрагменте **<restaurant>**Holliwood**</restaurant>** - о забегаловке.

В случае, если элемент не имеет содержимого, то есть нет данных, которые он должен определять, то он называется пустым.Примером пустых элементов в HTML могут служить такие тэги HTML, как **
 <hr>, <img>**. Необходимо только помнить, что начальный и конечные тэги пустого элемента как бы объединяется в один, и надо обязательно ставить косую черту перед закрывающей угловой скобкой (например, **<empty/>**;)

Имена тегов и атрибутов можно писать и по-русски. Опыт HTML показал, сколь важна тщательная и своевременная интернационализация всех аспектов языка, претендующего на какую-то роль в Интернете. Поэтому создатели XML позаботились, в частности, о том, чтобы в именах тегов и атрибутов можно было пользоваться не только латинскими буквами, но и кириллицей, иероглифами и вообще всеми символами из репертуара Unicode, которые считаются "буквами" хотя бы в одном языке или системе письменности.

Лекция 7
Язык описания схемы данных XML (DTD). Способ формального описания структуры XML-документа (XSDL).

DTD (англ. *Document Type Definition* — определение типа документа) — включает в себя два понятия:

- Термин, который используется для описания схемы документа или его части *языком схем DTD*.
- *Язык схем DTD (DTD schema language)* — искусственный язык, который используется для записи фактических синтаксических правил метаязыков разметки текста SGML и XML. С момента его внедрениядругие языки схем для спецификаций, такие как XML Schema и RELAX NG, выпускаются с дополнительной функциональностью.

Из-за определённых отличий между XML и SGML, применение DTD также имеет некоторые особенности в зависимости от целевого документа

Сейчас идёт отказ от использования DTD в XML-технологии по ряду причин:

1. Используется отличный от XML синтаксис.
2. Отсутствует типизация узлов.
3. Отсутствует поддержка пространств имён.

На смену DTD пришёл стандарт консорциума W3C XML Schema.

DTD описывает схему документа для конкретного языка разметки посредством набора объявлений (объектов-параметров, элементов и атрибутов), которые описывают его класс (или тип) с точки зрения синтаксических ограничений этого документа. Также DTD может объявлять конструкции, которые всегда необходимы для определения структуры документа, но, зато, могут влиять на интерпретацию определённых документов.

Объявление объектов-параметров

Объявление объекта-параметра определяет макрос определённого типа, на который можно ссылаться и который может быть развернут где-нибудь в DTD. Эти макросы могут не появляться в самом документе, а быть только в DTD. Если на объект-параметр ссылаются по имени их DTD, то он разворачивается в строку, в которой указано содержимое этого объекта.

Объявление элементов

Объявления элементов образовывают перечень разрешенных названий элементов в документе, а также определяют информацию относительно тегов (являются ли они обязательными) и модели содержимого для каждого элемента.

Различные ключевые слова и символы определяют содержимое элемента:

- EMPTY — пустое содержимое
- ANY — любое содержимое
- , — указывает порядок
- | — разделение альтернатив
- () — группировка
- * — любое количество элементов (ноль и более)
- + — по крайней мере один элемент (один и более)
- ? — необязательное наличие элемента (ноль или один)
- Если нет *, + или ? — элемент должен быть только один

Примеры:

<!ELEMENT DL - - (DT|DD)+>

Элемент DL должен содержать один и более элементов DT или DD в произвольном порядке.

<!ELEMENT FORM - - (%block;|SCRIPT)+ -(FORM)>

Элемент FORM должен содержать в себе один или более элементов с объектом-параметром block или элементы SCRIPT в произвольном порядке, однако исключена возможность содержать ещё один элемент FORM.

Определение атрибутов

С каждым элементом DTD-документа можно сопоставить список атрибутов. Для этого используется директива!ATTLIST, в которой указываются имя элемента, с которым может быть сопоставлен список атрибутов и параметры каждого атрибута: его имя, тип и свойства по умолчанию.

Например:

<!ATTLIST MAP name CDATA #REQUIRED>

В этом примере определен атрибут name для элемента MAP. Он является обязательным.

Существуют такие типы атрибутов:

- CDATA (Character set of data) — значением атрибута могут быть любые символьные данные
- ID — значением атрибута должен быть уникальный идентификатор элемента
- IDREF — значением элемента является ссылка на элемент по его ID
- IDREFS — тоже что и IDREF, но с возможностью ссылок не по одному идентификатору, а по нескольким
- NMTOKEN — значением атрибута может быть последовательность символов, в чём-то схожая с именем (отсюда и названием — name token). Это строка,

которая содержит любую комбинацию тех символов, которые разрешено использовать для имен XML.

- NMTOKENS — значением атрибута является список значений
- ENTITY — значение используется для ссылки на внешнюю сущность.
- ENTITIES — позволяет задать список внешних сущностей, разделённых пробелами.
- NOTATION — значением атрибута может быть одна из ранее определённых нотаций
- NOTATIONS — позволяет задать список нотаций.
- Listings и NOTATION-listings
- ENUMERATION — задаёт список возможных альтернатив значений.

Существуют такие свойства по умолчанию:

1. IMPLIED — значение атрибута указывать не обязательно;
2. REQUIRED — значение атрибута обязательно должно быть указано;
3. FIXED — значение этого атрибута задано как константа в DTD и в документе не может быть изменено;
4. некоторое конкретное значение, которое используется по умолчанию.

Связь документа с определённым DTD

Чтобы связать документ с определённым DTD, необходимо в начале текста документа указать элементОбъявление Типа Документа.

В зависимости от места расположения DTD, Объявление Типа Документа может быть двух видов:

- **Внутреннее подмножество DTD**

Набор объявлений DTD содержится в самом тексте документа. Например:

```
<!DOCTYPE foo [ <!ENTITY greeting "helloworld"> ]>

<!DOCTYPE bar [ <!ENTITY greeting "helloworld"> ]>
```

- **Внешнее подмножество DTD**

Набор объявлений DTD располагается в отдельном текстовом файле с расширением .dtd В этом случае ссылку на файл можно сделать через публичный идентификатор и (или) через системный идентификатор. Например:

```
<!-- Валидация простого HTML 4.01 -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
```

Пример

Пример очень простого XML DTD, описывающего список людей:

```
<!ELEMENT people_list (person*)>
<!ELEMENT person (name, birthdate?, gender?, socialsecuritynumber?)>
<!ELEMENT name (#PCDATA) >
<!ELEMENT birthdate (#PCDATA) >
<!ELEMENT gender (#PCDATA) >
<!ELEMENT socialsecuritynumber (#PCDATA) >
```

Начиная с первой строки:

1. Элемент `<people_list>` содержит любое число элементов `<person>` . Знак `<*>` означает что возможно 0, 1 или более элементов `<person>` внутри элемента `<people_list>`.
2. Элемент `<person>` содержит элементы `<name>`, `<birthdate>`, `<gender>` и `<socialsecuritynumber>`. Знак `<?>` означает что элемент необязателен. Элемент `<name>` не содержит `<?>`, что означает что элемент `<person>` *обязательно должен* содержать элемент `<name>`.
3. Элемент `<name>` содержит данные.
4. Элемент `<birthdate>` содержит данные.
5. Элемент `<gender>` содержит данные.

6. Элемент <socialsecuritynumber> содержит данные.

Пример XML-документа, использующего этот DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE people_list SYSTEM "example.dtd">
<people_list>
  <person>
    <name>
      Fred Bloggs
    </name>
    <birthdate>
      27/11/2008
    </birthdate>
    <gender>
      Male
    </gender>
    <socialsecuritynumber>
      1234567890
    </socialsecuritynumber>
  </person>
</people_list>
```

Лекция 8
Интеграция XML с корпоративными бизнес-моделями. Электронная коммерция и XML

Разработка XML началась в 1996 году, и с февраля 1998 года язык XML является стандартом W3C. Язык XML определяет стандарты построения Интернет-приложений класса "предприятие-предприятие". XML является платформой, идущей на смену разрозненным стандартам и диалектам-это платформа для “самоописания“ информации, универсальный язык форматирования структурированных документов и данных.. XML позволяет представлять сложные, иерархические объекты в текстовом формате. Если язык разметки гипертекста HTML определяет, как элементы будут расположены на Web-сайте, спецификации HTML позволяют лишь форматировать текст, то XML определяет, что эти элементы будут содержать, XML обеспечивает создание собственных дескрипторов, помогающих идентифицировать объекты. Т.е. XML от HTML отличается тем, что позволяет определять (создавать) собственные элементы, адаптированные под специфические нужды компании или какой-либо отрасли. С помощью XML можно описать данные практически любой предмерной области. Язык XML используется для создания документов, управления, хранения и передачи информации, в том числе и в сети Интернет. XML-документ не зависит от операционной системы и может создаваться при помощи различных языков программирования. Данные из прикладной системы любого типа можно выгрузить в виде XML-документов, используя встроенный в нее язык программирования, даже если он не располагает специальной библиотекой поддержки XML.

Технология XML предоставляет возможность универсального доступа к данным и используется для обмена информацией в бизнесе:

- обмена данными между различными бизнес-приложениями;
- обмена данными с удаленными филиалами предприятия/банка;
- обмена данными между разными организациями;
- обмена данными между БД и Интернет-приложением.

XML позволяет перевести разнородные данные из каталогов, счетов, заказов в единую форму и совершать операции над ними автоматически, без участия человека. Интернет начинает использоваться для обмена всеми документами в реальном времени - от заказов на покупку до подтверждающих квитанций. Заказы в реальном времени по стандарту XML пришли на смену автоматизированному приему документов по факсу.

Язык XML позволяет компаниям автоматизировать операции, связывать деловые процессы и строить системы по платному предоставлению информационных услуг.

Язык XML позволяет перейти действительно к информационному обмену между сайтами, сайтами и клиентами, он дает возможность компьютерным системам обмениваться сведениями о формах заказов, ценах и уровнях складских запасов, существенно повышая эффективность и снижая затраты. Описание данных средствами XML позволяет автоматически публиковать дубли информационных блоков как на общекорпоративном сайте, так и на web-странице торговой системы с учетом принятых на обоих ресурсах оформления.

В XML можно интегрировать потоковую и текстовую информацию. Это способ писать приложения так, чтобы программы могли легко воспринимать и рукописный текст и человеческую речь. XML открывает путь к совершенно новому классу Интернет-услуг. На основе стандарта XML происходит революция в способе общения компьютеров друг с другом.

При организации электронной торговли между юридическими лицами самым сложным технологическим барьером, пожалуй, является интеграция унаследованных систем. XML в настоящий момент является наиболее удачной базой для объединения программ разных разработчиков. Применение средств XML позволяет навести мосты между новейшими системами электронной коммерции и системами поставщиков и их клиентов и осуществлять интеграцию гораздо быстрее и независимо от тех систем или платформ, на основе которых построен интерфейс.

XML легко встраивается в HTML - страницу. XML-данные распознаются XML-анализатором а затем интепретируются с помощью JavaScript или VBScript(обрабатываются, отображаются, пересылаются на др. сайты). Новые версии Интернет-серверов поддерживают язык XML. XML- анализаторы присутствуют в последних версиях браузеров от Microsoft и Netscape и скоро станут стандартом любых ОС.

Специалисты исследовательской компании The Intellor Group сообщают, что 42% предприятий электронного бизнеса во всем мире считают основным преимуществом использования языка XML то, что он является общим стандартом для всего B2B-сообщества. Сегодня консорциум World Wide Web (W3C) призывает компании и учебные заведения переходить на использование XML, который позволяет сделать информацию доступной для разных устройств, включая персональные компьютеры, ноутбуки, мобильные телефоны и т.д.

В феврале 2002г. Консорциум World Wide Web Consortium (W3C) объявил о создании единого языкового стандарта цифровой XML-подписи (XML Signature). Возможность подписывать XML-сообщения с помощью XML-подписи является крайне важной в силу того, что этот язык является основой Web-сервисов. XML Signature сможет использоваться с различными средствами разработки, в том числе и с Visual Studio .NET от Microsoft. Кроме того, XML Signature позволит подписывать документ более чем одному человеку, таким образом, у документа может быть несколько владельцев и он может состоять из множества подсекций. Это важно в том случае, если один и тот же документ проходит через руки нескольких сторон. Каждая из уполномоченных сторон сможет прочитать документ, внести в него необходимые коррективы, подписаться под ними и направить другой стороне. С коммерческой точки зрения, такая технология может быть с успехом использована при многосторонних сделках. В данном случае каждый из контрагентов сможет подписать только ту часть документа, которую составлял он сам. Кроме того, новая норма применяется не только к документам, но также и к графическим файлам различных форматов, включая BMP и JPG.

Многие надеются, что XML полностью вытеснит EDI, Сегодня компании пытаются внедрить новые подходы, при которых уже существующие данные формата EDI конвертируются в XML.

В России при реализации федеральной целевой программы "Электронная Россия", одной из задач является объединение государственных органов власти в единую систему электронного документооборота. По сути необходимо объединить в единое

информационное пространство унаследованные системы электронного документооборота, построенные на различных платформах и использующие различные форматы данных. Принятый закон об электронно-цифровой подписи отчасти решил эту проблему. Использование языка XML - один из наиболее перспективных вариантов, применение которого позволит перейти на единую систему электронного документооборота, позволит существенно повысить эффективность контактов организаций со своими партнерами, удаленными филиалами, государственными органами и другими организациями.

Существует более 350 диалектов XML, не все они соответсвуют стандарту W3C в должной мере. Наиболее известные:

- BizTackFramework (компания Microsoft)
- cXML (компания Ariba)
- xCBL (XML Common Business Language)от Commerce One
- ebXML(electronic business XML)- представляет собой набор основанных на XML спецификаций, предоставляющий возможность создавать модульные сети электронной коммерции. IBM предложила ebXML в 2000г. Стандарт ebXML расширялся и продвигался Центром по упрощению торговли при ООН (UN/CEFACT) и консорциумом высокотехнологических компаний OASIS-Organisation for the Advancement of Structured Information Standards(Организация за развитие стандартов структурирования информации), куда входят IBM, Sun Microsystems, Hewlett-Packard и др. В мае 2001г. в Вене была проведена первая демонстрация использования ebXML для онлайновой торговли и обмена данными между деловыми партнерами, банками и торговыми площадками(обмен и маршрутизация документов, загрузка и выгрузка каталогов товаров, ввод заказов). Более 200 организаций, имеющих отношение к электронному бизнесу, одобрили внедрение языка ebXML. Поддержку ebXML включили в свои продукты компании Fujitsu, IONA, Oracle, Sun, Sybase, webMethods. Корпорация IBM не намерена взимать лицензионную оплату по своим патентам, которые лежат в основе стандарта ebXML.

VoiceXML - язык разметки голосовых диалогов. Используется в голосовых порталах для управления голосом через Интернет. Становится общепринятым стандартом для доступа к речевой информации

SOAP(Simple Object Access Protocol) - "Протокол простого доступа к объектам"- расширение XML, описывающее принципы онлайнового взаимодействия приложений. SOAP упрощает взаимодействие между программами и платформами и делает приложения доступными для поставщиков и партнеров. Таким образом, SOAP позволяет предприятиям совместно пользоваться деловой информацией в режиме реального времени, участвовать в бизнес-стратегиях партнеров по глобальным альянсам или интегрировать системы покупок/заказов .

GEML(Genetic Expression Murkup Language) - расширение XML для обмена большими объемами генетической информации.

XMCL- (eXtensible Media Commerce Language)-основанный на XML язык для описания бизнес-правил в сфере онлайновой торговли медиа-продукцией. С инициативой создания XMCL в июне 2001г. выступила компания RealNetworks. XMCL должен стать основой онлайновых бизнес-процессов, связанных с покупкой, арендой, подпиской, передачей прав, видео-по-запросу и прочими видами деятельности, характерными для индустрии развлечений. За счет разработки стандартного языка для описания бизнес-правил, планируется добиться независимости от технологий кодирования, систем управления авторскими правами и приложений электронной коммерции. XMCL должен кардинально упростить доставку медиа-продукции и ускорить развитие рынка электронной коммерции в этой отрасли. RealNetworks намерена подать XMCL на утверждение в соответствующие органы по стандартизации и убедить лидеров отрасли принять этот стандарт. www.xmcl.org

PSML (Provisioning Services Markup Language)- язык разметки сервисов снабжения. Новый язык, разработка инициирована консорциумом OASIS. Спецификация нового

языка представляет собой правила внедрения автоматизации разграничения прав пользовательского или административного доступа к электронным сервисам. Снабжение становится ключевым фактором в функционировании практически всех веб-сервисов от ASP-услуг до товарно-сырьевых бирж. В подобных условиях возникает спрос на стандартизированный подход к организации процесса снабжения в рамках отдельных сервисов, так как сегодня на системных администраторов ложится слишком большая ответственность при установке ограничений прав доступа.

BPML (Business Process Modelling Language)- расширение XML. Определяет формальную модель описания бизнес-процессов, представляющих широкий диапазон различных видов и типов деятельности предприятия, в том числе обработку транзакций, управление данными, конкурентную борьбу, действия в исключительных ситуациях и т.п.

Сайты языка XML

www.w3.org/2000/xp/ - XML Protocol

www.w3.org/Encryption/2001/ - XML Encryption

www.xml.com

http://members.xoom.com/xmlsite

www.xml.apache.org

www.xmlhack.com

www.TopXML.com

www.raleigh.ru - Новости в мире XML

статья "XML в 10 тезисах" Б.Бос (W3C)

"Платформа XML" на портале Российские электронные библиотеки www.elbib.ru

Базы данных – сфера успешного применения XML

Согласно отчету IDC, поставщики XML-ориентированных систем управления базами данных, такие, например, как немецкая фирма Software AG, которые давно и успешно применяют XML, в следующие несколько лет могут ожидать бурного роста спроса на свои системы. Бурное развитие этого сектора обусловлено рядом фактором Во-первых, XML-серверы будут играть все более важную роль, так как они упрощают интеграцию приложений для е-бизнеса. Во-вторых, у производителей СУБД с возможностью хранения XML-данных множество лояльных к XML клиентов, которые пропагандируют распространение и использование XML.
"The XML Academy" компании Software AG включает два набора курсов: по стандартам в области XML-технологий и по продуктам компании Software AG, поддерживающим эти технологии. Академия АйТи в июле 2002 присоединилась к инициативе компании Software AG по обучению технологиям XML.

"Xperanto для баз данных" на сервере "Открытые системы"(www.osp.ru) Корпорация IBM работает над проектом, получившим кодовое название Xperanto, в рамках которого будет создана СУБД со встроенной поддержкой XML. СУБД Xperanto, которая будет выполнять функции подмножества DB2 и опираться на использование самого XML и языка запросов XQL, станет критически важным компонентом долговременной стратегии IBM, направленной на создание решений, позволяющих объединить обработку структурированных и неструктурированных данных. XML даст возможность работать с самыми разнообразными типами данных; он предлагает своего рода структуру для неструктурированных данных. Тем самым обеспечиваются доступ к максимально широкому спектру наборов данных внутри организации, таких как файлы MS Office. Цель-предоставить пользователям возможность эффективнее использовать накопленные в их организациях знания. Организации смогут обращаться к информационному наполнению, содержащемуся в файлах Word, которые находятся в настольных системах отдельных пользователей. Широкое распространение XML делает реализацию идеи о сочетании структурированных и неструктурированных данных вполне реальной. Корпорация IBM окончательно оформить данную технологию к концу 2002г.

Примеры использования XML

Бизнес-подразделение компании General Electric -производитель технической термопластмассы фирма GE Plastics использует электронно-коммерческое технологическое решение, предоставленное службой GE Global eXchange Services, для

проведения онлайновых транзакций на основе стандарта XML со своими клиентами в США, Европе и странах тихоокеанского региона. Клиенты GE Plastics используют Интернет для обмена всеми документами в реальном времени - от заказов на покупку до подтверждающих квитанций.Заказы в реальном времени по стандарту XML пришли на смену автоматизированному приему документов по факсу, когда на оформление одного заказа требовалось не менее 48 часов и все данные вводились вручную. Электронные проекты GE Global eXchange Services и FedEx e-Commerce Network заняли первое место в конкурсе издания eAI Journal "Best e-Business Solution".

OpenOffice -полноценная свободная офисная система с открытыми исходными текстами. Формат файлов является открытым и основан на стандарте XML. Это позволяет применять его также и в других программах, причём с использованием имеющихся обширных наработок для работы с XML. По мнению некоторых экспертов, для специалистов наличие полноценного открытого формата для офисных документов даже важнее, чем собственно доступность OpenOffice.

«Интернет-пейджер» Jabber, основан на стандарте XML. Он успешно развивается как альтернатива закрытым сервисам типа ICQ и MSN Messenger. В 2002 году эта система стала широко известной и массово применяемой.

Interwoven и XML. www.intel.ru Как утверждают специалисты Interwoven, XML открывает эпоху универсального обмена данными. Отделяя презентационную часть данных от содержательной, XML способствует свободному взаимодействию между различными системами.

По утверждению Interwoven, преимущества XML таковы:

- Простота обмена данными: XML обеспечивает преимуществами электронной коммерции компании, которые раньше считались недостаточно крупными. Создание взаимосвязей между отдельными компаниями приведет к образованию новых деловых контактов.
- Универсальность: язык XML, позволяющий отделить презентационную часть данных от содержательной, обеспечивает возможность многофункционального использования содержательной части. Кроме того, один и тот же код XML можно использовать для передачи таких данных, как, например, информация о ценах на ПК и карманные компьютеры.
- Скорость: XML сделает возможным быстрое создание эффективных и содержательных Web-сайтов. Особенно важно то, что XML сократит время разработки приложений для делового сотрудничества.

Простые решения для повышения производительности: www.intel.ru использование XML для автоматизации сбора информации, поступающей от поставщиков. Корпорацией Intel(R) внедрена система документооборота, основанная на использовании XML и призванная улучшить взаимодействие между производственными подразделениями корпорации и ее основными поставщиками химических веществ и топлива. Для формирования учетных записей и защиты данных корпорация Intel использовала серверы Business Document Exchange* (BDE) и клиентские приложения на базе Java-технологий. Intel разработала XML-сертификат качества простого формата. Кроме того, она предоставила спецификацииXML-сертификата, а также инструкции по подключению к серверу BDE. Intel использовала основанную на XML систему документооборота для обмена сертификатами качества.Данная система позволила снизить эксплуатационные расходы, сократить время обработки данных и исключить возможность возникновения ошибок. *

Лекция 9
Динамические Web-страницы. (JavaScript, VBScript, Java-апплеты, ActivX-объекты и т.п.)

Изо дня в день работая над обновлением содержимого своего Web-сайта, насыщая его интересными материалами, вы, вероятно, задумываетесь о том, что ежедневно создаются сотни новых Web-сайтов, которые также ежедневно пополняются сотнями новых документов. Как создаются все эти новые массивы страниц и каким образом они

так быстро обновляются? Все это не так сложно, как кажется на первый взгляд, поскольку здесь используется концепция динамических Web-страниц.

В этой статье мы рассмотрим этапы создания механизма публикации на Web-сайте пресс-релизов. Наш сайт будет соединять «на лету» пресс-релизы, хранящиеся в базе данных, с шаблонными Web-страницами. Мы не ставили целью ознакомить читателей с основами средств разработки Web-сайтов, поскольку об этом написано множество книг и статей. Данная статья предназначена в основном для тех пользователей, которые уже имеют опыт создания Web-страниц и простых сайтов. Наша главная цель — показать, как начать разрабатывать свой первый динамический Web-сайт. Для понимания статьи желательно иметь базовые знания об архитектурах информационных систем, о языке разметки гипертекста (HTML) и языке программирования Perl . Для создания этого сайта мы воспользуемся тремя мощными открытыми технологиями: Apache, MySQL и Perl/DBI.

Что такое статический Web-сайт?

Перед тем, как погрузиться в разработку динамического Web-сайта, важно понять, что представляют собой статический Web-сайт и статические Web-страницы, составляющие его основу. Статические Web-страницы создаются вручную, потом сохраняются и загружаются на сайт. Всякий раз, когда требуется изменить содержимое такой страницы, пользователь модифицирует ее на своем рабочем компьютере, применяя, как правило, HTML-редактор, сохраняет ее и затем заново загружает на Web-сайт. Внимательно присмотревшись к какому-нибудь порталу, допустим к CNN.com или BBC.co.uk, можно подумать, что для обновления содержимого своих сайтов эти компании привлекают армию верстальщиков. На самом же деле существует лучший способ — использование концепции динамического Web-сайта.

Что такое динамический Web-сайт?

Каждая отображаемая страница динамических Web-сайтов основана на шаблонной странице, в которую вставляется постоянно меняющееся информационное наполнение, которое обычно хранится в базе данных. Когда пользователь запрашивает страницу, соответствующая информация извлекается из базы, вставляется в шаблон, образуя новую Web-страницу, и пересылается Web-сервером в пользовательский браузер, который и отображает ее должным образом. Кроме информационного наполнения, динамически могут создаваться также и элементы навигации по Web-сайту. Таким образом, если вам нужно обновить содержимое своего сайта, вы просто добавляете текст для новой страницы, который затем вставляется в базу данных с помощью определенного механизма. В результате получается, что Web-сайт как бы сам себя обновляет.

Почему динамические сайты лучше

Сразу после того как динамический сайт создан и запущен в работу, начинают проявляться его преимущества. Теперь в вашем распоряжении имеется сравнительно небольшое количество шаблонных страниц, с помощью которых генерируются сотни, а может быть, и тысячи Web-страниц. Вид (дизайн) сайта может быть легко изменен с помощью модификации этих шаблонов. Изменение содержимого базы данных можно производить через Web-интерфейс с использованием HTML-формы, не вторгаясь при этом в технические детали каждой специфической СУБД.

Создание динамического сайта

Первое, что нужно для создания динамического сайта, — это Web-сервер, например Apache.

Web-сервер может использоваться для обслуживания электронного магазина, сервера новостей, поискового механизма, системы дистанционного обучения и даже для всей совокупности перечисленных сфер. Выбор Web-сервера зависит от того, каким видом деятельности частное лицо или организация собирается заниматься в Интернете.

Немногие из принимаемых в бизнесе стратегических решений столь же значимы, как выбор платформы для Web-сервера. Характеристики сервера — это чрезвычайно важный фактор, определяющий надежность узла, его «отзывчивость» на запросы клиентов, а также то, какие усилия необходимо предпринимать для поддержания его в рабочем состоянии. При правильном выборе компонентов и качественном проекте Web-узел может стать для клиентов и партнеров новым, более удобным способом

взаимодействия с вашей компанией. Перегрузка Web-сервера может привести к тому, что сервер баз данных или какой-либо иной ресурс станет недоступным для клиентов.

Крупные компании до недавнего времени делали ставки на Microsoft Internet Information Server, Netscape FastTrack, IBM WebSphere, а Apache в основном использовался небольшими компаниями. Однако сейчас ситуация несколько изменилась, и Apache начинает поддерживать работоспособность некоторых крупных Интернет-проектов, в частности Yahoo.

Apache предоставляет богатые возможности, позволяющие настроить Web-сервер в соответствии с потребностями индивидуальных и корпоративных пользователей. Настройка производится с помощью директив, содержащихся в конфигурационных файлах. Apache позволяет создавать виртуальные Web-узлы, а также выполняет функции proxy-сервера. Если нужно предоставить доступ к содержимому сервера лишь ограниченному кругу лиц, Web-сервер можно настроить так, чтобы при обращении к указанным каталогам сервер проверял регистрационные имена и пароли в собственной или в одной из подключенных к нему баз данных.

Далее вам нужно решить, как вы собираетесь хранить информационное наполнение (контент), которое отображается на Web-странице. В данной статье на конкретном примере мы покажем, как создать базу данных в СУБД MySQL, которая позволит нам разбить Web-контент на таблицы, содержащие поля и записи с данными. Поле — это дискретная единица данных в таблице. Например, мы можем создать таблицу tbl_news_items с полями col_title, col_date, col_fullstory, col_author. СУБД MySQL — отличный выбор для создания такой базы данных вследствие простоты в использовании и администрировании, свободной распространяемости для разных платформ, включая Linux и Windows, и быстро растущей популярности.

JavaScript позволяет разрабатывать различные динамические эффекты, которые могут преобразовать внешний вид сайта, обеспечить простоту и удобство навигации по сайту, сократить время, количество движений и кликов пользователю сайта. В общем, может существенно преобразовать сайт в лучшую сторону.
Для того чтобы сделать сайт на JavaScript реагирующий на действия пользователя, нужно как-то в сценариях JavaScript следить за действиями пользователя. Отслеживать действия пользователя позволяют обработчики событий. Обработчики событий следят за действиями пользователя, и при возникновении какого-либо события, соответствующий обработчик события вызывает привязанный к нему код JavaScript. Например, есть кнопка в форме HTML-документа и при клике пользователем на эту кнопку браузер вызовет обработчик события, отвечающий за клик по элементу страницы - **onClick**, и если к этому обработчику события привязан код JavaScript, то он будет выполнен.
Для наглядности давайте выполним очень интересный и нужный эффект: при клике пользователем по определенному тексту, ниже будет появляться какой-либо текст (например, подробное описание или дополнительная информация).
Как уже я говорил выше, за щелчок по элементу веб-страницы следит обработчик события **onClick**. Проще всего его привязать прямо к необходимому тегу, указав его в качестве атрибута и передав ему код JavaScript, который должен быть выполнен при вызове этого обработчика. Обработчики событий можно указывать большинству тегов, мы выберем тег **SPAN**:

Код

```
<span onClick="clickText()">Кликните по мне</span>
```

В этом примере обработчику события передана функция, хотя в общем случае можно передавать любой код JavaScript. Функцию мы напишем далее. Текст, который должен появиться будет находиться в блоке **div** и сделаем его первоначально невидимым. Для этого воспользуемся стилевым свойством **display**, которому зададим значение **none**. При клике по тексту заключенному в теге **SPAN**, будем менять значение свойства**display**: либо **none** на **block**, либо **block** на **none**. На данном этапе получим

Код

```
<span onClick="clickText()">Кликните по мне</span>
<div id="info_text" style="display:none; ">Это поясняющий текст к выше расположенному.</div>
```

С помощью идентификатора *id="info_text"* в последующем найдем нужный нам блок **div**. Теперь можно написать функцию *clickText()*

Код

```
function clickText()
{
obj_div=document.getElementById("info_text");
if (obj_div.style.display=="block")
obj_div.style.display="none";
else
obj_div.style.display="block";
}
```

В функции вроде все просто, только прокомментирую, что в переменную *obj_div* помещается объект блока **div**, со свойствами которого и работаем. Теперь можно написать результирующий код, добавив немного оформления:

Код

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<title>Поясняющий текст</title>
</head>
<body>
<div style="border-bottom:dashed 1px #000;">
<span onClick="clickText()" style="cursor:pointer; font-size:110%;">Кликните по мне</span>
</div>
<div id="info_text" style="display:none; background:#a1ecff;">Это поясняющий текст к выше расположенному.</div>
<script type="text/javascript">
function clickText()
{
obj_div=document.getElementById("info_text");
if (obj_div.style.display=="block")
obj_div.style.display="none";
else
obj_div.style.display="block";
}
</script>
</body>
</html>
```

Если после загрузки страницы кликнуть по тексту расположенному в теге **SPAN**, то появиться дополнительный текст. При последующих кликах, дополнительный текст будет скрываться и заново появляться.

Реакция на наведение курсора мыши

Для того чтобы реализовать реакцию сценария JavaScript на наведение на какой-либо элемент HTML-документа, нужно воспользоваться обработчиками событий:

- **onMouseOver** - обрабатывает событие: пользователь передвинул указатель мыши на элемент;

- **onMouseOut** - обрабатывает событие: пользователь убрал указатель мыши с элемента.

Для иллюстрации возможностей данных обработчиков событий реализуем изменение цвета фона и текста блока при наведении на него курсора мыши. Для изменения цвета при наведении курсора мыши на элемент воспользуемся обработчиком событий **onMouseOver**:

Код

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<title>Изменение цвета текста и фона при наведении курсора на элемент</title>
</head>
<body>
<div onMouseOver="this.style.color='#fff' ;this.style.background='#a2d' ">При наведении указателя мыши на этот текст измениться цвет фона и текста</div>
</body>
</html>
```

При просмотре данного кода в браузере попробуйте навести курсор мыши на текст: цвет фона и текста изменится, но после увода курсора цвет не изменится на первоначальный. Поэтому нужно добавить еще обработчик событий **onMouseOut**, которому поручим вернуть цвета в исходное значение.

Код

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<title>Изменение цвета текста и фона при наведении курсора на элемент</title>
</head>
<body>
<div onMouseOver="this.style.color='#fff' ;this.style.background='#a2d' " onMouseOut="this.style.color='#000' ;this.style.background='#fff' " >При наведении указателя мыши на этот текст измениться цвет фона и текста</div>
</body>
</html>
```

В этом примере мы использовали оператор **this**, который содержит объект текущего элемента. В нашем случае оператор **this** содержит объект блока **div**, поэтому мы смело использовали его для изменения свойств. Использование оператора **this** значительно облегчает написание скриптов, так как получить доступ к объекту не составляет особого труда, делает код компактнее. Сравните с предыдущим примером, где для получения объекта необходимого блока **div** использовали конструкцию *document.getElementById(идентификатор)*, предварительно указав идентификатор в атрибуте **Id** для нужного блока **div**.

Динамическое отображение текста страницы

Есть еще большое количество обработчиков событий, о которых мы еще поговорим на страницах этого сайта, теперь давайте рассмотрим способы изменения содержимого (текста) страницы. Для подобных целей удобно использовать свойства элемента **innerHTML** и **innerText**. Эти два свойства доступны как для чтения, так и для записи. Рассмотрим работу с ними подробнее на следующем примере: пусть при наведении курсора мыши на блок с текстом, текст меняется. Для отслеживания событий наведение курсора на элемент и увода с элемента будем использовать уже знакомые нам обработчики событий**onMouseOver** и **onMouseOut**.

Код

```
<div onMouseOver="newText(this)" onMouseOut="backText(this)">Наведите на текст курсор мыши</div>
```

При наведении курсора мыши на блок будет выполняться функция *newText()*, которая заменит текст в блоке и некоторые свойства. При уводе курсора с элемента будет вызываться функция, которая отменит все изменения.

Код

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<title></title>
</head>
<body>
<div onMouseOver="newText(this)" onMouseOut="backText(this)">Наведите на текст курсор мыши</div>
<script type="text/javascript">
function newText(obj_div)
{
obj_div.innerHTML="Вы что наделали? :) Уберите немедленно курсор мыши с этого текста! :)";
obj_div.style.background="#FFA8A8";
}
function backText(obj_div)
{
obj_div.innerHTML=" Наведите на текст курсор мыши ";
obj_div.style.background="#fff";
}
</script>
</body>
</html>
```

Изменить текст в элементе, можно изменив значение свойства **innerHTML** или **innerText** соответствующего объекта. Если надо определить объект по идентификатору, то можно записать следующим образом

Код

```
<div id="blockText">Тут будет новый текст</div>
<script type="text/javascript">
document.getElementById("blockText").innerHTML="<b>Это новый текст!</b>";
</script>
```

Свойства **innerHTML** отличается от **innerText**, тем, что при использовании свойства **innerHTML** можно использовать теги HTML (то есть будет текст форматироваться тегами при выводе на экран), а при использовании **innerText** теги будут отображены как обычный текст (и естественно текст не будет форматироваться тегами). Но также нужно учитывать, что свойство **innerText** не поддерживается браузером Mozilla Firefox.

С помощью рассмотренных нами простых способов создания динамических эффектов, можно создавать более сложные манипуляции и эффекты с содержимым страницы.

Лекция 10
Создание сценариев. Основы языка программирования. (JavaScript, VBScript, Java-апплеты, ActivX-объекты и т.п.)

JavaScript - язык программирования предназначенный для создания интерактивных HTML-документов с использованием сценариев. С помощью сценариев поддерживается диалог с пользователем, обеспечивается привлекательный вид web-страниц, осуществляется навигация по страницам сайта, поиск элементов на странице и многое другое. Основой языка является понятие объекта (каждый элемент с которым работает JavaScript представляет собой объект). JavaScript внедрен в HTML (т.е. работает только совместно с HTML), обеспечивает работу в среде, поддерживаемой браузерами. Язык JavaScript позволяет обрабатывать исходные данные, представленные с помощью различных элементов управления, создавать тестирующие программы, осуществлять контроль вводимых данных и многое другое.
Программа (сценарий) на языке JavaScript представляет собой последовательность операторов, которые обрабатываются встроенным в браузер интерпретатором. Надо стремиться к тому, чтобы написанные сценарии корректно выполнялись в любом браузере. На первоначальном этапе обучения удовлетворить этому требованию сложно. Поэтому предлагаемые сценарии отлаживались в Internet Explorer версии 4.01 и выше.

Как поместить сценарий в документ

Сценарии, написанные на языке JavaScript, могут располагаться непосредственно в HTML-документе между тегами <script> и </ script>. Эти тэги означают, что в документ помещен сценарий JavaScript.Одним из параметров тега <script> является параметр language, он определяет используемый язык сценариев. Для языка JavaScript значение параметра равно "JavaScript". Если используется язык сценариев VBScript, то значение параметра должно быть равным "VBScript". В случае использования языка JavaScript параметр language можно опускать, так как этот язык используется браузером по умолчанию.
Обычно браузеры, не поддерживающие какие-либо тега HTML, их просто игнорируют. Попытка браузера проанализировать содержимое неподдержанных тегов может привести к неверному отображению страницы. Чтобы избежать такой ситуации, рекомендуется помещать операторы языка JavaScript в теги комментария <!--...->. Дли правильной работы интерпретатора перед закрывающим тегом комментария -> следует поставить символы //. Итак, для размещения сценария в HTML-документе следует написать следующее:

```
<script language= "JavaScript">
<! -
операторы языка JavaScript.
//->
</script>
```

Документ может содержать несколько тегов <script>. Все они последовательно обрабатываются интерпретатором JavaScript. Тег <noscript> определяет HTML-код, отображаемый на экране в случае, если JavaScript не поддерживается браузером или поддержка отключена. Этот тег следует после кода, заключенного в теги <script> и </script>. Если поддержка включена, то тег<noscript> игнорируется. В дальнейших примерах будем считать, что поддержка JavaScript включена. В программах на JavaScript можно использовать комментарии. Для того чтобы задать комментарий, располагающийся на одной строке, достаточно перед текстом комментария поставить две косые черты. Если же поясняющий текст занимает несколько строк, то его следует заключать между символами /* и */. В JavaScript строчные и прописные буквы алфавита считаются разными символами.

Пример 1. Этот сценарий познакомит вас с самыми основами создания и размещения JavaScript на веб-странице:

```
<SCRIPT LANGUAGE="javascript">
document.write("<FONT COLOR='RED'>Это красный текст</FONT>")
</SCRIPT>
```

Наш первый сценарий помещает текст на веб-страницу. В данном случае текст будет красного цвета.

Задание 1. Поместите этот сценарий в простейший Web-документ. Откройте полученный файл в броузере. Измените сценарий так, чтобы вышли две строки текста, красная и синяя.

Функции

Основным элементом языка JavaScript является функция. Описание функции имеет вид: function F (V) {S}, где F - идентификатор функции, задающий имя, по которому можно обращаться к функции, V - список параметров функции, разделяемых запятыми, S - тело функции, в нем задаются действия, которые нужно выполнить, чтобы получить результат. Необязательный параметр return e определяет возвращаемое функцией значение.
Описание функции не может быть вложено в описание другой функции. Параметры функции внутри ее тела играют ту же роль, что и обычные переменные, но начальные значения этим параметрам задаются при обращении к функции. Если описание функции имеет вид function F <v1,v2, . . . ,vn {S}, то вызов функции должен иметь вид: F (e1,e2,. .. ,en), где e1,e2,. . . ,en -выражения, задающие фактические значения параметров. Параметры v1,v2,. . . ,vn, указанные в описании функции, называются формальными параметрами, чтобы подчеркнуть тот факт, что они получают смысл только после задания в вызове функции фактических параметров e1, e2,. . ., en, с которыми функция затем и работает. Бели в функции параметры отсутствуют, то есть описание функции имеет вид function F {S}, то наличие скобок в операторе вызова функции обязательно, то есть вызов функции в этом случае должен иметь вид: F().
Обычно все определения и функции задаются в разделе <HEAD> документа. Это обеспечивает интерпретацию и сохранение в памяти всех функций при загрузке документа в браузер.

Пример 2. Этот сценарий показывает пример использования функций:

```
<HTML>
<HEAD>
<TITLE>Пример использования функций</TITLE>
<SCRIPT LANGUAGE="javascript">
<!--
function dateinbar() // описываем функцию вывода даты
{
var d = new Date(); // получаем системную дату
var y = d.getFullYear(); // выделяем год из даты
var da = d.getDate(); //  выделяем число из даты
var m = d.getMonth() + 1;  // выделяем месяц из даты
var t = da + '/' + m + '/' + y; // формируем информацию для вывода
defaultStatus = "Вы прибыли на страницу " + t + "."; // выводим готовую дату
}
// -->
</SCRIPT>
</HEAD>
<BODY BGCOLOR="white" onLoad="dateinbar()"> // вызываем функцию вывода даты
<p> Дата вашего прихода на страницу </p>
</BODY>
</HTML>
```

В результате действия сценария в строке состояния отображается дата открытия пользователем Web-страницы.

Задание 2. Создайте Web-страницу, используя описанный выше код. Откройте полученный файл в броузере. Измените сценарий так, чтобы сначала указывался год, затем месяц, а затем день посещения Web-страницы.

Литералы

Простейшие данные, с которыми может оперировать программа, называются литералами. Литералы не могут изменяться. Литералы целого типа могут быть заданы в десятичном (но основанию 10), шестнадцатеричном (по основанию 16) или восьмеричном (по основанию 8) представлении. Литерал целого типа в десятичном представлении записывается как последовательность десятичных цифр со знаком или без него, например, 15, 123, -156, +3567.

Кроме целых и вещественных значений, в языке JavaScript могут встречаться так называемые логические значения. Существуют только два логических значения: истина и ложь. Первое представляется Литералом true, второе - false. В некоторых реализациях JavaScript может быть использована единица в качестве true и ноль в качестве false. Строковый литерал представляется последовательностью символов, заключенной в одинарные или двойные кавычки. Примером строкового литерала может быть строка "результат" или'результат'.

Переменные

Переменные используются для хранения данных. Переменные в сценарии представляются с помощью идентификаторов. Идентификатор должен начинаться с буквы латинского алфавита, либо с символа подчеркивания. Далее может следовать последовательность, содержащая буквы латинского алфавита, цифры или знак подчеркивания, например, d, y, testl, _my_test, test_l. (см. код примера 2).Тип переменных зависит от хранимых в них данных. При изменении типа данных меняется тип переменной. Определить переменную можно с помощью оператора var, например, var testl. Тип переменной tesl1 не определен. Тип переменной станет известен только после присвоения неременной некоторого значения. Значение переменной изменяется в результате выполнения оператора присваивания. Оператор присваивания может быть использован в любом месте программы. Он может изменить не только значение, но и тип переменной. Оператор присваивания выглядит так a=b, где a - переменная, которой мы хотим задать некоторое значение, b - выражение, которое определяет новое значение переменной.

Переменные, описанные в сценарии как в части <HEAD>, так и в части <BODY>, имеют одну и ту же область действия, доступны любому сценарию текущего документа. Такие переменные называются глобальными, в отличие от локальных переменных, определенных в теле функции.

Выражения

Выражения строятся из литералов, переменных, знаков операций, скобок. В результате вычисления выражения получается единственное значение, которое может быть либо числом (целым или вещественным), строкой, либо логическим значением. Используемые в выражении переменные должны быть инициализированы. Если при вычислении выражения встречается неопределенная или неинициализированная переменная, то фиксируется ошибка. В JavaScript определен литерал null для обозначения неопределенного значения. Если переменной присвоено значение null, то она считается инициализированной.

Обработчики событий

Интерактивные документы создаются с помощью форм. Действие пользователя (например, щелчок кнопкой мыши) вызывает событие, которое производится, в основном, с элементами форм HTML. Обычно перехват и обработка события задается в параметрах элементов форм. Имя параметра обработки события начинается с приставки on, за которой следует имя самого события. Например, параметр обработки события Click будет выглядеть onClick. Значением параметра обработки события могут быть операторы языка JavaScript оператор присваивания.

Организация ветвлений в программах

При составлении программы часто необходимо выполнение различных действий, в зависимости от результатов проверки некоторых условий. Для организации ветвлений можно воспользоваться условным оператором, который имеет вид

```
if B {S1}
else {S2}
```

где B - выражение логического типа, S1 и S2 - операторы. Выполнение условного оператора осуществляется следующим образом: вычисляется значение выражения B, если оно истинно, то выполняются операторы S1, если ложно - операторы S2. Если последовательность операторов S1 или S2 состоит лишь из одного оператора, то фигурные скобки можно опустить.

Пример 3.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function rand() // создаем функцию генерирующую случайное число
{now=new Date() // определяем системную дату
num=(now.getSeconds())%10 //генерируем случайное число
num=num+1
}
function guessnum() //создаем функцию диалога
{guess=prompt("Угадай, какое?") //запрос пользователю
if (eval(guess) == num)
{alert("ПРАВИЛЬНО!!!")
rand()
}
else
alert("Нет. Попробуй еще раз.")
}
</SCRIPT>
<BODY onLoad="rand()">

<h3>Я загадал число от 1 до 10</h3>

<FORM NAME="myform">

<INPUT TYPE="button" VALUE="Угадай"
NAME="b1" onClick="guessnum()">

</FORM>
</BODY>
</HTML>
```

В результате действия сценария мы получим Web-страничку с небольшой игрой "Угадай число".

Лекция 11
Синтаксис языка. Операторы. Функции. Объекты.

Синтаксис: (особенности)

Чувствительность к регистру. Все ключевые слова пишутся в нижнем регистре. Все переменные и названия функций пишутся точно так же, как и были определены (переменные Str и str - различны)

Пробелы, табуляция и перевод строки. Эти символы игнорируются в JavaScript, так что можно использовать их для форматирования кода с тем, чтобы его было удобно читать.

Символ точка с запятой (;). Все операторы должны быть разделены этим символом. Если оператор завершается переводом строки, то точку с запятой можно опустить. При этом нужно следить за тем, чтобы при разрыве строки одного оператора, новая строка не начиналась бы с самостоятельного оператора.

Комментарии. JavaScript игнорирует любой текст расположенный между символами /* и */. Также игнорируется текст начинающийся символами // и заканчивающийся концом строки.

Индентификаторы. Индентификаторами являются имена переменных, функций, а также меток. Индентификаторы образуются из любого количества букв ASCII, подчеркивания (_) и символа доллара ($). Первым символом не может быть цифра.

Ключевые слова. Ключевые слова не могут использоваться в качестве индетификаторов. Ключевыми словами являются: break, case, continue, default, delete, do, else, export, false, for, function, if, import, in, new, null, return, switch, this, true, typeof, with.

Функции задаются объявлением function со списком параметров((возможно без них):

```
function sayHello(name) {
alert("Hello "+name)
}
```

Объекты браузеров

Браузеры поддерживают объекты различных типов. HTML-объектами являются объекты, которые соответствуют тегам языка HTML. К ним относятся метки, гиперсвязи и элементы формы - текстовые поля, кнопки, списки и др. Объекты верхнего уровня, или объекты браузера, - это объекты, поддерживаемые в среде браузера: window, location, history, document, navigator. Объекты, перечисленные в таблице, создаются автоматически при загрузке документа в браузер.

window Объект верхнего уровня в иерархии объектов языка JavaScript. Фреймосодержащий документ также имеет объект window.

document Содержит свойства, которые относятся к текущему HTML-документу, например имя каждой формы, цвета, используемые для отображения документа, и др. В языке JS большинству HTML-тегов соответствуют свойства объекта document.

location Содержит свойства, описывающие местонахождение текущего документа, например адрес URL.

navigator Содержит информацию о версии браузера. Свойства данного объекта обычно только для чтения. Например свойство: navigator.

appname содержит строковое значение имени браузера.history Содержит информацию обо всех ресурсах, к которым пользователь обращался во время текущего сеанса работы с браузером.

Объекты:

Объекты могут иметь свойства и методы.

Объекты в JavaScript также могут иметь свойства, например объект массив имеет свойство length позволяющее узнавать количество его элементов.

При обращении к свойству объекта необходимо отделять его точкой от названия объекта (объект.свойство).

Функции – это один из основных способов объединения операторов в логически связанные блоки. В языке JavaScript функция представляет собой группу выражений, служащих для выполнения какой-либо определенной задачи, объединенных под общим именем.

В отличие от большинства других языков программирования, JavaScript не делает различий между собственно функциями и процедурами. Подобно функциям, процедуры так же представляют собой программные блоки. Однако результаты выполнения процедур непосредственно сказываются на выполнении программы, в то время как функции должны возвращать значения. С этой точки зрения функции JavaScript можно рассматривать и как процедуры.

Определение и вызов функций

Прежде, чем вызывать и использовать функцию, ее надо определить. Определение функций в JavaScript имеет следующий синтаксис:

function ИмяФункции (аргументы) { блок выражений }

Таким образом, функция состоит из следующих частей, предваряемых ключевым словом function:

идентификатора, определяющего имя функции;

списка аргументов, заключенного в круглые скобки и разделенного запятыми;

операторов JavaScript, заключенных в фигурные скобки. Эти операторы могут включать вызовы других функций или даже самой этой функции (рекурсия).

В простейшем случае аргументы могут отсутствовать, а блок операций моет быть представлен единственным оператором:

function MyFirstFunc () { var MyMessage="Это – моя функция!"; alert(MyMessage); }

Здесь мы определили функцию, которая будет выдавать окно с сообщением «Это – моя функция!». Следует заметить, что даже если функция не принимает никаких аргументов, она все равно должна иметь пару круглых скобок после своего названия.

ВНИМАНИЕ

Важное замечание следует сделать по поводу переменных, объявляемых в теле функций. Такие переменные видны программе только внутри той функции, в которой они определены. Так, в примере с MyFirstFunc, доступ к переменной MyMessage возможен только внутри этой функции, но не вне нее.

Но чаще всего функции все-таки принимают какие-либо значения в качестве своих аргументов. Возьмем для примера ранее рассмотренный блок, вычисляющий список чисел, на которые 100 делится без остатка. Если этот блок вынести в отдельную функцию, то можно будет использовать его для того, чтобы выводить список делителей для любого числа. Для этого нам потребуется всего один аргумент, который и будет определять число, для которого нам нужно получить такой список:

function remainder_free(j) { var i=0; while (i<j) { i++; if (j%i != 0) continue; document.writeln(i); }; }

В этом случае значение переменной j функция получает при своем вызове. И как раз тут важно учесть, тот факт, что когда браузер загружает документ и интерпретирует объявление функции, он только считывает ее в память. А для того чтобы функция была выполнена, ее надо вызвать. Вызов функции может производиться из любого места JavaScript программы, и имеет такой вид:

ИмяФункции(параметры)

ИмяФункции означает имя ранее объявленной функции, а параметры – список из одного или нескольких значений, разделенных запятыми и передаваемых функции в качестве аргументов. Количество и порядок параметров, передаваемых функции, должен точно соответствовать списку аргументов, указанных при объявлении функции. Так, для случая с только что рассмотренной функцией поиска целых делителей, где используется 1 аргумент, вызов функции может быть таким:

remainder_free(100);

Если параметров будет меньше или больше чем надо, то в первом случае недостающим параметрам будет назначено значение NaN, а во втором – лишние параметры будут проигнорированы. Таким образом, следующие варианты вызова этой же функции не являются верными:

remainder_free; // так функции не вызывают вообще remainder_free(); // недостаточно параметров remainder_free(100,0); // лишний параметр

В случае если функция вызывается не просто как процедура (как только что рассмотренная), а именно как функция, т.е. если она должна возвращать значение, то в ней должен содержаться специальный оператор return, в котором указывается выражение, значение которого возвращает функция. Например, если требуется создать функцию, которая будет принимать в качестве аргумента число, и возвращать его третью степень, то она может выглядеть примерно так:

function cube(value) { return value * value * value; }

Таким образом, чтобы в любом месте программы вычислить значение числа в 3-й степени, достаточно написать такое выражение:

var x = cube(3);

В результате выполнения этого выражения переменной x будет присвоено значение 27. Примеры вызова функций вы можете посмотреть в JavaScript-калькуляторе (файл calc.html).

Как уже отмечалось, функции могут содержать в себе вызовы других функций. Например, в случае с поиском делителей можно было бы вывести, к примеру, не сами делители, а их 3-ю степень, для чего из функции remainder_free вызывалась бы функция cube:

function remainder_free(j) { var i=0, ic=0; while (i<j) { i++; if (j%i != 0) continue; ic = cube(i); document.writeln(ic); }; }

Таким образом, функции можно вызывать друг и друга, управляя тем самым ходом исполнения сценария.

Вложение функций и рекурсия

Как и в некоторых других языках программирования (например, как в Паскале), функции в JavaScript могут быть вложенными. Суть использования вложенных функций состоит в том, что такая локальная функция не видна из других частей программы, что может быть удобными при разработке достаточно больших проектов, например, чтобы исключить переопределения имен переменных. Допустим, если бы рассмотренная нами функция cube была бы ненужной по ходу всего сценария, за исключением одного частного случая (скажем, функции, складывающей кубы двух чисел), то можно было бы сделать ее вложенной:

function AddCubes(x,y) { function cube(value) { return value * value * value; }; return cube(x) + cube(y); }

В данном случае определена функция AddCubes, внутри которой определена еще одна функция – cube. При этом следует учитывать, что для вложенной функции видны переменные, используемые во внешней («наружной») функции, но не наоборот. В то же время сама вложенная функция доступна только для выражений из внешней функции: в данном случае функцией cube можно будет воспользоваться только из функции AddCubes.

Язык JavaScript поддерживает также и такую весьма полезную возможность, как рекурсивный вызов функций, или просто рекурсию. Под рекурсией понимают вызов функции из самой себя. Рассмотрим этот вопрос на следующем примере:

function recfunc(x) { x--; if (x>5) recfunc(x); alert(x); }

Здесь мы объявили функцию recfunc, принимающую один аргумент, и вызывающую саму себя до тех пор, пока значение этого аргумента больше 5. Хотя на первый взгляд может показаться, что такое поведение функции похоже на обычный цикл, на самом деле все работает несколько по-иному: если вы вызовите ее со значением 8, то она выдаст вам 3 сообщения в следующей последовательности: 5, 6, 7. Иначе говоря, функция вызывала саму себя до тех пор, пока значение x было больше 5, и собственно вывод сообщений начала 3-я по уровню получившейся вложенности функция, которая и вывела первое сообщение (в данном случае им стало декрементированное 6, т.е. 5).

Массивы аргументов

В языке JavaScript аргументы функции могут рассматриваться как массив. Для обращения к массиву аргументов используется следующий синтаксис:

arguments[i]; ИмяФункции.arguments[i];

Здесь i должно быть целым числом, являющимся порядковым номером аргумента, при этом отсчет аргументов начинается с нуля. В случае, когда обращение к массиву аргументов происходит из самой функции, имя функции можно опустить. Чтобы узнать число аргументов, используйте выражение свойство массива length:

arguments.length;

Использование массива аргументов призвано разрешить задачу вызова функции с числом аргументов, превышающим указанное при определении функции. Это может быть полезным в том случае, если вы не знаете заранее, сколько аргументов должна принять функция. В таком случае при описании функции обычно определяют только обязательные

аргументы, а с остальными работают через массив. Допустим, нам нужно написать функцию, которая будет склеивать несколько строк при помощи указанного разделителя. В таком случае нам требуется определить только один аргумент – разделитель, а все остальные значения такая функция должна будет принимать через массив аргументов:

function ConcatSeparated(separator) { result=""; // инициализируем возвращаемое значение for (var i=1; i<arguments.length; i++) // обход всех аргументов в массиве { result = result + arguments[i] + separator; // собственно склеивание }; return result; // возврат результата }

Эта функция будет возвращать одну строку, состоящую из списка аргументов, начиная со второго, и «склеенную» первым аргументом. Пример ее использования можно посмотреть в файле argarray.html.

Но, на самом деле, применение массива аргументов позволяет обходиться вообще без объявления параметров. Допустим, нам нужна функция, возвращающая среднее значение нескольких чисел. Как известно, для этого достаточно сложить все числа и разделить результат на их количество. Соответственно, нам пригодится свойство массива length, отвечающее за количество аргументов. В результате функция получит примерно такой вид:

function calcAvg() { var summ=0; // инициализируем переменные for (var i=0; i<arguments.length; i++) // обход всех аргументов в массиве { summ += arguments[i]; // вычисление суммы }; return summ / arguments.length; // вычисление и возврат результата }

Здесь хотелось бы обратить внимание на то, что обход массива начинается с нулевого элемента. Это вполне естественно, поскольку нумерация элементов массивов начинается с 0. Если же у вас возник вопрос, почему в предыдущем примере отсчет начинался с 1, то обратите внимание на то, что первым аргументом там был separator, объявленный явно, и который ни с чем склеивать не требовалось.

Другой момент – это использование сокращенной формы записи для подсчета суммы. Конечно, можно было бы написать это выражение в более привычном виде:

summ = summ + arguments[i];

Но, поскольку уж в JavaScript предусмотрены сокращенные формы записи, то почему бы их и не использовать? Ну и, наконец, после оператора return записана не переменная, а выражение, что так же вполне допустимо с точки зрения синтаксиса языка.

Что касается использования только что созданной функции, то ее вызов может быть произведен одним из следующих способов:

var avg = calcAvg(1,3,5); // переменной avg будет присвоено значение 3 var avg = calcAvg(2); // переменной avg будет присвоено значение 2 var avg = calcAvg(0,4,8,14); // переменной avg будет присвоено значение 6,5

Заметьте, что во всех случаях используется некоторая переменная (avg), которой присваивается значение, возвращаемое функцией. Если бы такой переменной не было указано, то результат вычислений попросту пропал бы. Иначе говоря, если функция возвращает какое-либо значение, то оно должно использоваться – не обязательно путем присвоения переменной, но и любым другим способом:

alert(calcAvg(10,100));

В данном случае возвращаемое функцией значение будет использовано в качестве параметра метода alert(). В использовании возвращаемого значения и кроется главное отличие между собственно функциями и функциями, используемыми в качестве процедур (без оператора return): последние ничего не возвращают, а только производят какую-либо работу.

Предопределенные функции

В JavaScript имеется ряд функций, являющихся частью самого языка, и не связанных с какими-либо конкретными объектами (вернее, они являются функциями некоего глобального объекта global, на который, тем не менее, никогда не требуется ссылаться). Такие функции называются предопределенными.

К предопределенным функциям JavaScript 1.3 относятся: eval, isFinite, isNaN, parseInt, parseFloat, Number, String, escape и unescape. В JavaScript 1.5 вместо функций escape и unescape используются 4 новых функции: encodeURI, encodeURIComponent,

decodeURI, и decodeURIComponent. Краткое описание всех этих функций приведено в таблице.

Таблица. Предопределенные функции языка JavaScript

Функция	Синтаксис	Описание
eval	eval(выражение)	Обрабатывает строку как код JavaScript
isFinite	isFinite(значение)	Проверяет, что значение является конечным числом
isNaN	isNaN(значение)	Проверяет, что значение не является числом
parseInt	parseInt(строка, система)	Преобразует строку в целое
parseFloat	parseFloat(строка)	Преобразует строку в вещественное число
Number	Number(объект)	Преобразует объект в число
String	String(объект)	Преобразует объект в строку
escape	escape(строка ASCII)	Преобразует строку в последовательность символов
unescape	unescape(строка)	Преобразует последовательность escape-символов в строку
encodeURI	encodeURI(URL в ASCII)	Кодирует URI полностью путем замены символов на их коды UTF-8
encodeURIComponent	encodeURIComponent(URL в ASCII)	Кодирует URI по составным частям путем замены символов на их коды UTF-8
decodeURI	decodeURI(URL в UTF-8)	Раскодирует значение, созданное при помощи encodeURI
decodeURIComponent	decodeURIComponent(URL в UTF-8)	Раскодирует значение, созданное при помощи encodeURIComponent

Рассмотрим некоторые функции подробнее. Начнем с функции eval, которая передает интерпретатору строку с кодом JavaScript без ссылки на какой-либо конкретный объект. Способ использования у этой функции следующий:

eval(выражение);

Здесь «выражение» – строка, которая должна содержать код на языке JavaScript для обработки интерпретатором. Ценность функции eval в том, что с ее помощью можно динамически составлять фрагменты программы и отправлять их на выполнение. При дальнейшем написании сценариев мы не раз воспользуемся этой функцией.

Функция isFinite проверяет, является ли переданное ей значение конечным числом. Если в качестве значения, передаваемого этой функции, указать -3, то результатом будет истина. А если указать Infinity (специальное ключевое слово, обозначающее бесконечность), результатом будет ложь, как, впрочем, и в том случае, если вместо числа указать строку.

Функция isNaN выполняет противоположную задачу – она проверяет, что указанный аргумент не является числом. Здесь следует отметить, что в JavaScript имеется еще одно ключевое слово – NaN. Оно используется, когда требуется указать значение, не являющееся числом. Например, такой вызов функции isNaN возвратит истину:

isNaN(NaN);

Для преобразования строки в число с плавающей запятой служит функция parseFloat. Например, чтобы присвоить некоторой переменной значение, которое должно быть вещественным числом, следует написать следующее выражение:

var a = parseFloat("31416e-4");

В этом случае переменной a будет присвоено значение 3,1416. В том случае, если преобразовать строку в число не представляется возможным, функция parseFloat возвращает значение NaN.

Кроме функции parseFloat, для преобразования строк в число имеется функция parseInt. В отличие от parseFloat, она может принимать как один, так и два аргумента:

parseInt("25"); parseInt("25", 10);

Здесь первый аргумент является числом для преобразования, а второй указывает на систему счисления, по которой следует производить преобразование. По умолчанию используется десятеричная система счисления, поэтому оба приведенных выше вызова функции аналогичны. Но чтобы точнее разобраться с этой функцией, давайте рассмотрим пример, в котором все ее вызовы возвращают значение 15:

var x = parseInt("15"); /* если не оговорено иначе, подразумевается, что система счисления – десятеричная */ var x = parseInt("15", 10); /* здесь мы явно указываем десятеричную систему счисления */ var x = parseInt(15.99, 10); /* в данном случае исходным значением служит не строка, а число с плавающей точкой */ var x = parseInt("F", 16); // шестнадцатеричное F равно десятеричному 15 var x = parseInt("FXX123", 16); /* если какой-либо знак распознать не удается, следующие за ним значения отбрасываются */ var x = parseInt("17", 8); // Восьмеричная система тоже иногда используется var x = parseInt("1111", 2); // Двоичная система – тоже не редкость

В том случае, когда даже первый символ преобразовать в число не удастся, функция возвращает значение NaN:

parseInt("8",8); // В восьмеричной системе счисления нет цифры 8!

Для преобразования строк и чисел в JavaScript имеются еще 2 функции – String и Number. Они используются для преобразования объектов в строку и число соответственно. Например, если нам надо что-либо преобразовать в строку, например, объект document, то можно написать так:

var doc = String(document);

В данном случае в качестве значения переменной doc мы получим строку, описывающую объект как «[object HTMLDocument]». Примеры работы этих и других предопределенных функций можно найти в файле predef.html.

И, наконец, осталось рассмотреть функции кодирования. На самом деле, в клиентских сценариях они практически не используются, и предназначены для работы на стороне сервера (для случая, если web-сервером является Netscape Enterprise). Так что мы не будем на них останавливаться, и отметим лишь, что, поскольку функции escape и unescape не работают с национальными символами, то, начиная с JavaScript 1.5, они объявлены устаревшими, а вместо них следует использовать encodeURI, decodeURI, encodeURIComponent и decodeURIComponent.

Лекция 12
Управляющие структуры JavaScript

Обычно операторы в скрипте JavaScript выполняются один за другим в том порядке, в котором они записаны. Это называется последовательным выполнением. По умолчанию программа работает именно таким образом.

Альтернативой последовательному выполнению является передача программного потока в другую часть скрипта. Вместо выполнения следующего по порядку оператора может выполняться другой.

Чтобы скрипт был полезен, эта передача управления должна осуществляться логически организованно.Передача управления в программе основывается на решении, результатом которого является оператор истинности (возвращающий логическое значение **true** или **false**). Создается выражение, которое затем проверяется на наличие результата true. Это выполняется с помощью двух основных разновидностей структур программы.

Первая разновидность — структура выбора. Она используется для указания альтернативных вариантов выполнения программы, создавая точку ее разветвления (подобную развилке на дороге). В JavaScript доступны четыре структуры выбора:

- структура единственного выбора (**if**);
- структура двойного выбора (**if/else**);
- встроенный троичный оператор **?:**;
- структура множественного выбора (**switch**).

Вторая разновидность управляющей структуры программы — структура повторения. Она используется для указания необходимости повторять действие, пока определенное условие остается истинным. Когда выполняются условия управляющего оператора (обычно после выполнения определенного числа итераций), управление передается следующему оператору за пределами структуры повторения. В JavaScript доступны четыре структуры повторения:

- проверка выражения в начале цикла (**while**);
- проверка выражения в конце цикла (**do/while**);
- выполнение для каждого свойства объекта (**for/in**);
- повторение под управлением счетчика (**for**).

Используя вложенные и следующие друг за другом управляющие структуры выбора и повторения, можно создавать довольно сложные скрипты.

Третья форма структурированного программного потока обеспечивается обработкой исключений, которая в данном документе не рассматривается.

Использование условных операторов

В JavaScript поддерживаются условные операторы **if** и if...else. При использовании операторов **if**проверяется выполнение условия. В случае положительного результата выполняется соответствующий код JavaScript. Если условие не выполнено, выполняется альтернативный код (в операторе **if...else**).Простейшая форма оператора **if** представляется одной строкой, однако чаще всего операторы **if** и**if...else** содержат несколько строк.

В следующих примерах демонстрируется синтаксис, используемый для операторов **if** и **if...else**. В первом примере показывается простейшая форма логического условия. Оператор или блок операторов, расположенных после **if**, выполняется в том, и только в том, случае, если элемент в скобках принимает значение **true** (или может быть преобразовано в это значение).

JavaScript

```
function GetReaction(newShip, color, texture, dayOfWeek)
{
  // The test succeeds if the newShip Boolean value is true.
  if (newShip)
  {
    return "Champagne Bottle";
  }

  // The test succeeds if both conditions are true.
  if (color == "deep yellow" && texture == "large and small wrinkles")
  {
    return "Is it a crenshaw melon?";
  }

  // The test succeeds if either condition is true.
  if ((dayOfWeek == "Saturday") || (dayOfWeek == "Sunday"))
  {
    return "I'm off to the beach!";
  }
  else
  {
    return "I'm going to work.";
  }
}

var reaction = GetReaction(false, "deep yellow", "smooth", "Sunday");
document.write(reaction);

// Output: I'm off to the beach!
```

Условный оператор

В JavaScript также поддерживается неявная условная форма. Вместо слова **if** перед условием в ней используется знак вопроса после проверяемого условия. Определяется 2 варианта действий: для случая выполнения условия и для случая его невыполнения. Альтернативные варианты разделяются двоеточием.

JavaScript

```
var AMorPM = (theHour >= 12) ? "PM" : "AM";
```

Если необходимо одновременно проверить несколько условий и известно, что одно из них выполняется или не выполняется с большей вероятностью по сравнению с другими, можно использовать так называемую быструю оценку, ускоряющую выполнение скрипта. При оценке логического выражения JavaScript оценивает ровно столько подвыражений, сколько необходимо для получения результата.

Например, если имеется выражение "И" ((x == 123) && (y == 6)), JavaScript сначала проверяет, равен ли x 123. Если нет, все выражение не может быть истинным, даже если y равен 6. Поэтому проверка значения y не выполняется, и JavaScript возвращает значение **false**.

Аналогично, если только одно из нескольких условий должно давать результат **true** (при использовании оператора ||), проверка останавливается, как только выясняется, что какое-либо условие выполняется.Это эффективно в том случае, если проверяемые условия предполагают выполнение вызовов функций или других сложных выражений. Учитывая вышесказанное, при создании выражений "ИЛИ" сначала размещайте условия, для которых выше вероятность значения **true**. При создании выражений "И" сначала размещайте условия, для которых выше вероятность значения **false**.

Преимущество создания скрипта таким способом можно увидеть в следующем примере. Здесь функция **runsecond()** не будет выполняться, если **runfirst()** возвращает значение 0.

JavaScript

```
if ((runfirst() == 0) || (runsecond() == 0)) {
   // some code
}
```

Использование циклов

Есть несколько способов повторного выполнения оператора или блока операторов. Повторяющееся выполнение называется *циклом* или *итерацией*. Проще говоря, итерация — это однократное выполнение цикла. Для управления циклами обычно выполняется проверка переменной, значение которой изменяется при каждом выполнении цикла. В JavaScript поддерживается 4 типа циклов: циклыfor, циклы for...in, циклы while и циклы do...while.

Использование циклов for

Оператор **for** указывает переменную счетчика, условие проверки и действие, обновляющее счетчик.Условие проверяется перед каждой итерацией цикла. В случае успешной проверки выполняется код внутри цикла. Если проверка не пройдена успешно, код внутри цикла не выполняется, а программа продолжает работу с первой строки, следующей непосредственно после цикла. После выполнения цикла переменная счетчика обновляется перед началом следующей итерации.

Если условие цикла не выполняется, цикл не запускается. Если условие цикла выполняется всегда, образуется бесконечный цикл. Циклы первого типа иногда бывают полезны, но бесконечные циклы используются крайне редко, поэтому будьте внимательны при определении условий цикла.

JavaScript

```
// The update expression ("icount++" in the following examples)
// is executed at the end of the loop, after the block of
// statements that forms the body of the loop is executed, and
// before the condition is tested.
```

```
// Set a limit of 10 on the loop.
var howFar = 10;

// Create an array called sum with 10 members, 0 through 9.
var sum = new Array(howFar);
sum[0] = 0;

// Iterate from 0 through 9.
var theSum = 0;
for(var icount = 0; icount < howFar; icount++)
{
  theSum += icount;
  sum[icount] = theSum;
}

// This code is not executed at all, because icount is not greater than howFar.
var newSum = 0;
for(var icount = 0; icount > howFar; icount++)
{
  newSum += icount;
}

// This is an infinite loop.
var sum = 0;
for(var icount = 0; icount >= 0; icount++)
{
  sum += icount;
}
```

Использование циклов "for...in"

В JavaScript предусмотрен особый тип цикла для перебора всех определяемых пользователем свойствобъекта или всех элементов массива. Счетчик цикла **for...in** представляет собой не число, а строку. Он содержит имя текущего свойства или индекс текущего элемента массива.

JavaScript

```
// Create an object with some properties
var myObject = new Object();
myObject.name = "James";
myObject.age = "22";
myObject.phone = "555 1234";

// Enumerate (loop through)_all the properties in the object
for (var prop in myObject)
{
   // This displays "The property 'name' is James", etc.
   document.write("The property '" + prop + "' is " + myObject[prop]);
   // New line.
   document.write("<br />");
}
```

Хотя циклы **for...in** напоминают циклы **For Each...Next** VBScript, они действуют иначе. В цикле **for...in**JavaScript выполняется перебор свойств объектов JavaScript, а в цикле **For Each...Next** VBScript — перебор элементов коллекции. Для перебора коллекций в JavaScript необходимо использовать объектОбъект Enumerator (JavaScript) или метод **forEach** (если он присутствует) объекта коллекции. Хотя некоторые объекты, например в Internet Explorer, поддерживают как цикл **For Each...Next** VBScript, так и цикл **for...in** JavaScript, большинство объектов не поддерживает оба цикла.

Использование циклов while

Цикл **while** аналогичен циклу **for**. Различие заключается в том, что в цикле **while** нет встроенной переменной счетчика и выражения обновления. Цикл **while** следует использовать, если для управления повторным выполнением оператора или блока операторов требуется более сложное правило, чем просто "выполнить код n раз". В следующем примере используется объектная модель Internet Explorer и цикл **while**, чтобы задать пользователю простой вопрос.

```
JavaScript
var x = 0;
while ((x != 5) && (x != null))
{
   x = window.prompt("What is my favorite number?", x);
}

if (x == null)
   window.alert("You gave up!");
else
   window.alert("Correct answer!");
```

Примечание

Так как циклы **while** не имеют явных переменных счетчика, они более подвержены бесконечной цикличности, чем другие типы циклов. Более того, поскольку положение и время обновления условия цикла не всегда просто найти, цикл **while**, в котором условие никогда не обновляется, можно написать легко. По этой причине следует проявлять осторожность при построении циклов**while**.

Как указано выше, в JavaScript также есть цикл **do...while**, аналогичный циклу while, но отличающийся тем, что он гарантированно выполняется по крайней мере один раз, так как условие проверяется в конце цикла, а не в начале. Например, показанный выше цикл можно переписать следующим образом:

```
JavaScript
var x = 0;
do
{
   x = window.prompt("What is my favorite number?", x);
} while ((x != 5) && (x != null));

if (x == null)
   window.alert("You gave up!");
else
   window.alert("Correct answer!");
```

Использование операторов break и continue

В JavaScript оператор break используется для остановки цикла при выполнении определенного условия. (Обратите внимание, что оператор **break** также используется для выхода из блока **switch**). Операторcontinue можно использовать для мгновенного перехода к следующей итерации с пропуском оставшейся части блока кода при обновлении переменной счетчика в цикле **for** или **for...in**.

Следующий пример основан на предыдущем. В нем для управления циклом используются операторы**break** и **continue**.

```
JavaScript
var x = 0;
do
{
   x = window.prompt("What is my favorite number?", x);
```

```
    // Did the user cancel? If so, break out of the loop
    if (x == null)
       break;

    // Did they enter a number?
    // If so, no need to ask them to enter a number.
    if (Number(x) == x)
       continue;

    // Ask user to only enter in numbers
    window.alert("Please only enter in numbers!");

} while (x != 5)

if (x != 5)
   window.alert("You gave up!");
else
   window.alert("Correct answer!");
```

Лекция 13
Графика в JavaScript

При создании дизайна сайта, очень удобно использовать рисунки и анимацию основанную на JavaScript фреймверках. Это быстро, удобно и красиво. Но…

Перед тем как начать рисовать что-то в своем браузере, задайте себе три вопроса:

1. **Нужна ли вам поддержка старых браузеров?** Если ответ ДА, вашим выбором будет Raphael.js. Он поддерживается браузерами начиная с 7 и 3. Некоторые элементы графики будут работать даже в 6. Хотя эта библиотека не поддерживает технологий которые предоставляют следующие библиотеки…
2. **Нужна ли вам поддержка Android устройств?** Android не поддерживает SVG графики, поэтому вам придется использовать Paper.js или Processing.js. Ходят слухи, что Android 4 поддерживает обработку SVG, но большинство устройств работают под старой версией Android.
3. **Нужна ли вам интерактивность?** Raphael и Paper.js предрасположены на интерактивность через клики, перетягивания и касания. Processing.js не поддерживает никаких объектных событий, поэтому действия пользователя будет сложно отслеживать. Processing.js может рисовать отличную анимацию на главной странице, но для интерактивных приложений лучше использовать другие средства.

Paper.js, Processing.js и Raphael лидирующие библиотеки для рисования на веб страницах средствами javascript. Вы можете использовать технологию Flash, но эти три отлично работают с HTML5 и поддерживают наибольшее число браузеров.

Выбор правильного фреймверка определит успех вашего проекта. В этой статье мы будем обозревать преимущества и недостатки каждой библиотеки, только полезная информация, которая позволит сделать правильный выбор.

Код использованный в этой статье доступен на демо странице.

Обзор фреймверков для рисования

	Paper.js	Processing.js	Raphaël
Технология	canvas tag	canvas tag	SVG
Язык	PaperScript	Processing script	JavaScript
Браузеры	IE 9	IE 9	IE 7
Мобильные	Да	Да	Только iOS
Модель	Векторная и растровая	Растровая	Векторная
Размер	56 KB	64 KB	20 KB

Они все основаны на JavaScript, но рисование происходит немного разными способами. Raphael написан на чистом javascript коде, но Paper.js использует PaperScript, и Processing.js использует собственный скрипт.

Они все поддерживают FireFox, Opera, Chrome и Safari, но Internet Explorer исключение – Paper.js и Processing.js используют тег canvas, который поддерживается в IE9 и выше.

PaperScript это расширение к javascript, которое позволяет писать скрипты, которые не взаимодействуют с именами функций и переменных javascript. Это исключает конфликты этой библиотеки с другими js кодами.

Processing.js основан на фреймверку Processing, который запускается на Java виртуальной машине. При написании код может нагадывать Java, но он более похож на JavaScript и не нуждается в сложном синтаксисе.

Рисование с помощью этих троих библиотек простое, если вы когда либо сталкивались с JavaScript.

Начнем

Начинаем с импорта библиотеки. Процесс настройки каждой из них будет немного отличаться.

Настройка Paper.js

```
<head>
<script src="paper.js" type="text/javascript" charset="utf-8"></script>
<script type="text/paperscript" canvas="paperCircle" src="paper_circle.pjs" id="script">
</script>
</head>
<body>
<canvas id="paperCircle" class="canvas" width="200" height="200" style="background-
color: white;"></canvas>
```

Paper.js потребует тип скрипта text/paperscript и ID canvas тега, который вы будете использовать.

Настройка Processing.js

```
<head>
<script src="processing.js" type="text/javascript" charset="utf-8"></script>
</head>
<body>
<canvas width="200" height="200" class="canvas" data-processing-
sources="processing_circle.java"></canvas>
```

Processing.js использует data-processing-sources атрибут canvas тега, для загрузки вашего рисунка. Я использую расширение .java для исходного файла Processing библиотеки, в таком случае мой редактор правильно подсвечивает код. Некоторые автора используют .pde или .pjs расширение. Выбирать вам.

Настройка Raphael

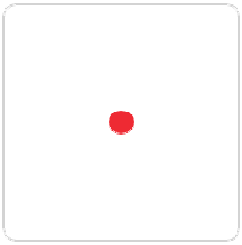
```
<head>
<script src="raphael-min.js" type="text/javascript" charset="utf-8"></script>
<script src="raphael_circle.js" type="text/javascript" charset="utf-8"></script>
</head>
```

Raphael подключается как и любой другой файл . Он отлично работает с функциями jQuery или другими JavaScript фреймверками.

Теперь начнем рисовать.

Объектно-ориентированное рисование на JavaScript

Как Paper.js так и Raphael используют объектно-ориентированное рисование: вы можете нарисовать круг и получить объект круга. Processing.js рисует круг и ничего не возвращает обратно. Следующий пример все разъяснит. Давайте начнем с круга в центре экрана в точке 100, 100.



Paper.js:

```
var circle = new Path.Circle(new Point(100, 100), 10);
circle.fillColor = '#ee2a33';
```

Raphaël:

```
var paper = Raphael('raphaelCircle', 200, 200);
var c = paper.ellipse(100, 100, 10, 10);
c.attr({'fill': '#00aeef', 'stroke': '#00aeef'});
```

Processing.js:

```
void setup() {
  size(200, 200);
}
void draw() {
  background(#ffffff);
  translate(100, 100);
  fill(#52b755);
  noStroke();
  ellipse(0, 0, 20, 20);
}
```

Каждый сниппет кода рисует тот же круг. Разница в том, что вы можете с этим делать.

Paper.js создает круг как path объект. Мы можем использовать этот объект и изменить его позже. В Paper.js, **circle.fillColor = 'red';** - делает наш круг красным, и**circle.scale(2)** – увеличивает круг в два раза.

Raphaël также объектно-ориентированная модель. Мы также можем изменить цвет круга с помощью **circle.attr('fill', 'red');**, также увеличить его в два раза с помощью**circle.scale(2, 2);**. Главное, что наш круг остается объектом, с которым мы можем работать позже.

Processing.js не использует объекты. Функция **ellipse** ничего не возвращает. Если мы уже нарисовали круг в Processing.js, это уже часть сформированной картинки, как краска на холсте. Это не отдельный объект, который может быть изменен с помощью атрибутов. Чтобы изменить цвет, нам придется нарисовать новый круг поверх прежнего.

Когда мы вызываем **fill**, это изменяет цвет заливки для всех объектов, которые мы будем рисовать после. После вызова **translate** и fill, каждая форма будет залита зеленым цветом.

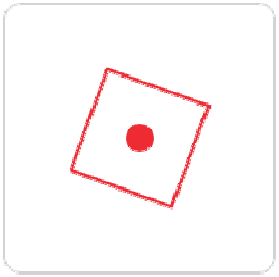
Так как функции изменяют все, мы можем получить нежелательный результат. Вызвать функцию и все станет зеленым! Для этого существуют **pushMatrix** и**popMatrix** функции, для изоляции изменений. Не забывайте их вызывать.

Если Processing.js не объектно-ориентированный фреймверк, это не значит, что он хуже. Paper.js и Raphael содержат настройки для всего что вы рисуете, поэтому используют больше памяти, на более сложных анимациях ваше приложение может тормозить.

Processing.js содержит минимум настроек для рисования форм, поэтому использует минимум памяти.

Анимируем рисунки на JavaScript

Вращая круг мы ничего интересного не увидим, поэтому мы создадим квадрат вокруг.



Анимация в Processing.js

Processing.js поддерживает анимацию с предустановленными настройками и функциями отрисовки, как в примере:

```
float angle = 0.0;
void setup() {
  size(200, 200);
  frameRate(30);
}
void draw() {
  background(#ffffff);
  translate(100, 100);
  fill(#52b755);
  noStroke();
  ellipse(0, 0, 20, 20);
  rotate(angle);
  angle += 0.1;
  noFill();
  stroke(#52b755);
  strokeWeight(2);
  rect(-40, -40, 80, 80);
}
```

Функция setup вызывается однажды, при старте приложения. Мы указываем**frameRate(30)**, что значит, наша функция будет вызвана 30 раз в секунду, для перерисовки изображения. Так будет создаваться анимация.

Функция draw начинается с заливки фона canvas. Заливаем все белым цветом, другими словами очищаем холст. Как вы помните, здесь мы не можем манипулировать объектами.

Далее мы устанавливаем систему координат 100, 100 с помощью translate. Это значит, все что в квадрате 100 x 100 будет белым. Далее мы будем вращать определенный

angle, который возрастает с каждой отрисовкой, что создает впечатление анимации. Последний шаг, нарисовать квадрат с помощью функций **fill**и **rect**.

Функция rotate в Processing.js принимает радианы вместо градусов. Поэтому мы увеличиваем на 0,2 с каждой отрисовкой фрейма.

Анимация в Paper.js

Paper.js делает создание простой анимации более удобным способом нежели предыдущий фреймверк.

```
var r;
function init() {
  var c = new Path.Circle(new Point(100, 100), 10);
  c.fillColor = '#ee2a33';
  var point = new Point(60, 60);
  var size = new Size(80, 80);
  var rectangle = new Rectangle(point, size);
  r = new Path.Rectangle(rectangle);
  r.strokeColor = '#ee2a33';
  r.strokeWidth = 2;
}
function onFrame(event) {
  r.rotate(3);
}
init();
```

Мы определяем квадрат как объект, и Paper.js будет рисовать его на экран. При каждом обновлении фрейма, мы будем немножко поворачивать его. Нам не нужно перерисовывать его каждый раз и беспокоиться за сохранность других объектов.

Анимация в Raphael

Анимация в Raphael основана на стандартном JavaScript синтаксисе. Мы будем использовать стандартную **setInterval** функцию.

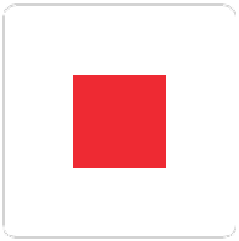
```
var paper = Raphael('raphaelAnimation', 200, 200);
var c = paper.ellipse(100, 100, 10, 10);
c.attr({
  'fill': '#00aeef',
  'stroke': '#00aeef'
});
var r = paper.rect(60, 60, 80, 80);
r.attr({
  'stroke-width': 2,
  'stroke': '#00aeef'
});
setInterval(function() {
  r.rotate(6);
}, 33);
```

Raphael схож с Paper.js, та же объектная модель. Мы имеем квадрат, к которому вызываем функцию **rotate**. Так мы можем анимировать квадрат с минимальными затратами кода.

Интерактив в Raphael.js, Paper.js, Processing.js

Преимущество в рисовании на JavaScript в том, что мы не ограничиваемся только рисунками и анимацией. Также можно взаимодействовать с пользователем, что не маловажно для веб страницы.

Raphael поддерживает модель событий подобно JavaScript, поэтому будет просто отследить клики, перетаскивания и касания. Давайте создадим кликабельный квадратик.



Интерактивность с помощью Raphael

```
var paper = Raphael('raphaelInteraction', 200, 200);
var r = paper.rect(60, 60, 80, 80);
r.attr({'fill': '#00aeef', 'stroke': '#00aeef'});
var clicked = false;
r.click(function() {
  if (clicked) {
    r.attr({'fill': '#00aeef', 'stroke': '#00aeef'});
  } else {
    r.attr({'fill': '#f00ff0', 'stroke': '#f00ff0'});
  }
  clicked = !clicked;
});
```

Функция **click** в Raphael работает так же, как и в jQuery. Вы можете добавить ее к любому событию. Когда мы получили событие клика, нам будет просто изменить цвет квадрата. Raphael имеет больше функций: перетаскивание, наведение и другие функции интерактивности.

Интерактивность с помощью Paper.js

Paper.js управляет интерактивностью разными способами, но все они просты и понятны:

```
var hitOptions = {
  fill: true,
  tolerance: 5
};
function init() {
  var point = new Point(60, 60);
  var size = new Size(80, 80);
  var rectangle = new Rectangle(point, size);
  r = new Path.Rectangle(rectangle);
  r.fillColor = '#ee2a33';
}
function onMouseUp(event) {
  var hitResult = project.hitTest(event.point, hitOptions);
  if (hitResult && hitResult.item) {
    if (hitResult.item.clicked) {
      hitResult.item.fillColor = '#ee2a33';
    } else {
      hitResult.item.fillColor = '#f00ff0';
    }
    hitResult.item.clicked = !hitResult.item.clicked;
  }
}
init();
```

Paper.js взаимодействует с мышью с помощью концепта «hit testing», который находит точку в которой установлена мышь и определяет какие объекты лежат под этой

точкой. С помощью этого мы можем «уточнять», насколько курсор должен быть близко к квадрату, или что будет происходить по мере приближения к центру квадрата.

Интерактивность с помощью Processing.js

Processing.js определяет нажатие мыши с помощью некоторых приспособлений. Эта библиотека не поддерживает объектных моделей, поэтому приходиться приспосабливаться.

```
float bx;
float by;
int bs = 20;
boolean bover = false;
boolean clicked = false;
void setup() {
  size(200, 200);
  bx = width/2.0;
  by = height/2.0;
  noStroke();
  fill(#52b755);
  frameRate(10);
}
void draw() {
  background(#ffffff);
  // Проверка на наличие курсора над квадратиком
  if (mouseX > bx-bs && mouseX < bx+bs &&      mouseY > by-
bs && mouseY < by+bs) {
    bover = true;
  } else {
    bover = false;
  }
  translate(100, 100);
  rect(-40, -40, 80, 80);
}
void mousePressed() {
  if (bover) {
    if (clicked) {
      fill(#52b755);
    } else {
      fill(#f00ff0);
    }
    clicked = !clicked;
  }
}
```

После того, как Processing.js нарисовала квадрат, она тут же забыла о нем. Если мы хотим поменять цвет квадрата при нажатии мыши, нам придется все высчитывать самим. Функция draw определяет позицию курсора, если он лежит в области квадрата, мы перерисовываем на новый.

Этот код не настолько плох для квадрата, но для круга и более сложных форм это неприемлемо.

Нет абсолютного победителя

Каждый фреймверк имеет свои преимущества. Естественно мы не можем этого увидеть на простых кружочках и квадратиках. Для этого нужны серьезные проекты.

Много примеров графических решений можно найти на официальных сайтах:

http://raphaeljs.com/

http://processingjs.org/exhibition/

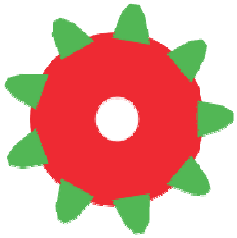
http://paperjs.org/examples/

Также стоит учитывать возможности подключения специальных инструментов, для решения конкретных задач. Таких как рисования графиков, специальных форм и других. Более подробно об инструментах можно узнать на самом сайте фреймверка.

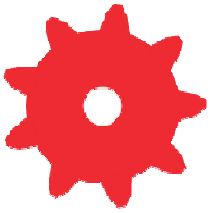
Рисуем что-то более сложное на JavaScript

Перед этим мы игрались с рисованием простых элементов. Теперь мы расширим горизонты и увидим, как проявит себя каждая библиотека в более сложной задаче. Заодно и научимся совмещать формы и создавать комплексную анимацию.

Мы создадим шестерни, которые состоят из двух кругов, с зубчиками по внешнему периметру.



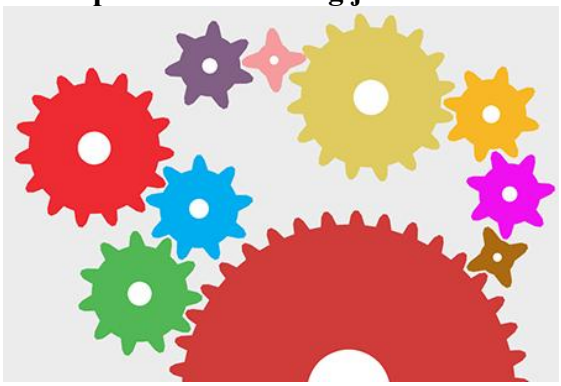
Когда все формы имеют один цвет, они выглядят как одна шестерня.



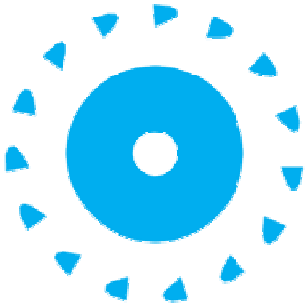
Каждая шестерня будет немного вращаться с каждым обновлением фрейма. Первой шестерни будет задана скорость вращения, все остальные будут вращаться соответственно первой.

Шестерни на Paper.js

Шестерни на Processing.js:



Шестерни на Raphaël:



На Raphael функция вращения работает по-другому, в отличии от Paper.js и Processing.js. Raphael не поддерживает вращение объектов вокруг определенной точки. Вместо этого, зубчики шестерни рисуются и перерисовываются вокруг центра. Они летают, вместо вращения. Единственным способом заставить шестерню вращаться, является создание единого объекта, как след (path), но это занимает больше времени на разработку, нежели мой способ.

Если кто-то хочет посмотреть, как это работает, кликните на картинки, все это открытый ресурс на GitHub.

На что полагаться? – будущее веб рисования

Углубляясь в каждую новую технологию, мы надеемся, что из нее будет выгода. Технологии подымаются и падают, по мере того, насколько ими пользуются.

Сейчас, Flash выглядит как плохая инвестиция. Flash имеет ряд достойных инструментов, им пользуются годами, но даже Adobe уходит от этой технологии.

SVG в той же ситуации. Браузеры поддерживают его сейчас, но он не получает должного внимания.

Каждый браузерный вендор работает над быстрой обработкой **canvas** тега. Поэтому, на мой взгляд, лучше выбрать Paper.js или Processing.js. Все мобильные устройства также поддерживают canvas, и их разработчики делают все, чтобы улучшить это.

Вот такая специфика и тенденции веб рисования на javascript. Если есть что добавить или убавить, пишем в комментарии.

Лекция 14
BOM - Объектная модель браузера

С точки зрения JavaScript браузер представляет собой набор объектов, объединенных в иерархическую структуру. Именно эта иерархия объектов, доступная для использования в сценариях, и называется **Объектной Моделью Браузера — Browser Object Model** или просто **BOM**. BOM имеет довольно сложную и разветвленную структуру, и насчитывает несколько десятков типов объектов (все зависит от типа и версии браузера). Поэтому, целью данного и нескольких последующих уроков, является знакомство с общей структурой BOM и изучение основных объектов этой модели.

BOM (Browser Object Model / объектная модель браузера) - набор объектов, описывающих содержимое документа. BOM уникальна для каждого браузера, что вносит проблемы при создании межбраузерных приложений. Поэтому Веб-консорциум предложил объектную модель документа (DOM), являющуюся стандартным способом представления веб-страниц с помощью набора объектов. В отличие от DOM объектная модель браузера (BOM) содержит набор объектов связанных непосредственно с браузером, объектов позволяющих управлять окнами (открывать, перемещать, менять размер, закрывать), строкой состояния браузера, cookie-наборами и т.п., которых нет в DOM. При написании приложений в целях поддержки межбраузерной переносимости необходимо придерживаться стандартов DOM, а к BOM прибегать лишь при крайней необходимости. Такая необходимость может возникнуть, например, при управлении окнами, строкой состояния, при необходимости узнать разрешение экрана пользователя (объект screen), или информацию о браузере (объект navigator) или данные о документе (объект location) и т.п.

Сценарии, не зависящие от браузера

Как упоминалось выше, набор объектов и их свойств сильно зависит от типа и версии браузера. Одни объекты доступны не во всех браузерах, а другие обладают разными свойствами и методами в разных браузерах. Эта «обратная сторона медали» и является основной причиной «головной боли» многих web-разработчиков.

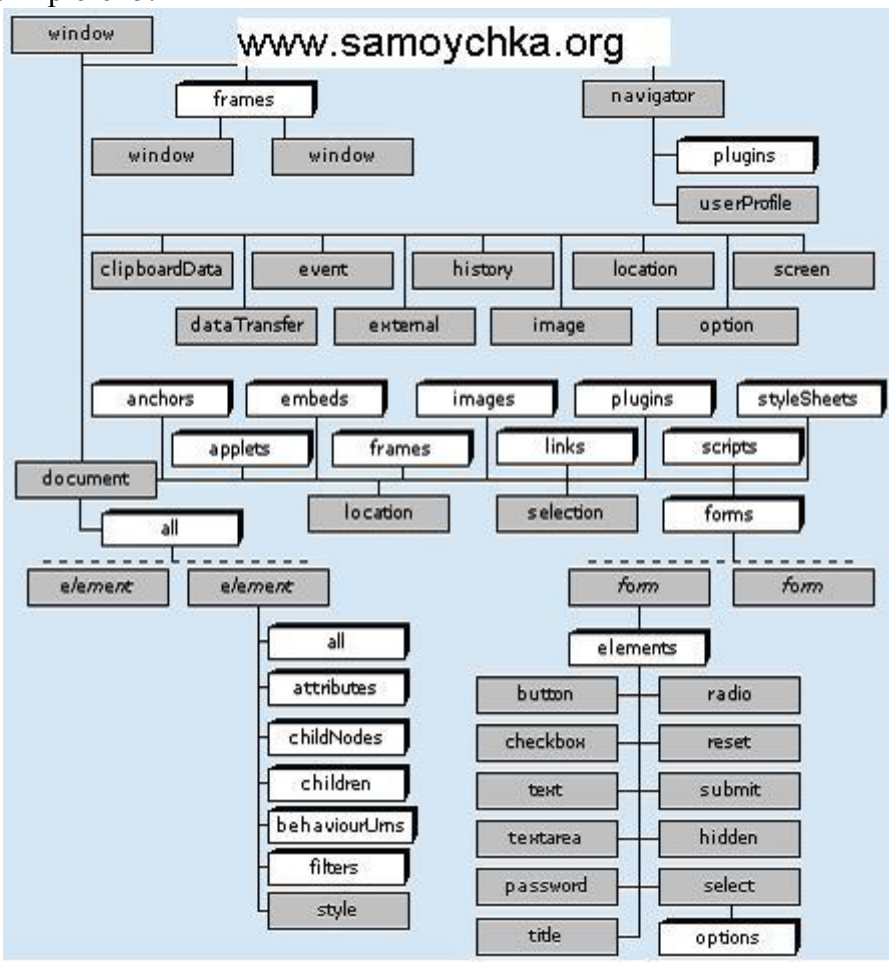
Однако, в этом и кроется парадокс: по наличию или отсутствию того или иного элемента объектной модели можно достаточно точно судить о типе и версии используемого браузера. Например, браузер MS Internet Explorer (IE) обладает коллекцией document.all, содержащей все элементы страницы, в то время, как браузер Netscape Navigatior (NN) этой коллекции не содержит. В то же время, браузер NN обладает уникальной для него коллекцией document.layers, которая отсутствует в других браузерах. В связи с этим можно написать следующий сценарий, определяющий тип браузера, с которым работает пользователь:

var isIE=false, isNN=false; // признаки типа браузера // получение информации о браузере if (document.all) // проверка на IE isIE = true; else if (document.layers) // проверка на NN isIE = true; else isIE = false; … // использование полученной информации о браузере if (isIE) { // код для MS IE } else if (isNN){ // код для NN } else { // код для других браузеров }

К вопросу определения типа и версии браузера мы еще вернемся в нашем уроке. А сейчас вашему вниманию предлагается на рассмотрение объектная модель браузера Microsoft Internet Explorer 5.

Объектная модель браузера Microsoft Internet Explorer 5

Ниже приводится графическое представление объектной модели браузера Microsoft Internet Explorer 5.



Некоторые объекты этой модели, такие как, document или event нам уже в той или иной степени знакомы. В данном уроке мы освоим новые объекты и свяжем их в единое целое.

Рассмотрим основные элементы объектной модели и их назначение.

Табл.1

Основные элементы BOM MSIE 5 объект Пояснение

window Глобальный объект, связанный с окном браузера или с фреймом (кадром), в который загружен документ.

document Представляет собой непосредственно страницу.

location Позволяет получать различную информацию об адресе, с которого загружена страница.

history Ведет хронологию (историю) посещения страниц. Позволяет перемещаться по ним в любом направлении (назад или вперед).

navigator Содержит различную информацию о браузере и окружении клиента (тип и номер версии браузера, операционная система и пр.)

screen Позволяет получать информацию графической подсистеме клиента (разрешение, глубина цвета и т.п.)

Лекция 15
DOM - ОБЪЕКТНАЯ МОДЕЛЬ ДОКУМЕНТА

Объектная модель документа (Document Object Model – DOM) является стандартом, предложенным веб-консорциумом, и регламентирует способ представления содержимого документа (в частности веб-страницы) в виде набора объектов. Под содержимым понимается все, что может находиться на веб-странице: рисунки, ссылки, абзацы, текст и т. д.

В отличие от объектной модели браузера (BOM), которая уникальна для каждого браузера, объектная модель документа является стандартом и должна поддерживаться всеми браузерами. И хотя на практике поддержка DOM реализована не в полной мере, тем не менее необходимо стремиться следовать требованиям этого стандарта как производителям браузеров, так и разработчикам веб-сайтов.

Следует заметить, что DOM может применяться не только в веб-страницах, но и к любым других документам. В частности, она может использоваться с любыми словарями XML, причем одним из таких словарей является HTML, а точнее, XHTML.

DOM является развивающимся стандартом и разбит на три уровня. Первый уровень является первой версией стандарта и пока что единственной законченной. Он состоит из двух разделов: первый является ядром и определяет принципы манипуляции со структурой документа (генерация и навигация), а второй посвящен представлению в DOM элементов HTML, определяемых одноименными тегами.

Второй и третий уровни описывают модель событий, дополняют таблицы стилей, проходы по структуре.

Представление документа в виде древовидной структуры

В DOM документ представляется в виде древовидной структуры (рис. 1), являющейся одной из наиболее употребительных структур в программировании. Это обеспечивает унифицированный способ навигации по документу.

```
<html>
<head>
      <tytle>Пример представления HTML-документа в виде дерева</tytle>
<head>
<body>
       <h1>Представление документа в виде дерева</h1>
       <p1>Абзац 1</p>
       <p1>Абзац 1</p>
</body>
</html>
```

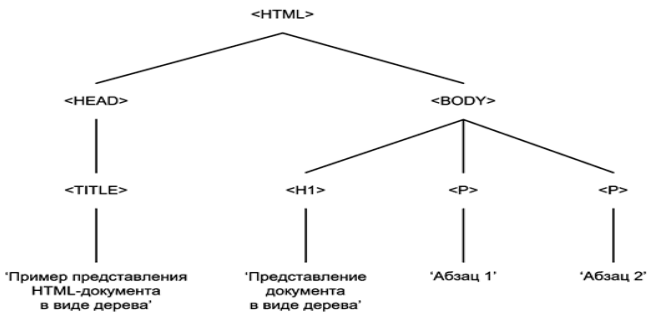


Рис. 1. Представление HTML-документа в виде древовидной структуры

Навигация по документу

В модели DOM к элементу можно обратиться непосредственно по его идентификатору id, воспользовавшись методом getElementById объекта Document:

```
<html>
<head>
<title>Основы DOM</title>
</head>
<body>
<h1 id = "head">Основы DOM</h1>
<p>A Text</p>
<script language = "JavaScript">
var a = document.getElementById("head");
alert(a);
</script>
</body>
</html>
```

Для получения коллекции всех элементов, соответствующих какому-либо тегу, используется метод объекта Document – getElementsByTagName. Например, var a = document.getElementsByTagName("TD") присвоит переменной a коллекцию всех элементов <td>. Обратите внимание, что имя элемента следует писать прописными буквами ("TD"). Рассмотрим пример использования метода getElementsByTagName:

```
<html>
<head>
<title>Основы DOM</title>
</head>
<body>
<h1 id = "head">Основы DOM</h1>
<table border = "2">
<tr>
<td>1,1</td>
<td>1,2</td>
</tr>
<tr>
<td>2,1</td>
<td>2,2</td>
</tr>
</table>
<script language = "JavaScript">
var a = document.getElementsByTagName("TD");
a.item(0).style.color = "red";
a.item(3).style.fontFamily = "arial";
a.item(3).style.color = "green";
</script>
</body>
</html>
```

Чтобы воспользоваться преимуществом древовидной структуры, принятой в DOM для представления документа, следует использовать навигационные атрибуты (рис. 2), представленные в табл. 1.

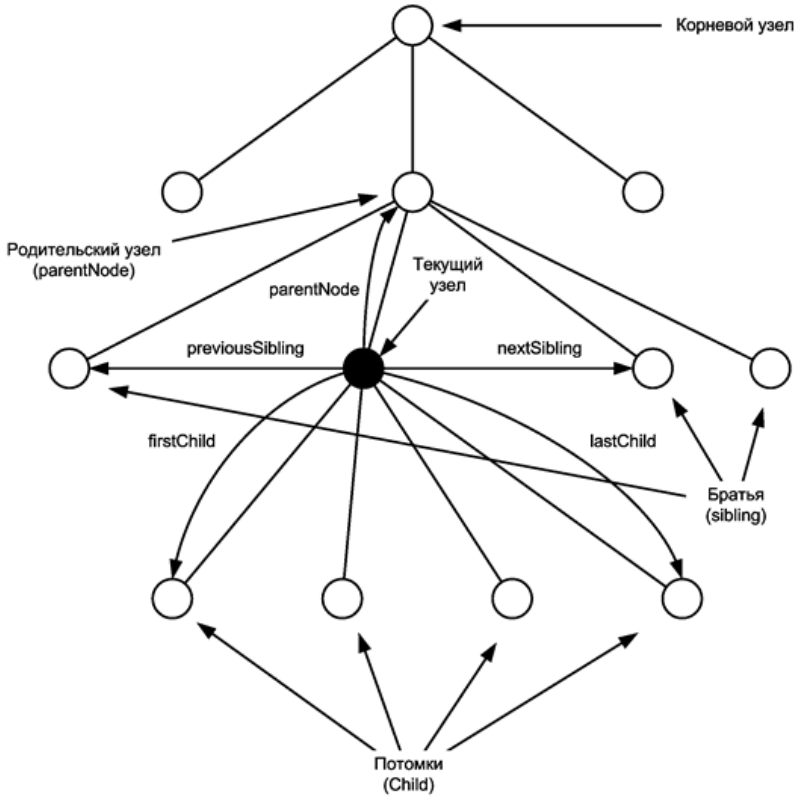


Рис. 2. Навигационные атрибуты объекта Node

Таблица 1

Навигационные атрибуты объекта Node

Атрибут	Описание
firstChild	Возвращает первый узел-потомок
lastChild	Возвращает последний узел-потомок
previousSibling	Возвращает предыдущий соседний узел, имеющий с текущим одного родителя
nextSibling	Возвращает следующий соседний узел, имеющий с текущим одного родителя
parentNode	Возвращает родительский узел
ownerDocument	Возвращает корневой узел документа, содержащий текущей узел
nodeName	Возвращает имя узла
nodeValue	Возвращает значение узла в текстовом формате
nodeType	Возвращает тип узла в виде числа

В следующем примере осуществляется проход по древовидной структуре документа:

```
<html>
<head>
<title>Навигация по документу</title>
</head>
<body>
<h1>Изучение навигации по документу</h1>
<p>Абзац 1</p>
```

```
<p>Абзац 2</p>
<script language = "JavaScript">
var temp = document.documentElement;
temp = temp.firstChild;
alert(temp.tagName);
if(temp.nextSibling == 3)
temp = temp.nextSibling.next.Sibling;
else
temp = temp.nextSibling;
alert(temp.tagName);
temp = temp.firstChild;
alert(temp.tagName);
temp.style.color = "red";
if(temp.nextSibling == 3)
temp = temp.nextSibling.nextSibling;
else
temp = temp.nextSibling;
alert(temp.tagName);
temp.style.color = "blue";
temp = temp.parentNode;
alert(temp.tagName);
</script>
</body>
</html>
```

Лекция 16

Характеристика программного средства, его назначение и возможности.

Объектная модель документа (DOM) обеспечивает программный интерфейс (API) для HTML и XML-документов. Она определяет логическую структуру документов и способы взаимодействия с ними.

При помощи DOM можно создавать новые документы и изменять существующие путем добавления, изменения и удаления отдельных элементов. Последнее применение более известно под названием Dynamic HTML (DHTML, динамический HTML) хотя на самом деле, возможности DOM этим не ограничиваются, поскольку с тем же успехом можно манипулировать и XML, и XHTML-документами, а также различными таблицами стилей (CSS, XSL) – была бы только поддержка со стороны программы просмотра.

По своей сути объектная модель документа – это не более чем отображение web-страницы во внутреннем представлении браузера, и сама по себе она ничего не значит. Но при помощи сценариев на JavaScript можно взаимодействовать с объектной моделью, изменяя тем самым внешнее представление и даже содержание документа. При помощи DOM сценарии получают доступ ко всем объектам документа, могут отслеживать события и оперативно изменять содержимое.

История DOM и совместимость

Впервые модель объектов HTML появилась в браузере Netscape 2.0. Изначально она позволяла манипулировать лишь минимальным количеством параметров документа с помощью JavaScript. При разработке MSIE 3.0 компания Microsoft взяла за основу именно эту модель. В разрабатываемом параллельно с ним Netscape 3 были добавлены новые объекты applets и images. Впоследствии данную модель приняли за основу построения дальнейших стандартов, и, как нам уже известно, условно окрестив ее DOM уровня 0. Однако говорить об истинно динамическом HTML было рано вплоть до выхода 4-й версии Internet Explorer. Именно в нем впервые была реализована объектная модель, позволявшая получить доступ к любому элементу документа. С некоторыми оговорками на перманентное нежелание Microsoft следовать каким-либо иным стандартам, кроме

своих собственных, заложенную в MSIE 4 объектную модель можно сопоставить с DOM уровня 1.

В то же время, 4-я версия Netscape продолжала плавный эволюционный подход к расширению объектной модели, ограничившись введением слоев (layers) и дальнейшим наращиванием возможностей языка JavaScript. Неэффективность столь мягкого подхода выразилась в том, что в «рейтинге популярности» браузеры поменялись местами. Можно, конечно, долго спорить относительно монополизма Microsoft на рынке операционных систем, но, с точки зрения рядового разработчика, появившаяся всего на несколько месяцев позже конкурента, 4-я версия Internet Explorer предлагала больше новых возможностей, нежели Netscape. Со временем ситуация вновь изменилась на противоположную: новые версии браузеров, основанные на ядре Gecko, предоставляют более широкие возможности для взаимодействия с документом, нежели чем медленно эволюционирующий MSIE: Mozilla Suite 1.0, вышедший примерно в одно время с MSIE 6, в целом, поддерживал DOM уровня 2, в то время, как Microsoft лишь продолжала «допиливать» компромиссные основы, заложенные в MSIE 4.

Впрочем, помимо объектных моделей документа, предложенных Netscape и Microsoft, имеется также вариант W3C. Причем, как уже было отмечено ранее, существуют несколько их вариантов:

- DOM Level 0 (объектная модель документа, уровень 0) – «неписаный стандарт», основанный, в основном, на ранних версиях Netscape;
- DOM Level 1 (объектная модель документа, уровень 1) – разработчики браузеров еще на момент выхода этого стандарта заверяли, что их модели «почти совместимы» со стандартами W3C, но на практике это не совсем верно. Тем не менее, современные версии браузеров полностью реализуют DOM 1;
- DOM Level 2 (объектная модель документа, уровень 2) – если DOM 1 определял только ядро объектной модели и ее HTML-часть, то начиная с этого уровня модель документа разделена на ряд составляющих: ядро, стили, события и т.д. Поддержку DOM 2-го уровня на сегодня имеют, помимо Firefox и SeaMonkey, браузеры Chrome/Safari и Opera, начиная с версии 8.0. MSIE смог подтянуться только с выходом версии 9.0.
- DOM level 3 (объектная модель документа, уровень 3) – в этом стандарте дальнейшее развитие получило ядро DOM 2, а также больше внимания уделено работе с XML (для чего был создан специальный стандарт «DOM level 3 Load & Save»). Поддержку ядра DOM 3 реализуют все встречающиеся на практике браузеры, кроме MSIE 6-8.

Учитывая все вышеизложенное, довольно трудно представить себе, как же удается создавать динамические документы, работающие во всех браузерах, включая различные «ответвления» от основного пути развития DOM, реализованные в MSIE 4.0/5.0. К счастью, в MSIE 6, реализация объектной модели расширена для совместимости с «официальным» DOM уровня 1, а браузеры Mozilla, Opera и Chrome изначально разрабатывались на основе стандартов W3C. Таким образом, констатируем тот факт, что для обеспечения максимальной совместимости следует использовать DOM уровня 1, обращаясь к более современным версиям лишь для выполнения задач, которые могут помочь улучшить функционирование сценариев во второстепенных, или специфических для конкретного браузера, случаях.

Иерархия объектов в браузерах

Прежде, чем приступать к созданию совместимых со всем на свете динамических страниц, необходимо изучить объектную модель браузеров, для чего не помешает заглянуть немного в прошлое. Так, иерархия объектов для четвертых версий MSIE и Netscape, при всех их различиях, имела достаточно много сходных черт. В частности, ряд свойств основного объекта – window, а так же наиболее востребованного при написании сценариев объекта – document, у них во многом совпадаю.

ПРИМЕЧАНИЕ

Отметим, что document, а так же другие объекты в DOM на самом деле являются не

объектами, а интерфейсами, т.е. тем, что осуществляет взаимодействие между сценариями и собственно объектами в документе.

Суть отличий состоит в том, что MSIE позволяет получить доступ из JavaScript-программы к любому элементу HTML-документа (через свойство document.all), а Netscape 4 – только к некоторым. Если же ограничиться только совпадающими у обеих программ свойствами (для объекта document), то мы получим как раз пресловутый DOM уровня 0, совместимый с этими и всеми последующими версиями браузеров. Собственно, все совместимые с DOM 0 свойства (links, images, anchors, applets и forms) перешли в DOM 1 HTML и DOM 2 HTML. Кроме всех этих свойств, являющихся массивами ссылок на элементы документа того или иного типа, для элемента document предусмотрены так же свойства cookie, и методы open, close, write, writeln и getElementsByName. С большинством методов мы уже знакомы по предыдущим примерам, а что касается свойств, то рассмотрим все их подробнее в таблице 1

Таблица 1. Свойства document

Свойство	Примечание	Описание
all	Только MSIE, Opera 6	Нестандартное свойство для доступа к любому объекту на странице
styleSheets	DOM 2 Styles	Ссылается на связанные с документом таблицы стилей
scripts	Только MSIE	Список внедренных в документ сценариев
selection	Только MSIE	Ссылается на выделенный фрагмент в документе. В DOM 2 для этого предназначены методы Range
frames	Только MSIE	Список элементов IFRAME
layers	Только Netscape 4	Список элементов LAYER
embeds	Нестандартный, но поддерживается всеми	Список элементов EMBED
applets	DOM 0, DOM 1, 2 HTML	Список Java-аплетов (элементов APPLET)
links	DOM 0, DOM 1, 2 HTML	Список ссылок (элементов A с атрибутом HREF)
anchors	DOM 0, DOM 1, 2 HTML	Список якорей (элементов A с атрибутом NAME)
images	DOM 0, DOM 1, 2 HTML	Список изображений (элементов IMG)
forms	DOM 0, DOM 1, 2 HTML	Список форм (элементов FORM)
title	DOM 1, 2 HTML	Ссылается на заголовок (содержимое элемента TITLE) документа
referrer	DOM 1, 2 HTML	Содержит URL страницы, ссылающейся на данную (т.е. с которой пользователь попал на текущую)
domain	DOM 1, 2 HTML	Доменное имя сервера, на котором находится данный документ
URL	DOM 1, 2 HTML	Адрес (URL) данного документа
body	DOM 1, 2 HTML	Тип элемента, содержащего данный документ (BODY или FRAMESET)
cookie	DOM 1, 2 HTML	Содержит cookie, в виде пар «имя=значение», разделенных точкой с запятой

Данная таблица дает представление о том, какие свойства документа доступны в тех или иных браузерах. Но на практике, если вы просмотрите все свойства объекта document в том или ином браузере (например, можно использовать для этого сценарий из файла Part_4\statements\for_in.html), то убедитесь, что их гораздо больше, особенно в последних версиях, поддерживающих DOM 3. Связано это с тем, что помимо унаследованных от DOM 0 свойств (forms, images), вы увидите все иные свойства и методы, связанные с интерфейсом document, причем относиться они могут к ядру DOM, его HTML-части, а так же к модели событий, XML и т.д.

Поскольку настоящее издание не являются справочником по объектной модели, то по ходу дальнейшего ознакомления с DOM мы ограничимся только теми свойствами и

методами, которые часто применяются на практике. Полное же изучение стандарта DOM имеет скорее академический интерес, нежели практический: помимо проблем с реализацией стандарта W3C в браузерах, необходимо учитывать и тот факт, что пока HTML не используется как основа для построения полноценных интерфейсов (скажем, наподобие окон Windows), многие методы остаются невостребованными.

> СОВЕТ
>
> Ряд методов в DOM 2 или 3 ориентирован на XML, который в данной книге не рассматривается. Если вы заинтересованы в детальном изучении объектной модели документа, то можете обратиться к спецификациям DOM 1, DOM 2 и DOM 3, которые можно найти на сайте w3c.org. Кроме того, среди примеров, в каталоге Samples\Part_4\, находится пример inspector.html, позволяющий исследовать свойства объектов navigator, window и document любого браузера.

Доступ к объектам документа из JavaScript-программы

Знания самой объектной модели недостаточно для того, чтобы работать с ней, поскольку требуется также и знание средств конкретно применяемого языка программирования. Поскольку в данной книге рассматривается JavaScript, то и взаимодействие с DOM рассматривается в контексте реализации этого языка.

Как вы уже знаете, в JavaScript используется доступ к объектам через точку. Таким образом, для доступа, например, к параграфу, имеющему ID=”My1”, требуется написать:

document.getElementByID("MyP1");

Возможно, это не совсем удачный пример, т.к. он не иллюстрирует «классического» варианта доступа (доставшегося в наследство от ранних версий Netscape, или DOM 0, поэтому рассмотрим более универсальный пример – получение доступа к элементам массива images в HTML-документе. Во всех встречающихся сейчас версиях браузеров поддерживается такой синтаксис:

document.images[i];

Например, чтобы заменить 5-ю по счету в документе картинку, можно написать:

document.images[4].src="new_file.gif"; // первый элемент массива - 0

Давайте теперь создадим документ, в котором будет изображение, изменяющееся в зависимости от того, где пользователь провел указатель мышки (листинг 4.7).

Листинг 4.7. Смена картинки при помощи document.images

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"> <html> <head> <title>DOM 0 и массив document.images[]</title> <script type="text/JavaScript"><!-- //функция для смены картинки function change_image(i) { document.images[4].src=document.images[i].src; } //--></script> </head> <body> <p>Подведите мышку к цифре:<br /> <!--images[0]--> <img src="dom0/d0.gif" alt="0" onmouseover="change_image(0)" /> <!--images[1]--> <img src="dom0/d1.gif" alt="1" onmouseover="change_image(1)" /> <!--images[2]--> <img src="dom0/d2.gif" alt="2" onmouseover="change_image(2)" /> <!--images[3]--> <img src="dom0/d3.gif" alt="3" onmouseover="change_image(3)" /> </p> <hr /> <p>В последний раз мышка была над цифрой:<br /> <!--images[4]--> <img src="dom0/d0.gif" alt="?" /> </body> </html>
```

Здесь в сценарии, расположенном в заголовке документа (сценарии, которые ничего не выводят в документ, предпочтительно располагать именно в части HEAD), определена функция change_image, принимающая в качестве параметра число. Этим числом является порядковый номер изображения в документе, в данном случае это будет одно из 4-х значений – от 0 до 3. Функция заменяет изображение пятой картинки – document.images[4] – на такое же, как и у той картинки, над которой находится указатель мышки

Путем небольшой доработки можно изменить эту функцию таким образом, чтобы она показывала еще и предыдущую позицию. Для этого, во-первых, понадобится добавить еще одну картинку (6-ю), а во-вторых – модернизировать функцию таким образом, чтобы она не только меняла второе изображение, но и запоминала последнее значение при помощи глобальной переменной:

```
//глобальная(!) переменная для хранения предыдущего значения var old_img=0; function change_image(i) { //Отображаем текущее значение
```

document.images[4].src=document.images[i].src; //Отображаем предыдущее значение document.images[5].src=document.images[old_img].src; //Заменяем хранимое предудущее значение на текущее old_img=i; }

Полностью модернизированный пример продемонстрирован в файле dom0.html, а в примере dom1.html приведен один из вариантов использования более прогрессивных методов DOM 1. В частности, в нем создается массив, содержащий все изображения (т.е. все элементы IMG) при помощи метода getElementsByTagName, определенного в ядре DOM для интерфейса document:

images = document.getElementsByTagName("img");

Такой синтаксис является более удобным с точки зрения своей универсальности: если вместо «img» указать любой другой элемент, скажем, P, то мы получим массив со всеми элементами P, имеющимися в данном документе. Дальнейшие манипуляции также могут быть произведены над любым атрибутом элемента, а не только на предусмотренные в DOM 0 для массива images отдельные свойства. Делать это можно при помощи таких методов интерфейса element, как getAttribute и setAttribute.

Помимо выборки элементов по типу, DOM позволяет обращаться к конкретным элементам документа по их идентификатору. Если обратиться к унаследованным от DOM 0 и HTML 2.0/3.2 методам, то для доступа к элементу можно использовать метод getElementsByName, который использует атрибут NAME для идентификации HTML-элемента на странице. При этом, поскольку значения атрибута NAME, в принципе, могут дублироваться, то данный метод не гарантирует выбор конкретного элемента, а возвращает массив.

> ВНИМАНИЕ
>
> Тут следует сделать замечание, что в XHTML 1.0 метод getElementsByName работает только с элементами форм, а начиная с XHTML 1.1 такой метод вообще не приемлем ввиду отсутствия атрибута NAME. Поддержка этого метода может быть прекращена в новых версиях браузеров.

Более удобным и часто применяемым на практике методом интерфейса document является метод getElementById, который возвращает ссылку на элемент документа с указанным идентификатором ID, который, как известно, должен быть уникальным в рамках одного документа. При помощи данного метода можно получить доступ к любому именованному (т.е. имеющему атрибут ID) элементу документа:

Абзац P с ID и TITLE ... <script type="text/JavaScript"><!-- var MyP = document.getElementById("Par1"); alert(MyP.getAttribute("title")); //--></script>

Этот и несколько других вариантов использования методов для доступа к элементам документа приведены в файле getxxx.html.

Лекция 17
Динамическая генерация веб-страниц средствами DHTML на основе DOM

Иногда требуется динамически формировать веб-страницы, например, в случае создания чатов, форумов, либо динамически создаваемых веб-страниц, содержимое которых хранится в базе данных. DOM позволяет решить такую задачу.

Для создания объектов у объекта Document имеются следующие методы (табл. 12):

Таблица 12

Методы объекта Document, позволяющие создавать объекты

Метод	Описание
createElement(имя_элемента)	Создает новый узел элемента с указанным именем
createTextNode(текст)	Создает текстовый узел с указанным текстом
createAttribute(имя_атрибута)	Создает новый узел атрибута с указанным именем

Вновь созданные объекты добавляются в структуру документа при помощи методов объекта Node (табл. 13):

Таблица 13

Методы объекта Node, добавляющие и удаляющие элементы документа

Метод	Описание
appendChild(новый_узел)	Добавляет объект Node в конец списка узлов-потомков
cloneNode(потомок-опция)	Создает объект Node, идентичный указанному в аргументе. В качестве аргумента можно использовать и все узлы-потомки одновременно
hasChildNodes()	Возвращает true, если узел имеет потомков
insertBefore(новый_узел, текущий_узел)	Вставляет объект Node в список потомков перед узлом, указанным в качестве второго параметра
removeChild(узел-потомок)	Удаляет узел-потомок, указанный в качестве параметра
replaceChild(новый_потомок, старый_потомок)	Заменяет старого потомка на нового

Приведем пример динамической генерации документа средствами DOM (рис. 21).

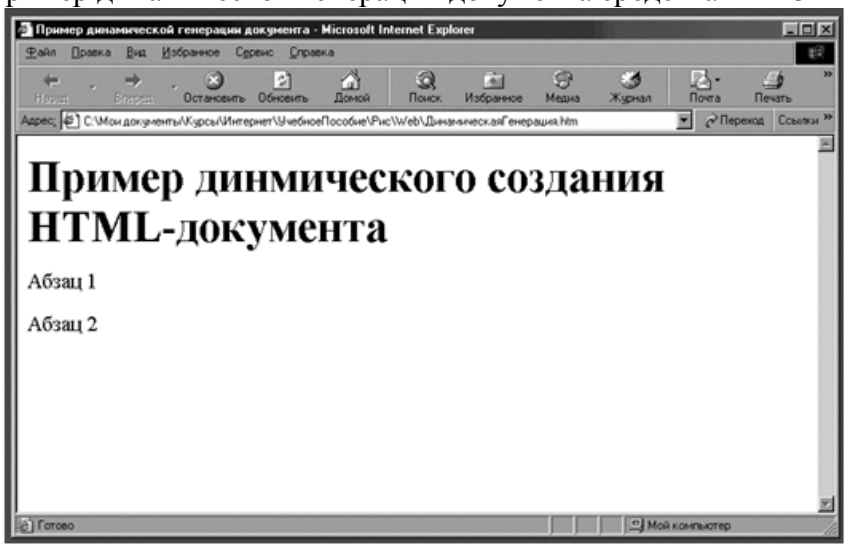


Рис. 21. Динамически сгенерированная веб-страница

```
<html>
<head>
<title>Пример динамической генерации документа</title>
</head>
<body>
<script language = "JavaScript">
var newText;
var newElem;
newText = document.createTextNode("Пример
динмического создания HTML-документа");
newElem = document.createElement("H1");
newElem.appendChild(newText);
```

```
document.body.appendChild(newElem);
newText = document.createTextNode("Абзац");
newElem = document.createElement("P");
newElem.appendChild(newText);
document.body.appendChild(newElem);
</script>
</body>
</html>
```

Для чтения и установки атрибутов используются следующие методы объекта Element (табл. 14).

Таблица 14

Методы объекта Element

Метод	Описание
getAttribute(имя_атрибута)	Возвращает значение атрибута
setAttribute(имя_атрибута, значение)	Устанавливает значение атрибута
removeAttribute(имя_атрибута)	Устанавливает значение атрибута по умолчанию, затирая текущее значение

Ниже приведен пример на задание атрибутов. И хотя применение атрибутов физического форматирования не рекомендовано к применению (для этих целей используются стили), они были выбраны в качестве примера, так как наглядно демонстрируют идею задания атрибутов методами DOM.

```
<html>
<head>
<title>Пример динамического создания HTML-документа</title>
</head>
<body>
<script language = "JavaScript">
var newText;
var newElem;
newText = document.createTextNode("Пример динамического
создания HTML-документа");
newElem = document.createElement("H1");
newElem.appendChild(newText);
newElem.setAttribute("align", "center");
document.body.appendChild(newElem);
alert(newElem.getAttribute("align"));
newText = document.createTextNode("Абзац");
newElem = document.createElement("P");
newElem.appendChild(newText);
newElem.setAttribute("align", "right");
document.body.appendChild(newElem);
alert(newElem.getAttribute("align"));
newElem.removeAttribute("align");
</script>
</body>
</html>
```

Модель событий DOM

Особенностью программ, создаваемых для среды веб является то, что они управляются событиями. Чтобы узнать, какое событие произошло, в DOM имеется объект

события event(табл. 15). Объект event является локальным и его следует явным образом передавать в обработчик события.

Таблица 15

Свойства объекта event

Свойство	Описание
bubbles	Указывает возможность «всплывания» события (передачи управления вверх по иерархической структуре)
cancelable	Указывает возможность отмены действия события, заданного по умолчанию
currentTarget	Указывает событие, обрабатываемое в данный момент
eventPhase	Указывает фазу возбуждения события
target (только NN 6)	Указывает элемент, вызвавший событие
timestamp (только NN 6)	Указывает время возникновения события
type	Указывает имя события

События, связанные с мышью, генерируют объект mouse (табл. 16).

Приведем пример динамически изменяемого текста.

```
<html>
<head>
<script language = "JavaScript">
function onMouseover()
{
var temp = document.getElementById("DynamicText");
temp.firstChild.nodeValue = "Указатель мыши на тексте";
temp.style.color = "red";
}
function onMouseout()
{
var temp = document.getElementById("DynamicText");
temp.firstChild.nodeValue = "Динамический текст";
temp.style.color = "black";
}
</script>
<title>Динамический текст</title>
</head>
<body>
<p id = "DynamicText" onmouseover = "return onMouseover()"
onmouseout = "return onMouseout()">Динамический текст</p>
</body>
<html>
```

Таблица 16

Свойства объекта mouse

Свойство	Описание
altKey	Указывает, была ли нажата клавиша <Alt> в момент возникновения события

button	Указывает, какая клавиша мыши была нажата
clientX	Сообщает горизонтальную координату указателя мыши в окне браузера на момент возникновения события
clientY	Сообщает вертикальную координату указателя мыши в окне браузера на момент возникновения события
ctrKey	Указывает, была ли нажата клавиша <Ctrl> в момент возникновения события
metaKey	Указывает, была ли нажата метаклавиша в момент возникновения события
relatedTarget (только NN 6)	Указывает цель события
screenX	Сообщает горизонтальную координату указателя мыши в окне браузера, вычисленную от начала экранных координат, на момент возникновения события
screenY	Сообщает вертикальную координату указателя мыши в окне браузера, вычисленную от начала экранных координат, на момент возникновения события
shiftKey	Указывает, была ли нажата клавиша <Shift> в момент возникновения события

Пример, определяющий координаты нахождения курсора мыши.

```
<html>
<head>
<title>Определение координат курсора</title>
<script language = "JavaScript">
function onClick(e)
{
alert("X = " + e.clientX + "; " + "Y = " + e.clientY);
}
</script>
</head>
<body onclick = "onClick(event)">
<p>Щелкните мышью на экране<p>
</body>
</html>
```

Лекция 18
Современные автоматизированные системы веб-разработки и управления контентом (CMS).

Что такое CMS

Аббревиатура CMS расшифровывается как «Content Management Software» («программное обеспечение для управления содержимым»). В нашей стране принято последнюю букву «S» расшифровывать как «System», а по-русски это обычно звучит как «Система управления контентом». Английское слово content означает «нечто, содержащееся внутри» и применительно к письменным работам обычно входит в словосочетание table of contents — содержание, перечень разделов (скажем, книги). Отличительная черта контента состоит в том, что он конструируется из отдельных

кусочков -графика, документы (в том числе отчеты, ведомости и.т.д), звуковые и видео-файлы. Иногда употребляется более простое название - "движок сайта".

CMS появились не так давно. Первой системой принято считать Vignette, которая появилась на западе в 1995 году. В нашей стране решения по управлению контентом появились значительно позже.

История управления контентом началась с управления документами в классическом понимании этого слова - текстовыми файлами. По мере развития понятия «документ», системы управления документами стали называть системами управления контентом, подчёркивая способность таких систем управлять информацией независимо от формы ее представления, а также отделить информацию-контент от документа-формы. Однако абстрактно управлять информацией невозможно — она обязательно должна быть представлена в какой-либо форме. Пытаясь управлять контентом, мы неизбежно приходим к управлению документами. Системы управления контентом, действительно, «научились» разделять управление документами (хранение, изменение и т.п.) и их представление конечному пользователю. Но они все-таки управляют документами в какой-то форме, а не информацией.

Само понятие «управление контентом» первоначально прочно ассоциировалось с процессом публикации и обновления информации на Web-сайтах — требовалась технология, позволяющая следить за ее актуальностью. Поэтому в качестве синонима content management часто используют термин Web content management. В результате термин content management расширился: им стали обозначать управление не только информацией на сайте, но и всеми разрозненными и разнообразными фрагментами корпоративной информации. Есть и другие определения. Скажем, в энциклопедии Wikipedia системой управления контентом названа система, применяемая для организации и упрощения совместного создания содержимого.

Зачем нужны CMS

Необходимость систем управления для владельцев сайтов начала проявляться в тот момент, когда количество материалов на веб-сайтах начало стремительно расти. Это привело к тому, что традиционные «ручные» технологии разработки и поддержки сайтов, когда сайт состоял из статических страниц и набора дополнительных специализированных скриптов, стали не успевать за быстро меняющимися условиями бизнеса. Ввод данных на сайт требовал (как минимум) знания технологий HTML/CSS верстки, изменения структуры сайтов были сопряжены с каскадным изменением большого количества взаимосвязанных страниц. Различные автоматизированные механизмы, вроде гостевых книг и новостных лент, внедренные на сайтах как отдельные скрипты и, как правило, написанные разными специалистами, перестали удовлетворять требованиям безопасности. На многих сайтах стали появляться коктейли из разных технологий и подходов к разработке, поэтому возникла потребность в стандартизации программных решений, в разделении дизайна и содержимого на две независимые составляющие. CMS действительно разделяют сайты на две составляющие: дизайн (внешний вид сайта в целом, отдельных страниц, конкретных блоков информации) и контент. Дизайн сайта, как правило «зашит» в шаблоны и изменяется значительно реже, чем контент.

CMS открывают изобилие технических возможностей в создании динамического веб-ресурса. Все серьезные сайты, содержащие большой объем информации и требующие постоянного ее обновления, используют системы обновления. Это и поисковые машины, и новостные серверы, и разнообразные каталоги. С помощью данных систем можно с легкостью добавлять разделы, размещать иллюстрации, управлять рассылками, публиковать закрытую информацию, доступ к которой есть только у определенных групп пользователей. И это лишь небольшой список всего того, чего можно добиться с помощью CMS.

Анализ основных функций современных систем управления сайтами

Система управления сайтами – это программный комплекс, позволяющий автоматизировать процесс управления как сайтом в целом, так и сущностями в рамках сайта: макетами страниц, шаблонами вывода данных, структурой, информационным

наполнением, пользователями и правами доступа, а также по возможности предоставляющий дополнительные сервисы: списки рассылки, ведение статистики, поиск, средства взаимодействия с пользователями и т. д. Обычно системы обновления делятся на две части: внешнюю – набор HTML-страниц, генерируемых при вызове страниц из браузера посетителя сайта и внутреннюю – систему администрирования. Обе части обычно используют общее хранилище данных, в роли которого, как правило, выступает реляционная база данных (иногда встречаются другие виды хранилищ, например XML-документы или даже текстовые файлы).

В хранилище помещается информация, содержащаяся на сайте (собственно контент), а также информация, описывающая его (макеты страниц, структура, права доступа и пр.). При вызове страницы скрипт, который должен эту страницу вывести, в зависимости от полученных параметров выбирает из базы данных необходимую информацию (какое содержимое показать, какие ссылки поставить, как это все расположить и т.д.) и генерирует HTML-документ, который и подается браузеру. Помимо этого обязательно имеется интерфейс к базе данных, реализующий систему администрирования, которая при авторизованном доступе позволяет изменять содержание и структуру сайта.

Функции систем управления контентом можно разделить на несколько основных категорий.

Создание — предоставление авторам удобных и привычных средств создания контента.

Управление — хранение контента в едином репозитории. Это позволяет следить за версиями документов, контролировать, кто и когда их изменял, убеждаться, что каждый пользователь может изменить только тот раздел, за который он отвечает. Кроме того, обеспечивается интеграция с существующими информационными источниками и ИТ-системами. CMS поддерживает контроль за рабочим потоком документов, т.е. контроль за процессом их одобрения. Короче говоря, управление контентом включает в себя хранение, отслеживание версий, контроль за доступом, интеграцию с другими информационными системами и управление потоком документов.

Публикация — автоматическое размещение контента на терминале пользователя. Соответствующие инструменты автоматически «подгоняют» внешний вид страницы к дизайну всего сайта.

Представление — дополнительные функции, позволяющие улучшить форму представления данных; например, можно строить навигацию по структуре репозитория.

Практически все современные CMS предлагают сходные возможности, однако при желании их можно классифицировать по уровню задач, которые способны выполнять эти системы. Ниже приведены основные возможности современных систем управления сайтами, а так же их достоинства и недостатки в использовании.

CMS с возможностью кэширования информации

При каждом вызове страницы сайта под управлением движка сайтов происходит не считывание HTML-страницы с жесткого диска сервера, а вызов скрипта, который, в свою очередь, может подключать другие скрипты, делать запросы к базе данных и пр. Все это дает определенную нагрузку на сервер. А чем больше загрузка, тем быстрее она достигнет критической отметки, тем меньше производительность сервера. Поэтому иногда страница генерируется не в момент запроса URL, а в момент ее обновления, после чего выкладывается на жесткий диск или в базу данных уже в готовом виде. Этот прием достаточно спорный: например, при изменении какого-то элемента дизайна сайта придется перезаписывать все страницы. Его также нельзя применять в случае динамических ресурсов, например, при необходимости предоставления пользователю возможности проводить поиск по записям базы данных. CMS с локальными модулями администрирования

Иногда управление сайтом происходит не напрямую на сервере через веб-интерфейс, а при помощи программ, запускаемых на рабочей станции. Этот прием позволяет экономить рабочее время (особенно на плохих каналах), предоставить более широкие возможности управления сайтом (Windows-интерфейс гораздо более

функционален, чем веб-интерфейс). Однако такой подход требует гораздо более тщательной проработки безопасности, решения проблем целостности базы данных (например, если с сайтом одновременно работают несколько разработчиков.

Попробуем подытожить. Функционал CMS должен осуществлять следующие пункты:

1. Контроль прав изнутри системы. Это означает, что можно назначить пользователей, которым доступны те или иные опубликованные документы.

2. Интеграция контента. Возможность перенести готовый контент в новое решение.

3. Поддержка документов различного типа. Хранение и сортировка любых документов, включая графику, аудио и видео, в центральном репозитарии.

4. Подробная качественная документация и контекстно-интеллектуальная справка.

5. Рейтинговая система оценки статей сайта.

6. Шаблонные изменения. Общие изменения форматирования информации одной части сайта отображаются на весь сайт.

7. Настраиваемые деловые процессы. Создание своих автоматизированных деловых процессов для конкретного контента (изображений, статей и др.).

8. Маркировка документов. Возможность добавлять новые категории и маркеры к документам до и после их размещения в репозитарии.

9. Контроль версий. Создание новых версий, просмотр и возврат к предыдущим версиям документов.

10. Инструмент визуальной администрации. Позволяет авторам, не прибегая к программированию, легко управлять контентом. Обычно это реализуется с помощью HTML-форм.

Какими качествами должна обладать современная CMS

Прежде всего, давайте уясним факт, что супер-системы, подходящей на все случаи жизни не существует. Нелья объять необъятное. Тем не менее, современная качественная CMS должна обладать следующими характеристиками:

1. Простая инсталляция

Самая первая стадия эксплуатации CMS – её инсталляция. Процесс должен быть максимально задокументирован, упрощён и последователен – не стоит сразу начинать настраивать таблицы баз данных или править конфигурационные файлы. Это должна быть простая процедура, выполняемая с помощью визарда или скрипта установки. Плохо, если процесс состоит из более чем двадцати шагов - изменения конфигурационных файлов, создания и удаления папок и т.д. Скачать, распаковать и запустить мастер установки - вот к чему должна сводиться процедура.

2. Быстрый старт

CMS должна быть максимально дружелюбной к пользователю уже с первых шагов использования системы. Задачи настройки должны усложняться постепенно, идти от простого к сложному. К примеру, пользователь сначала авторизуется в системе, далее создает новый web-документ. После чего необходимо добавить несколько стилей CSS (как вариант – выбрать шаблон, тему), затем связать новый документ гиперссылками с другими web-страницами. Далее идет построение системы навигации и добавление всевозможных сервисов, подключение функциональных блоков. Работа осуществляется постепенно, команды выполняются не все сразу - возможность создания приоритета разделов, форумов, пользовательских голосований и опросов, генерации PDF «на лету» следует оставить на потом - возможно, это даже и не понадобится на первых порах работы с CMS.

3. Качественная документация

Если и когда что-то начинает идти не так, зачастую самый быстрый способ решить проблему заключается в том, чтобы почитать документацию, а не ждать помощи извне. Инструкции по установке и апгрейду должны быть настолько просты, чтобы нетехнический персонал мог понять их, но достаточно подробны, чтобы их легко было выполнять "пошагово". Большинство систем содержит инструкции по установке такого типа: «Сперва сделайте это, потом вот это, затем это и еще это». Но когда дело доходит до непосредственного использования CMS, пользователя тут же перенаправляют к изучению

документации, в которой расписана каждая возможность системы управления содержанием.

Отдельным подпунктом идёт требование с отсутствием сленговых выражений. Пользователь может попросту не понять, что такое «portlet», «module» или «snippet». Это может путать людей, сбивать их с толку.

Более того, если продукт распространяется бесплатно, с открытым кодом, следует прикладывать к CMS еще документацию для разработчика – структура программы, ее логика, алгоритм функционирования – возможно, этот продукт будут развивать и улучшать, либо же просто править под конкретные нужды.

4. Разделение администрирования и управления содержанием

Практически все открытые CMS различают понятия «администратор» и «пользователь», но пользователь не обязательно должен переключаться между режимами администрирования и управления содержанием, чтобы внести необходимые изменения.

98% пользователей CMS – это люди, которые хотят управлять содержимым своего сайта, а не самой системы. Для оставшихся 2% пока еще нет оптимизированных CMS.

5. Уникальность

Не для каждого сайта подойдет первая попавшаяся система управления содержанием. Зачастую слышно: «Да, мы попробовали PHP-Nuke, но теперь наш сайт выглядит точь-в точь, как сам PHP-Nuke, а не иначе». К сожалению, большинство открытых CMS проектируется чуть ли не по единому образу и подобию. Функциональность может различаться и позиционироваться разработчиками как преимущество, однако часто в целом один программный пакет может походить на другой, как две капли воды.

Так как CMS это лишь набор функций, то внешний вид уже должен определяется в каждом конкретном случае и зависеть от используемого шаблона. CMS должна уметь работать с любым количеством шаблонов, а структура этих шаблонов, по возможности, не должна быть жесткой и ограниченной. Многие CMS ратуют за трехколоночное представление информации на сайте с возможностью включения/отключения определенных модулей данных. Большая ошибка разработчиков CMS заключается в том, что они встраивают структуру шаблона в свой «движок». Выход есть – следует предусмотреть вывод различных текстовых блоков (например, меню, новостная лента, последние сообщения в форумах) в различные текстовые единицы, а уж форму и расположение указывать в шаблоне. CMS должна «знать» лишь один конфигурационный файл из которого она сможет получить всю информацию о шаблоне. В итоге мы получаем абсолютно любую структуру шаблона. Так мы сделаем максимально различные по виду и структуре сайты.

6. Гибкость использования. Расширяемость.

Возможность приспосабливания ко всевозможным специфичным нуждам того или иного решения, той или иной организации. Скажем так, универсальный продукт должен одинаково хорошо справляться как с ролью Интернет-блога, так и с ролью варезного сайта. Системная архитектура и принцип модульного построения позволяет гибко настраивать возможности сайта – включать в нее только необходимые заказчику в данный момент функции и компоненты; На одном «движке» должны одинаково комфортно размещаться:

Сайт-визитка. Предназначены для справочной поддержки клиентов, анонсирования информации о предлагаемых товарах и услугах. Подобные сайты позволяют фирмам добиться ощутимой экономии за счет сокращения рекламы в обычных средствах массовой информации. Персонализированные клиентские и партнерские сайты; Интернет-магазин. Сайт, представляющий собой систему торговли через Интернет, а также системы сопряжения с автоматизацией учета через БД; Информационный портал.

7. Структура данных

На сайте, обслуживаемом CMS должны отображаться документы различных типов. Другими словами, разные страницы сайта могут быть различны не только по содержанию, но и по дизайну, но и по структуре. Было бы еще лучше, если бы CMS позволяла

отображать также и страницы в форматах отличных от HTML. Например, со временем может понадобиться вывод ленты новостей в формате RSS, так что бы новости сайта были доступны для пользователей популярных ныне клиентских программ сборщиков новостей. Возможно, возникнет необходимость в WAP-версии сайта для мобильных телефонов.

CMS должна позволить нам обойтись минимальными усилиями для подобных случаев. Следует сразу же учесть, что даже если мы ограничиваемся одной версией сайта для определенного языка содержания (скажем, русского), не мешало бы быть готовыми к тому, что скоро нам потребуется версия для еще одного или более языков. Необходим удобный и быстрый доступ ко всем объектам сайта (разделы, рубрики, страницы, темы и т.п.). Большая часть разделов системы представлено в виде древовидной структуры, аналогично представлению папок на диске компьютера в проводнике Windows.

Каждая страница сайта - это документ, имеющий свой персональный интернет-адрес. А сайт представляет собой совокупностью таких документов. А из чего состоит документ? Документ составляет информация, определенным образом структурированная и оформленная. Для того, что бы сайт был более прост в восприятии ссылки на документы располагают в многоуровневых навигационных меню по степени их логической взаимозависимости. Т.е. в соответствии с определенной структурой (документной структурой). Кроме того, в этой документной структуре предусматриваются и различные категории документов, идентичных по логической архитектуре и оформлению. Таким образом, в состоянии администрирования нашей CMS нам потребуется интерфейс с перечнем категорий документов и интерфейс представляющий документную структуру сайта. Первый интерфейс будет содержать шаблоны страниц, задающие логическую архитектуру и оформление документов. Интерфейс структуры, позволит добавлять, изменять и уда лять документы, а также задавать логическую зависимость их друг от друга. Ну а то, каким образом будет задаваться документная структура CMS выбор за вами. Самое распространенное и наиболее простое решение - иерархическое дерево или решетка Бете, хорошо знакомое нам по картам сайтов. С него можно и начать, но, припоминая о наших задачах, следует сделать корневыми разделами дерева языковые версии сайта, даже если пока планируется лишь одна версия.

8. Простота и прозрачность системы.

Очень часто готовые продукты губит слишком большое, избыточное количество встроенных функций, что существенно снижает эффективность решений на их базе. Следует остановиться на самых востребованных и необходимых возможностях, остальной же функционал наращивать модулями.

Одним из основных параметров оценки пользовательского интерфейса является скорость реакции интерфейса, которая в значительной степени зависит от производительности несущего сервера, скорости соединения и прочих внешних факторов. Однако немалое значение имеет и архитектура пользовательского интерфейса. Во всех перечисленных CMS максимально сокращено число необходимых действий для завершения типовых операций. • поддерживать подключение большого количества сервисных модулей (как минимум, модулей новостной ленты, почтовой рассылки, гостевой книги, голосований, каталога товаров и поиска по сайту). Любая CMS должна иметь удобный и гибкий интерфейс. Легкая навигация и интуитивно понятный интерфейс должны позволять работать пользователю любого уровня квалификации. Для редактирования содержимого страницы необходимо присутствие редактора WYSIWYG.

9. Адаптация для SEO

Search Engine Optimization – поисковая оптимизация. Сюда входит оптимизация HTML-кода, структуры, контента сайта и внешних факторов с целью поднятия его в поисковых системах. Оптимизация и продвижение сайта представляет собой комплекс работ по повышению рейтинга сайта в поисковых системах. Помимо стандартных возможностей по ручной установке ключевых слов и заголовков для любого элемента сайта (от страницы до товара, что используют далеко не все системы), следует максимально эффективно генерирует дружественные ссылки.

Сюда же добавим mod_rewrite. Основным недостатком большинства CMS является динамическая адресация, когда ссылка имеет вид типа:**http://www.oqbo.ru/index.php?option=com_content&task=view&id=2&Itemid=3**

Такая адресация страниц позволяет легко изменять значения подставляемых переменных, что ставит под угрозу систему безопасности. Также использование динамической адресации является нежелательным для восприятия сайта поисковыми системами, поскольку не все страницы сайта проиндексируются поисковыми системами (поисковые роботы не умеют читать скрипты). В результате работы mod_rewrite происходит подмена адреса запроса, на адрес, не содержащий имён переменным:

http://www.oqbo.ru/content/view/2/3/. Mod_rewrite - основанный на правилах механизм (синтаксический анализатор с применением регулярных выражений), выполняющий URL преобразования на лету. Модуль поддерживает неограниченное количество правил и связанных с каждым правилом условий, реализуя действительно гибкий и мощный механизм управления URL. URL преобразования могут использовать разные источники данных, например переменные сервера, переменные окружения, HTTP заголовки, время и даже запросы к внешним базам данных в разных форматах, — для получения URL нужного вам вида. Подобные ссылки необходимы при жесткой конкуренции с тематическими сайтами, а так же для самостоятельного внедрения дружественных ссылок без знания PHP и затрат на поддержку. Встроеный модуль по оптимизации ключевых слов web страниц сайта. Использование этого модуля позоляет повысить эффективность индексации страницы сайта в поисковых машинах, и как следствие увеличивает посещаемость данного ресурса.

10. Поддержка продукта. Простота обновлений.

Любая система управления содержит уязвимости, и зачастую администраторы забывают про обновления системы управления, что может стать причиной взлома сайта и всего сервера. Обновления системы управления является достаточно непростой процедурой и большинство систем управления не позволяют осуществить обновление автоматически - требуется их доработка вручную, что вызывает боязнь обновлений системы. Эту проблему возможно решить только при помощи системы автоматических обновлений. В большинстве систем управления автоматические обновления осуществляются частично по запросу администратора из системы управления.

Обновление тоже может вылиться в проблему, и критерием хорошей CMS является частый выпуск легко применяемых надежных патчей или скриптов.

11. Ориентировка на web 2.0

Хорошая CMS должна давать возможность использования в контент-менеджменте всего многообразия медиа-форматов. Можно использовать собственные решения для доставки содержания этих форматов или же популярные флеш-плееры - Youtube.com для видеоподкастов, Slideshare.net для презентаций, Scribd.com для документов MS-Word и т.д. Интеграция в платформу Wiki и блоговые системы, обеспечение единой политики пользовательских прав в рамках всех приложений интранет. Пользовательский интерфейс современной CMS должен быть обогащенным, но простым в использовании. При нынешнем обилии AJAX-фреймворков "обогатить" пользовательский интерфейс популярными эффектами не представляется особо сложной задачей. Однако, чтобы достичь гарантированного баланса между эффектностью и практичностью интерфейса можно следовать по одному из двух путей. Либо придерживаться GUI-модели популярных и привычных пользователю настольных приложений, либо разрабатывать аскетичные решения в стиле Google.

12. Безопасность

Сюда входит стойкость к SQL-injection, XSS-скриптингу, защита от подмены передаваемых параметров. Обязательна возможность осуществления резервного копирования и восстановления данных.

Защита от флуда - графическая или математическая каптча на выбор, анализатор поведения клиентов с отсеиванием ботов. Возможно ограничение на количество ссылок в одном сообщении, контроль флуда, бан ip и вести «черный список» рекламируемых сайтов, а также фильтровать сообщения по подстроке.

Борьба со спамом в интернете на данный момент ведется только в одном месте — комментарии к публикациям. Компонентов комментариев достаточно много, и о таковых, без поддержки captcha я не слышал (разве что очень древние и не используемые на данный момент). На этом собственно борьба со спамом заканчивается. Наибольшую же эффективность в борьбе со спамом можно добиться с помощью т.н. капчи — картинки с набором символов. В этом случае ставится полный заслон от автоматических спам-роботов.

13. Usability

Давайте подумаем над тем, насколько способен пользователь в той или иной CMS выполнять простые операции без тренировки, сможет ли он это сделать без обращения к поставщику решения. По определению CMS предназначена для управления содержанием. Таким образом, наиболее типовая операция для пользователя CMS – создание/изменение содержания документов сайта. Эта операция включает в себя нахождение требуемого документа и затем интерфейса для управления его содержанием. Если предположить, что документы на сайте представлены в той же структуре, что и административном интерфейсе, наиболее очевидный путь – пройти по иерархии структуры, выбрать требуемый документ и использовать ссылку перехода к свойствам/содержанию документа. В этом плане наиболее привычными и понятными является пользовательские интерфейсы, напоминающие Проводник Microsoft Windows.

Далее мы должны ответить на вопрос «не перегружен ли интерфейс, не слишком ли он технический?». В принципе, этот вопрос наиболее актуален для пользовательской роли «оператор контента», исполнители которой не обязаны обладать исчерпывающими техническими знаниями.

Привычность использования CMS зависит прежде всего от того, насколько ее интерфейс выглядит похожим на популярные приложения. Разработчики могут задаться целью максимально повторить интерфейсные решения, принятые в Microsoft Windows. Интерфейсные элементы, структурное дерево и список могут иметь вид аналогичных элементов интерфейса Windows и поддерживают контекстное меню по правой кнопке мыши и Drag & Drop. Для всех документов и записей в системе доступны организованные виде закладок интерфейсы управления свойствами.

CMS должна использовать стандартные GUI-элементы (меню, кнопки, элементы форм.

Логичность интерфейса

CMS должна давать возможность редактирования содержания документов в режиме WYSIWYG.

Скорость реакции интерфейса

Скорость реакции интерфейса в значительной степени зависит от производительности несущего сервера, скорости соединения и прочих внешних факторов. Однако немалое значение имеет и архитектура пользовательского интерфейса. CMS должна максимально сокращать число необходимых действий для завершения типовых операций. В CMS, использующих технологию AJAX, базовые интерфейсы не требуют перегрузки всей страницы.

Терпимость к провалам

Рассуждая о системах управления содержанием, следует учитывать, что зачастую от пользователей этого класса программного обеспечения не требуется специальных технической квалификации. Хотя, любой пользователь, являясь живым человеком, не застрахован от совершения ошибок в своей работе. Хорошая CMS либо не должна допускать подобных ошибок, либо обязана сообщать о них. Обязательно CMS иметь специальные средства отмены последних операций.

Масштабируемость.

Когда объем структуры сайта слишком велик, в CMS должна появляться постраничная навигация.

Пользовательский тюнинг.

Очень важный фактор – возможность обустроить систему согласно своим собственным предпочтениям. Полезны были бы возможности сворачивать/разворачивать

формы интерфейсов, сортировать списки, изменять их внешний вид, сортировки и фильтрации. Все эти изменения должны автоматически сохраняться для текущего пользователя.

Лекция 19

Состав панели управления типичной системы веб-разработки и управления контентом (CMS).

Основная идея систем управления контентом – разделение визуального дизайна сайта и его *информационного наполнения*. При создании сайта с помощью такой системы разрабатывается набор шаблонов страниц, в которых впоследствии размещается*информация*. В этом случае *роль разработчиков* (фактически это *группа* внедрения) ограничивается только созданием "начальной" информационной системы на основе системы управления контентом, затем пользователи сами публикуют требуемую информацию и определяют ее *представление*. Управление сайтом сводится к минимуму, – администратору остается только управлять пользователями.

Пользователи *CMS* делятся на две группы – создатели шаблонов страниц и авторы контента (*информационного наполнения*). Таким образом, одна *группа пользователей* создает структуру и оформление страниц, а другая наполняет его содержанием.

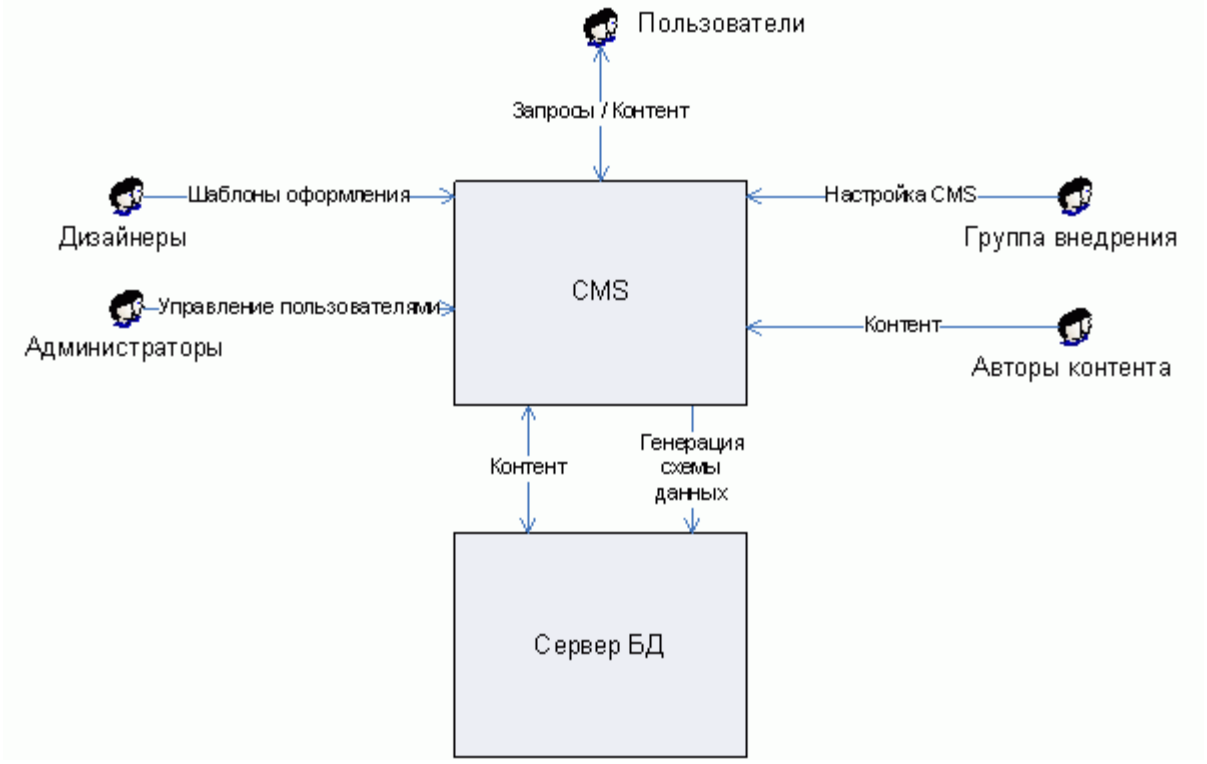


Рис. 7.1. Схема функционирования CMS

Функции систем управления контентом структурированы согласно жизненному циклу системы.

Сначала *группа* внедрения разворачивает *ядро CMS* и создает в *СУБД* информационное хранилище контента – *БД*. Далее*администратор* предоставляет *доступ* к системе различным пользователям, затем создается *контент*, он публикуется, и к нему применяются шаблоны оформления.

Создание контента

На первом этапе необходимо создать все типы контента и схемы их метаописаний, а также настроить систему на определенный*поток работ* (если система поддерживает создание *потоков работ*, а не использует единственный встроенный). Понятие типа контента аналогично понятию класса, а элементы контента представляют собой набор экземпляров таких "классов". Типами контента являются, например, текст и изображение; экземпляром контента конкретный документ или картинка.

Следующая важная возможность – хранение информации о версии контента. Это позволяет задать номер версии любых операций изменения контента и при необходимости восстановить его. В любой момент можно отказаться от изменений и, практически в режиме реального времени, откатиться на одну из предыдущих зафиксированных версий. Строгий *контроль версий* необходим для определения ответственности отдельных лиц, а также для резервного и аварийного восстановления системы.

Кроме управления контентом, система должна предоставлять возможность создавать *метаданные* о нем. *Метаданные* – это сведения о данных, свойства данных. Примером *метаданных* служат ключевые (характерные) слова документов, предназначенные для поисковых или отчетных систем. Системы управления контентом, рассматриваемые в данном обзоре, не поддерживают*метаданные*, хотя можно специально ввести дополнительные типы контента, представляющие собой *метаданные*.

После того, как все типы контента созданы, авторы *информационного наполнения* начинают создавать, изменять и удалять элементы контента указанного типа. *CMS* уже содержит некоторый набор визуальных компонентов, например, для редактирования текста, выбора изображений, выбора шаблона представления.

Кроме непосредственно редактирования элементов контента, необходимо предусмотреть разбиение контента по категориям или рубрикам.

Создание шаблонов оформления

В качестве решения проблемы представления в системах управления контентом используется технология шаблонов, определяющих внешний вид страницы. Разработчику шаблонов не нужно знать никаких технических тонкостей. На ранних этапах существования WWW шаблоны представляли "заготовки" HTML-кода, из которого путем манипуляций в *HTML-редакторе*получались готовые страницы. Сегодня такими заготовками манипулируют уже не дизайнеры в своих редакторах, а серверные web-приложения. Таким образом, современный шаблон Web-страницы представляет собой блок HTML, который благодаря специальным тегам или внедренным сценариям, облегчает включение динамически сгенерированного содержания на этапе выполнения. При использовании подобных шаблонов программистам необходим некоторый стандартизированный интерфейс для работы с ними – шаблонный движок (в английском языке существует устоявшийся термин – *template engine*), который может иметь разнообразные дополнительные функции, например, поддерживать кэширование шаблонов, их динамическое обновление и т.д.

Публикация контента

Механизм публикации информации в системе управления контентом отвечает за процесс создания, редактирования и удаления шаблонов страниц, а также за сопоставление типов контента и шаблонов страниц. В состав дополнительных возможностей системы публикации может входить предварительная генерация статической версии сайта. Эта опция очень полезна в случае размещения информационной системы на оборудовании с ограниченными возможностями.

Типичный процесс публикации информации в World Wide Web реализован в Microsoft *Content Management* Server. Обычным приемом обеспечения оформления *информационного наполнения* являются шаблоны представления информации. Поэтому первым этапом процесса является создание наборов шаблонов. Типичный шаблон содержит разметку HTML и места, куда в дальнейшем будут вставлены данные (*placeholder*'ы в терминологии Microsoft). Далее на основе этих шаблонов авторы*информационного наполнения* создают страницы и представляют их редакторам для одобрения. Редакторы, в свою очередь, могут либо отклонить страницу и вернуть ее автору на доработку, либо одобрить ее и передать модератору сайта. В первом случае процесс повторяется снова, во втором же модератор сайта проверяет расположение страницы на сайте, дату и срок ее публикации. Если все в порядке, страница становится видна пользователям. Несмотря на то, что *рабочий процесс* в Microsoft*Content Management* Server фиксирован и не может быть изменен в дальнейшем, подобное решение подходит большинству пользователей, которым необходимо публиковать информацию в World Wide Web.

Управление пользователями

Управление пользователями включает создание, изменение и удаление учетных записей отдельных пользователей и их групп, а также назначение прав для работы с элементами контента. Важной частью требований является наличие пользовательских профилей (*profiles*), с помощью которых можно сгенерировать персональное представление информации для каждого пользователя. Полезной является и возможность пользователя делегировать свои права. Это позволяет пользователям переназначать исполнителя конкретной работы и избегать простоев из-за отсутствия отдельного лица.

Системы управления контентом управляют учетными записями пользователей на основе собственных групп, не используя существующие идентификационные системы, например, Windows. Аутентификация средствами Windows позволила бы значительно упростить администрирование. При этом система управления контентом могла бы использовать операционную систему локального компьютера или контролера домена для проверки и сопровождения учетной записи пользователя.

Представление информации создается на основе данных, а также предпочтений конкретного пользователя. Персонификация достигается путем использования профилей – специальных записей, в которых хранится информация, специфичная для конкретных пользователей.

Visual Web Developer Express позволяет создавать как веб-страницы ASP.NET, так и страницы HTML. Веб-страницы ASP.NET являются динамическими.Она включают серверные веб-элементы управления ASP.NET и код, обрабатываемый ASP.NET на сервере. Во время обработки на сервере элементы управления и код выводят результат, передаваемый в обозреватель в виде HTML (или другой разметки). Дополнительные сведения см. в разделеОбщие сведения о веб-страницах ASP.NET.

Visual Web Developer Express содержит конструктор веб-страниц. В представлении **Конструирование** он предоставляет поверхность WYSIWYG разработки для ввода теста и добавления элементов управления. Если требуется непосредственное изменение разметки страницы, перейдите в представление **Исходный код**. В представлении **Источник** редактор предоставляет средства для создания разметки с правильным форматом, например автозавершение операторов и IntelliSense. Кроме того, редактор проверяет соответствие разметки выбранной схеме проверки, например XHTML 1.0. Представление **Разделение** позволяет просматривать одновременно представление конструирования и разметку. Дополнительные сведения см. в разделах Представление исходного кода конструктора веб-страниц и Представление конструктора. Конструктор веб-страниц.

Пользовательский макет и представление

Используя главные страницы, выступающие в роли шаблонов, можно создавать пользовательские макеты страниц. На главной странице создается общий макет, затем создаются страницы содержимого, которые объединяются с главной страницей. Для создания пользовательского внешнего вида для страниц веб-узла используются темы. С помощью тем определяется цвет, шрифт и другие характеристики элементов управления и страницы.

Visual Web Developer Express также предоставляет средства, упрощающие работу с каскадными таблицами стилей (CSS). В представлении**Конструирование** можно разрабатывать макет и содержимое стиля, используя средства пользовательского интерфейса, например окно **Свойства CSS**. В представлении **Конструирование** можно также изменять положение, заполнение и внешние поля с помощью WYSIWYG-средств визуального конструирования. Дополнительные сведения см. в разделах Главные страницы ASP.NET, Темы и обложки ASP.NET и Общие сведения о работе с CSS.

Веб-элементы управления

Для упрощения развертывания веб-страниц используются серверные веб-элементы управления ASP.NET. Серверные веб-элементы управления поддерживают знакомые типы функциональности для страниц, например отображение текстовых полей, кнопок, флажков, меню и т. д.

Серверные веб-элементы управления ASP.NET отличаются от элементов HTML тем, что их можно программировать в серверном коде. Благодаря этому можно создавать веб-страницы, представляющие пользователям сложный интерфейс, и одновременно динамически формировать содержимое веб-страницы на основе доступных с сервера данных.

ASP.NET включает серверные веб-элементы управления, предназначенные для выполнения разнообразных задач на веб-страницах. Эти элементы перечислены ниже.

- **Стандартные элементы управления** Позволяют добавлять в веб-страницы ASP.NET как основные, так и сложные функциональные возможности. К стандартным элементам управления относятся кнопки, изображения, флажки, гиперссылки, поля со списками и т. д. К другим стандартным элементам управления относится календарь, а также элементы управления для отправки файлов и отображения XML на веб-странице.
- **Элементы управления данными** Позволяют подключать веб-страницу к различным источникам данных, включающих базы данных и файлы XML. С помощью этих элементов управления данные на странице отображаются в таблицах или других форматах и доступны для изменений.
- **Элементы управления переходов** Позволяют добавлять различные типы меню на веб-страницы. В ним относятся статические и контекстные меню, представление в виде дерева и строка переходов (известная также как иерархическая навигация).
- **Элементы управления проверки** Обеспечивают проверку вводимых пользователем данных. Можно проверять обязательные для заполнения поля, диапазоны значений, минимальные и максимальные значения и определенные шаблоны.
- **Элементы управления входа в систему** Позволяют без труда создавать формы для входа в систему и выполнять проверку подлинности пользователей. Эти элементы управления можно использовать для регистрации пользователей на веб-узле и восстановления или замены их паролей.
- **Элементы управления веб-частей** Позволяют пользователям настраивать веб-страницы ASP.NET в обозревателе. Элементы управления веб-частей могут использоваться для настройки содержимого, например заголовков новостей или данных прогноза погоды. Пользователи могут выбрать элемент управления веб-частей для отображения, а также настроить макет и внешний вид этих элементов. Элементы управления веб-частей персонализированы, таким образом личные установки пользователей сохраняются между сеансами обозревателя.
- **Элементы управления расширением Ajax**: позволяют усовершенствовать веб-узел с помощью возможностей Ajax, включая асинхронную обратную передачу.

Лекция 20
Общее администрирование веб-ресурса, подготовка и наполнение контентом.

Visual Web Developer Express поддерживает элементы управления источников данных ASP.NET, выполняющие все задачи, необходимые для подключения к различным типам источников данных и для взаимодействия с ними. Например, элемент управления SqlDataSource содержит объект подключения к данным, который может подключиться к базе данных SQL Server. Кроме того, в него входят объекты команд данных для операторов SQL **Select**, **Update** и **Delete**. С помощью других элементов управления источников данных можно подключаться к источникам данных OLE-DB, источникам данных XML и другим. Преимуществом этих элементов управления является то, что они предоставляют единообразный интерфейс для привязки данных для всех элементов управления ASP.NET.

Visual Web Developer Express поддерживает LINQ. LINQ — это синтаксис запросов, с помощью которого операции запросов определяются непосредственно в C# и Visual Basic. LINQ позволяет запрашивать базы данных или находящиеся в памяти источники данных. Для поддержки запросов баз данных Visual Web Developer Express

предоставляет конструктора объектно-реляционного сопоставления, который позволяет быстро создавать и изменять классы, сопоставленные с объектами в базе данных.

Visual Web Developer Express также поддерживает платформу динамических данных ASP.NET. Это позволяет создавать расширяемые веб-приложения, управляемые данными, путем определения внешнего вида и поведения сущностей данных из схемы базы данных во время выполнения, а также наследования принципов работы пользовательского интерфейса.

Visual Web Developer Express также поддерживает различные элементы управления данными, которые можно добавить на веб-страницы ASP.NET для отображения данных. К ним относятся следующие элементы управления: GridView, DetailsView, FormView, ListView, DataList и Repeater. Каждый из них предназначен для представления данных разными способами. Каждый элемент управления поддерживает дополнительные функции, такие как сортировка, разбиение по страницам, изменение записей данных, вставка новых записей и т. д. Такие элементы управления, как ListBox и DropDownListможно заполнять данными из элементов управления источников данных.

Visual Web Developer Express позволяет перетаскивать таблицы данных на страницу. В этом случае Visual Web Developer Express автоматически создаст предварительно настроенные элементы управления.

Благодаря SQL Server Express данные можно сохранять в локальной базе данных. Этот продукт загружается дополнительно. Дополнительные сведения см. в разделе Использование SQL Server, экспресс-выпуск с ASP.NET.

Безопасность

Используя такие функции ASP.NET, как членство, роли и серверные элементы управления входом в систему, к веб-сайту можно добавить функции проверки подлинности (входа в систему) и авторизации с минимальным объемом кода или без него. Пользователям можно предоставить возможность регистрации на веб-узле, кроме того, можно создать страницу входа в систему, которая автоматически проверяет учетные данные пользователя. Можно защитить страницы, чтобы их могли просматривать только вошедшие в систему пользователи. На одной странице можно отображать одни данные для пользователей, которые выполнили вход, и другие для анонимных пользователей.

Дополнительные сведения см. в разделе Система безопасности ASP.NET.

Веб-узлы и проекты веб-приложений

Visual Web Developer Express содержит несколько типов проектов, помогающих создавать веб-приложения, сайты и службы:

Примечание
Visual Web Developer Express не поддерживает все типы проектов.

- **Проекты веб-сайтов ASP.NET** Проекты веб-сайтов ASP.NET включают в себя определенные типы файлов, которые ASP.NET распознает и обрабатывает. Кроме того, можно создавать папки для особых целей, например для хранения исходного кода, определения тем и управления ресурсами. Visual Web Developer Express включает в себя шаблон пустого проекта веб-сайта ASP.NET и шаблон проекта веб-сайта ASP.NET, содержащий множество автоматически создаваемых возможностей, поддерживающих настройку. Дополнительные сведения см. в разделеПроекты веб-сайтов ASP.NET.
- **Проекты веб-приложений ASP.NET** Проекты веб-приложений ASP.NET позволяют скомпилировать веб-сайт в единую сборку и явным образом определить ресурсы проекта. Этот тип предоставляет альтернативную модель проектов веб-сайтов, предлагая дополнительные варианты развертывания и поддержания веб-приложений. Visual Web Developer Express включает в себя шаблон пустого проекта веб-приложения ASP.NET и шаблон проекта веб-приложения ASP.NET, содержащий множество автоматически создаваемых возможностей, поддерживающих настройку. Дополнительные сведения см. в разделе Проекты веб-приложений ASP.NET.

- **Веб-проекты MVC 2 в ASP.NET** Шаблон проекта веб-приложения MVC 2 в ASP.NET и шаблон проекта веб-сайта MVC 2 в ASP.NET могут использоваться для создания веб-приложений, в которых применяется архитектурный шаблон "модель-представление-контроллер".Дополнительные сведения см. в разделе Общие сведения о ASP.NET MVC.
- **Веб-проекты сущностей на платформе динамических данных ASP.NET** Шаблон проекта веб-приложения на платформе динамических данных и шаблон проекта веб-сайта на этой же платформе используются для работы с платформой ADO.NET Entity Framework. Типы проектов предназначены для любой реляционной базы данных. Дополнительные сведения о динамических данных см. в разделе Карта содержимого разделов, посвященных динамическим данным ASP.NET.
- **Веб-проекты Linq to SQL на платформе динамических данных ASP.NET** Шаблон проекта веб-приложения Linq to SQL на платформе динамических данных и шаблон проекта веб-сайта Linq to SQL на этой же платформе используются для создания управляемых данными веб-приложений, в которых LINQ применяется для взаимодействия с базой данных SQL. Такое веб-приложение предназначено для любой реляционной базы данных. Дополнительные сведения о динамических данных см. в разделе Карта содержимого разделов, посвященных динамическим данным ASP.NET.
- **Проекты служб WCF** Шаблон приложения служб WCF предоставляет базовую структуру классов для разработки служб. Шаблон содержит основные определения для контракта службы, контракта данных, реализации службы и ее конфигурации. Этот шаблон можно использовать для создания службы, которая потребует минимального объема дополнительного кодирования, и которая может служить как стандартный блок для более сложных служб.
- **Проекты библиотек классов**: шаблон библиотеки классов предназначен для быстрого создания многократно используемых классов и компонентов, которые могут использоваться сразу несколькими проектами. Дополнительные сведения см. в разделе Class Library Template.
- **Проекты веб-сайтов Silverlight 1.0** Шаблон проекта веб-сайта Silverlight 1.0 создает базовый веб-сайт Silverlight 1.0 с помощью технологий JavaScript и XAML.

Возможности проектов веб-сайтов и веб-приложений

Visual Web Developer Express предоставляет множество возможностей, позволяющих создавать проекты веб-сайтов и веб-приложений и управлять ими. К числу дополнительных возможностей относятся следующие.

- **Настройка для различных версий** Веб-приложение можно настроить на работу с определенной версией .NET Framework. По умолчанию Visual Web Developer Express предназначен для .NET Framework 4. Функция настройки для различных версий гарантирует, что приложения используют только функциональные возможности, доступные в указанной версии .NET Framework. Кроме того, настройка для различных версий позволяет развертывать старые приложения без необходимости добавления новой версии .NET Framework в пакет развертывания.
- **IntelliSense** IntelliSense предлагает соответствующие контексту варианты кода при вводе, такие как свойства, функции и объекты. IntelliSense позволяет сохранять контекст кодирования, находить необходимые сведения и вставлять элементы языка прямо в код.
- **Отладка** Для тестирования страниц используется интегрированный отладчик, позволяющий находить ошибки в коде. Также можно включить трассировку, отображающую полезные отладочные сведения на каждой странице. Дополнительные сведения см. в разделе Общие сведение об отладке в ASP.NET.
- **Индивидуализация веб-страниц** Создание профилей пользователей позволяет сохранять указанные пользователем параметры, которые можно использовать для настройки страниц для каждого пользователя. Дополнительные сведения см. в разделе Общие сведения о свойствах профилей ASP.NET.
- **Управление состоянием** Возможности управления состоянием в Visual Web Developer Express позволяют хранить данные между запросами страниц, например

сведения о клиентах или содержимое корзины покупок. Можно сохранять и управлять сведениями, связанными с приложением, сеансом, страницей и пользователем, а также сведениями, определенными разработчиком.

- **Глобализация** Страницы можно настроить на автоматическое чтение текста из файла ресурсов, соответствующего выбранному пользователем языку и языковому стандарту. Дополнительные сведения см. в разделе Глобализация и локализация ASP.NET.
- **Развертывание** Различные средства позволяют публиковать веб-сайты на тестовых или рабочих серверах. Дополнительные сведения см. в разделе Карта содержимого развертывания ASP.NET.

Лекция 21
Размещение сайта на хостинге

Хо́стинг (англ. *hosting*) — услуга по предоставлению вычислительных мощностей для размещения информации на сервере, постоянно находящемся в сети (обычно Интернет). Хостингом также называется услуга по размещению оборудования клиента на территории провайдера с обеспечением подключения его к каналам связи с высокой пропускной способностью (колокация, от англ. *colocation*).

Обычно хостинг входит в пакет по обслуживанию сайта и подразумевает как минимум услугу размещения файлов сайта на сервере, на котором запущено ПО, необходимое для обработки запросов к этим файлам (веб-сервер). Как правило, в обслуживание уже входит предоставление места для почтовой корреспонденции, баз данных, DNS, файлового хранилища на специально выделенном файл-сервере и т. п., а также поддержка функционирования соответствующих сервисов.

Хостинг базы данных, размещение файлов, хостинг электронной почты, услуги DNS могут предоставляться отдельно как самостоятельные услуги, либо входить в комплексную услугу.

Выбор хостинга Одним из критериев выбора хостинга является используемая операционная система, поскольку от этого зависит программное обеспечение, которое будет поддерживать функциональность тех или иных сервисов. Важным аспектом описания хостинга является наличие тех или иных служб и возможностей:

- поддержка CGI: Perl, PHP, Python, ASP, Ruby, JSP, Java
- поддержка .htaccess/.htpasswd (для Apache)
- поддержка баз данных

А также установленные модули и фреймворки для каждой из возможностей.

Хостинг как услугу сравнивают, описывают и оценивают по количественным ограничениям:

- размер дискового пространства под файлы пользователя
- количество месячного трафика
- количество сайтов, которые можно разместить в рамках одной учетной записи
- количество FTP пользователей
- количество E-Mail ящиков и объём дискового пространства, предназначенного для почты
- количество баз данных и размер дискового пространства под базы данных
- количество одновременных процессов на пользователя
- количество ОЗУ, и максимальное время исполнения, выделяемое каждому процессу пользователя

качественным ограничениям:

- свободные ресурсы CPU, оперативной памяти, которые влияют на быстродействие сервера
- пропускная способность каналов, которая влияет на загрузку информации.

- удаленность оборудования хостера от целевой аудитории сайта, которая влияет на загрузку информации.

Некоторые платные хостинговые компании предоставляют бесплатный тест на определённый период, по истечении которого пользователь должен определиться подходит ли для него выбранная хостинговая компания, и имеет ли смысл оплачивать большие периоды. Как правило такие тесты предоставляются только владельцам доменов второго уровня во избежание спекуляций с тестовыми аккаунтами.

Помимо платных хостеров существуют также и бесплатные хостинг-компании, поддерживающие большинство описанных веб-технологий. Однако в России они не распространены, поэтому люди пользуются в основном услугами платного хостинга.

Хостинг в В России, с юридической точки зрения, услуга хостинга не относится к телематическим услугам связи в силу различия определения телематических услуг связи (предоставление доступа пользовательского оборудования к сети связи оператора[1]) и сути хостинга (предоставление ресурсов оборудования подключенного к сети связи для размещения и функционирования веб-сайта клиента).

В случае, когда оператор связи размещает у себя оборудование пользователя и обеспечивает его подключение к Интернету (колокация) — в части оказания услуги по обеспечению доступа к сети связи деятельность оператора подлежит лицензированию. В России лицензии выдаются Федеральной службой по надзору в сфере связи.

Колокация, колокейшн (от фр. *colocation* — совместное проживание, англ. *collocation* — расположение рядом, в свою очередь от лат. *collocatio(n-)* — помещать вместе[2]) — услуга, состоящая в том, что провайдерразмещает оборудование клиента на своей территории (обычно в датацентре), подключает его к электричеству, обеспечивает обслуживание и подключение к каналам связи с высокой пропускной способностью. Иногда указанное оборудование не принадлежит клиенту, а арендуется им у того же провайдера, в этом случае услуга называется «аренда выделенного сервера».

Такое размещение позволяет сэкономить на организации канала связи от провайдера до клиента — последней мили. Чаще всего на колокацию ставят серверы, предназначенные для поддержания веб-сайтов и других сетевых служб с большим объёмом трафика, а также оборудование, к которому требуется надёжный доступ из многих точек, например, VPN-концентраторы, шлюзы IP-телефонии.

Обычно в состав данной услуги помимо собственно размещения оборудования и подключения к каналу связи также входит:

- организация удаленного доступа к оборудованию (KVM, удаленный доступ);
- резервированное электропитание;
- обеспечение климатического режима;
- физическая охрана;
- простейшие услуги по обслуживанию оборудования (перезагрузить, вставить диск и т. п.);
- мониторинг.

Другие сопутствующие услуги (например, резервное копирование данных, защита информации) обычно оказываются оператором за отдельную плату.

Некоторые склонны классифицировать колокацию как подвид хостинга, другие же полагают её самостоятельной услугой.

С юридической точки зрения, на настоящий момент (июнь 2012 года) нет однозначного мнения, относится ли услуга колокации к услугам связи или нет. Ряд юристов и государственных органов относит колокацию к услугам передачи данных. Деятельность по оказанию услуг по передаче данных в России подлежит лицензированию, поэтому в настоящее время большинство провайдеров, оказывающих услуги колокации получают лицензии на услуги по передаче данных. Такие лицензии выдаёт Федеральная служба по надзору в сфере связи, информационных технологий и массовых коммуникаций

Хостинговая компания (также: *хостер, хостинг-провайдер, веб-хостер, HSP (Hosting Service Provider)*) — компания, занимающаяся предоставлением услуг размещения оборудования, данных и web-сайтов на своих технических площадках (хостинг). Зачастую, оказанием услуг хостинга занимаются также компании, для которых данный вид деятельности основным не является — интернет-провайдеры, регистраторы доменов.

Сфера деятельности

Может предоставлять услуги:

- Размещение оборудования (colocation)
- Виртуальный выделенный сервер (VDS)
- Выделенный сервер (Dedicated)
- Виртуальный хостинг (Shared hosting)
- Игровой сервер
- Сервер электронной почты
- Хранилище данных
- Регистрация доменных имен
- Software as a Service (SaaS)
- Cloud-hosting

Однако самыми распространенными являются услуги виртуального хостинга, регистрации доменов и VDS.

Технические аспекты

Задача хостинговой компании — предоставление доступа к оборудованию или данным клиента из сети Интернет. Основными характеристиками, определяющими техническое качество услуги хостинга являются:

- Постоянная доступность узлов сети (т. н. uptime).
- Связанность в глобальной сети (доступность из других сегментов Интернета).
- Обеспечение физической и информационной безопасности оборудования и данных.
- Предоставление достаточных для функционирования сайта ресурсов сервера (в случае виртуального хостинга и VDS).
- Обеспечение требуемых условий безопасной эксплуатации (электропитание, температура воздуха и т. д.) в случае dedicated и colocation.

Большая часть Российских хостинг-провайдеров не имеет своей технологической площадки (дата-центра), а арендует оборудование или место для его размещения (colocation) у других операторов.

Как правило, для упрощения настройки площадки для клиентов виртуального хостинга и аренды VDS используются специальные панели управления.

Правовое положение в России

Согласно законодательству Российской Федерации, хостинг является лицензируемым видом деятельности юридического лица. Для легального предоставления услуг хостинга требуется:

- Лицензия на оказание телематических услуг связи (ТУС). [1]
- Разрешение на эксплуатацию узла электросвязи (т. н. «сданный узел связи»).<в настоящее время: введенный в эксплуатацию узел связи>
- Сертифицированная система учета услуг.
- Сертификаты ССС Минсвязи на оборудование, используемое для предоставление услуг связи.

В случае, если оператор планирует оказывать услуги размещения оборудования (colocation) или аренды выделенных серверов, ему также потребуется:

- Лицензия на услуги связи по передаче данных, за исключением услуг связи по передаче данных для целей передачи голосовой информации.

Реселлинг

Основная статья: ***Реселлинг хостинга***

Часть хостинг-провайдеров (как правило — начинающих) не имеет собственного или арендуемого оборудования, а использует услугу реселлинга для перепродажи услуг вышестоящего оператора на условиях агентского договора (либо без него). Обычно, такие компании называют **реселлерами** или**агентами**.

Компании-реселлеры не имеют непосредственного доступа к оборудованию, которое используется для предоставления услуг. Техническое обеспечение услуги берет на себя вышестоящая хостинговая компания. Компания-реселлер занимается исключительно привлечением и поддержкой клиентов.

Как правило, реселлеры имеют возможность установки собственной ценовой политики, отличной от политики вышестоящего провайдера.

Лекция 22
Основы программирования информационного контента на языках высокого уровня.

Любая программа – это сложный набор команд, которые задаются с использованием специальных «переводчиков» или компиляторов. Их цель – преобразовать наши команды в понятный для компьютера язык. Принцип их работы заключается в том, что они задают некое число заранее определённых электрических разрядов, серия которых несёт за собой последствие в виде решения поставленной задачи. В обиходе эти компиляторы называют языками программирования, то есть языками, понятными и для программистов, и для компьютеров.

Развиваться именно как компьютерные программы эти языки начали в середине 60-х – начале 70х, когда были созданы первые настоящие компьютеры, способные исполнять несколько различных функций. Со временем количество компиляторов увеличивалось по мере того, как расширялись направления работы компьютеров. Например, некоторые из них использовались исключительно для создания операционных систем, другие – только для написания программного обеспечения разнообразных направлений. Процесс написания элементарного калькулятора – уже достаточно сложная процедура, требующая определённых знаний и навыков. Мы не будем рассматривать конкретные примеры создания программ. Вместо этого сравним самые популярные из ныне существующих языков программирования, которые используют профессионалы. Их десять:

1.C++ (1983)
2.Python (1990)
3.Visual Basic (1991)
4.PHP (1994)
5.Delphi (1995)
6.Java (1995)
7.JavaScript (1995)
8.Ruby (1995)
9.ActionScript (1998)
10. Nemerle (2006)

Кратко рассмотрим их плюсы и минусы.

Обзор самых популярных программирования

1) C++

Это один из первых, и невероятно популярный до сих пор язык, которым просто обязан владеть каждый программист. Не обязательно начинать изучение программирования именно с него, но со временем C++ всё равно станет одним из наиболее используемых компиляторов. С момента своего создания этот язык прошёл несколько стандартизаций и обновлений, благодаря чему остаётся актуальным и в наше время. Основным его плюсом является полная универсальность – на C++ можно писать всё. Именно благодаря этому его так часто используют профессионалы. Также можно отметить сравнительную простоту компилятора – если вы уже овладели C, Python или Java, то работать с C++ для вас не составит труда. Однако критиков у языка также достаточно. Основные их аргументы – это неудобный синтаксис и то, что иногда

результатом работы с C++ является слишком длинный программный код, что влечёт за собой некоторые неудобства в дальнейшей работе с программой. Правда, вторая проблема была частично решена в 1998 году, когда были введены дополнительные шаблоны компилятора. В целом, что бы ни говорили критики, а C++ вот уже почти 30 лет остаётся одним из наиболее популярных языков программирования в мире.

2) Python

Питон – достаточно популярный в наше время язык, суть которого сводится к тому, чтобы как можно проще было создавать сложные программы. Созданный на основе более ранних языков, он впитал в себя все их наработки и является более совершенным. К тому же он постоянно обновляется, последняя версия была выпущена в начале 2012 года. Основные плюсы компилятора – минимализм, многофункциональность и простота. Но в свою очередь, за минимализм приходится платить низким быстродействием, а за минимализм – наличием множества ошибок в системном коде, некоторые из которых присутствуют и в самых последних версиях. Как бы то ни было, но и Python нашёл своих поклонников.

3) Visual Basic

Этот продукт от Майкрософт знают все программисты, ведь большинство именно на нём познавало азы работы с компиляторами. Он прост, многофункционален и идеально подходит для быстрого создания прототипов программ. Кроме широких возможностей по созданию программного кода, также с его помощью можно конструировать пользовательский интерфейс программы. Именно ВБ чаще всего используют профессионалы из Майкрософт для создания своих программ. Недостатки у компилятор также есть. К ним относятся отсутствие указателей, низкоуровневый доступ к памяти компьютера. Также программа отличается невысокой скоростью работы и возможностью использования только на Windows и Mac OS X. Более подробно ознакомиться с Visual Basic можно на сайте http://www.vbrussian.com/

4) PHP

Это язык программирования, который чаще всего применяют специалисты для создания веб-страниц. В настоящее время его принимают и поддерживают большинство хостинг-провайдеров. Благодаря возможности применения на любой ОС, скорости выполнения, функциональности и простоте его используют практически все разработчики сайтов. Также с помощью PHP можно создавать графический интерфейс пользователя. Создатели языка заложили в него несколько пасхальных яиц. Например, если ввести в строку

любой_сценарий.php?=PHPE9568F36-D428-11d2-A769-00AA001ACF42

то на экране появится фотография (то, что на ней изображено, зависит от версии).

Из недостатков языка важно вспомнить то, что если вы написали код на более ранней версии, то на новой он, скорее всего, не откроется. Также в нём просто не предусмотрена возможность создания многопоточных программ. Узнать больше информации о языке можно на сайте www.php.net

5) Delphi

Delphi – многофункциональный язык, созданный на основе Object Pascal. Данный язык является ещё одним из самых популярных языков из-за наличия огромного количества компиляторов и диалектов. Каждый программист использует именно тот диалект, который подходит для его основного направления в работе. В целом, Delphi – императивный, объектно-ориентированный язык. Чаще всего на этом языке создаются разнообразные программы. Так, именно с помощью Delphi создали Light Alloy, AIMP, The KMPlayer, Total Commander, QIP, графический интерфейс Skype и многое другое. Именно наличие множества диалектов является одновременно и достоинством, и недостатком языка, так как программу, написанную на одном из них, другой компилятор, скорее всего, не откроет. Сайт delphi.about.com позволит более подробно ознакомиться с этим языком.

6) Java

Это ещё один известный язык, способный работать на любой платформе. Более того, сейчас большинство операционных систем просто обязаны включать его в свой состав, так как работа некоторых приложений без этого компилятора будет недостаточно результативной. Однако есть у языка и подводные камни. Так, программы, написанные на Java, имеют репутацию слишком медлительных и требующих больших объёмов оперативной памяти. Java имеет несколько модификаций (не диалектов, а именно разных видов), которые созданы для различных целей. Каждый из них имеет свои уникальные библиотеки данных и структуру, позволяющую более продуктивно работать над определённым направлением программирования. Больше информации на сайте http://oracle.com/technetwork/java/

7) JavaScript

JavaScript – необходимый атрибут большинства современных браузеров. Чаще всего цель этого языка – придать веб-страницам больше интерактивности. На его создание повлияли множество более ранних языков программирования, на основе которых он соединил многофункциональность и простоту. По крайней мере для опытных программистов. Область применения JavaScript необычайно широка – это и веб-приложения, и браузеры, и прикладное ПО, и офисные и серверные приложения. Широкую популярность также обеспечивают ему специальные библиотеки, позволяющие достичь высокого уровня абстракции. Из минусов JavaScript стоит отметить очень низкий уровень безопасности его приложений, а также массу ошибок в песочнице, браузере, плагинах и расширениях, с которыми хотя бы однажды сталкивался каждый. Но в то же время, совсем отказаться от этого языка пока что нельзя, ведь без него откажут в работе большинство интернет-браузеров. Более подробно на www.javascript.ru

8) Ruby

Это кроссплатформенный и невероятно многофункциональный язык, полностью предназначенный для объектно-ориентированных программ. Синтаксис языка простой и лаконичный, без множества ненужных кодов. Также язык содержит в себе сборщик ненужного мусора, поддерживает блоки команд и замыкания с полной привязкой к переменным. У Руби довольно обширное количество разнообразных библиотек, каждая из которых применяется в определённых случаях, в зависимости от необходимости. Также особенностью этого скриптового языка является то, что программы, написанные на нём, могут в дальнейшем использовать его для расширения своих возможностей. На данный момент уже выпущено несколько версий языка, каждая из которых принесла что-то новое в принцип его работы. Кроме того, этот язык один из самых экономных по отношению к ресурсам компьютера, за что его так любят обычные программисты. Самая последняя из них – 1.9.3-p194, релиз которой состоялся 20 апреля 2012 года. На официальном сайте вы можете более подробно ознакомиться с достоинствами и недостатками этого языка www.ruby-lang.org

9) ActionScript

Этот язык также менее популярен среди обывателей из-за того, что с его помощью можно писать в основном объектно-ориентированные программы, даже чаще всего не программы, а Flash – приложения. Синтаксис и функциональность компилятора стандартны для подобного рода языков и заключаются в работе, запрограммированной лишь на указанный вид приложений. Плюсом этого языка является его многофункциональность по отношению к Flash. Как ни странно, но это же одновременно и его минус, так на нём больше нельзя написать практически ничего. Узнать больше о этом компиляторе можно на сайте actionscript.org

10) Nemerle

Это высокоуровневый гибридный язык, который сочетает в себе возможности как функционального, так и объектно-ориентированного программирования. Созданный всего несколько лет назад, он впитал в себя соответствующие наработки всех предыдущих поколений языков программирования, и, одновременно с этим, привнёс в создание программ кое-что новое – возможность метапрограммирования. Суть этого метода в создании программ, которые во время своей работы генерируют другие программы. То

есть, можно сказать, что он сам создаёт языки программирования? И да, и нет. Он создаёт возможные сценарии для дальнейшего генерирования исполняемых файлов. В целом, Nemerle просто-напросто объединил в себе все самые лучшие и необходимые функции более ранних языков. В общем, вышло так, что наиболее новый язык имеет меньше всего недостатков.

Лекция 23
Основы разработки серверного программного обеспечения

Разрабо́тка програ́ммного обеспе́чения (англ. software engineering, software development) — это род деятельности (профессия) и процесс, направленный на создание и поддержание работоспособности, качества и надежности программного обеспечения, используя технологии, методологию и практики из информатики, управления проектами, математики, инженерии и других областей знания.

Серверное программное обеспечение представляет собой интегрированный пакет программ, осуществляющий управление системой позиционирования в реальном времени (RTLS); сбор, обработку, хранение и автоматизированный анализ данных о местоположении маркированных объектов; формирование и выдачу сигналов об отклонении поведения объектов от сформулированных для них правил, а также предоставляющий графический интерфейс для взаимодействия пользователей с системой.

Для успешного применения технологии «клиент-сервер» должно использоваться соответствующее программное обеспечение, включающее клиентскую и серверную части. В частности, широко используемый пакет Microsoft Office представляет собой комплекс программ для клиентского компьютера. В его состав входят: текстовый процессор Word, табличный процессор Excel, система подготовки презентаций PowerPoint, система управления базами данных Access и программа управления информацией Outlook. В связи с успехом распространения этого пакета корпорация Microsoft решила собрать воедино комплекс программ для сервера - так появился пакет MS BackOffice.

В состав названного пакета входят следующие компоненты:

- Windows NT Server - сетевая операционная система;
- System Management Server - система администрирования сети;
- SQL Server - сервер управления базами данных;
- SNA Server - сервер для соединения с хост-компьютерами;
- Exchange Server - сервер системы электронной почты;
- Internet Information Server - сервер для работы с Internet.

Windows NT/2000 Server способна обеспечить совместное использование файлов, печатающих устройств, предоставить услуги по соединению с рабочими станциями (клиентскими компьютерами) и другой сервис.

Windows NT Server целесообразно использовать в случаях, когда предполагается наличие нескольких процессоров (обычно до четырех). Кроме того, Windows NT Server обеспечивает совместное использование ресурсов многими пользователями, возможность соединения с удаленными сетями через сервис удаленного доступа - RAS (Remote Access Service), а также через средства связи с сетями других фирм (Novell, Digital Pathworks и Apple).

System Management Server (SMS) позволяет сетевому администратору централизованно управлять всей сетью. При этом обеспечивается возможность администрирования каждого компьютера, подключенного к сети, включая установленное на нем программное обеспечение. SMS предоставляет следующий сервис:

1. управление инвентаризацией программного и аппаратного обеспечения;
2. автоматизация установки и распространения программного обеспечения, включая его обновление;
3. удаленное устранение неисправностей и предоставление полного контроля администратору за клавиатурой, мышью и экранами всех компьютеров в сети, работающих под управлением MS-DOS или Windows;
4. управление сетевыми приложениями.

SQL Server представляет собой систему управления реляционными базами данных, использующую принципы технологии «клиент-сервер». MS SQL Server поддерживает систему обработки транзакций, систему сохранения ссылочной целостности, механизм распределенных транзакций, тиражирование данных.

SNA Server обеспечивает возможность связи с IBM AS/400 и мэйнфреймами IBM (ЕС ЭВМ). Этот продукт позволяет нескольким настольным ПЭВМ, работающим под управлением MS-DOS, Windows, Windows NT, Macintosh, Unix или OS/2, «видеть» хост-компьютеры.

Exchange Server обеспечивает средства передачи и приема сообщений в информационной сети организации. Этот сервис включает электронную почту (E-mail) и обмен информационными сообщениями для рабочих групп. Microsoft Exchange Server построен на принципах технологии «клиент-сервер» и масштабируется в соответствии с возрастанием вычислительных возможностей сети.

Internet Information Server обеспечивает возможность создания Web-, FTP- и Gopher-серверов для сети Internet, поддерживает управление ими с помощью встроенной программы Internet Service Manager

Серверные программы делятся на следующие четыре вида.

1. Исполняемые программы, работающие через интерфейс CGI (Common Gateway Interface — общий интерфейс обмена), так называемые CGI-nпрограммы. Эта разновидность серверных программ — самая старая, однако отнюдь не устаревшая.
2. Расширения Web-сервера (приложения формата ISAPI, NSAPI, модули расширения Apache и т. п.). Новый способ, позволяющий встраивать серверные программы в сам Web-сервер, делая их его составными частями. Впервые предложен фирмой Microsoft для их сервера Microsoft Internet Information Server (интерфейс ISAPI) и разработчиками популярного бесплатного Web-сервера Apache.
3. Активные серверные страницы (ASP, JSP и др.). Фактически это обычные статические Web-страницы, сохраненные в файлах, Которые, кроме обычного HTML-кода, включают в себя команды, обрабатываемые либо самим Web-сервером, либо его расширением. Также новый способ, впервые предложенный Microsoft для того же Internet Information Server.
4. Серверные сценарии, написанные на интерпретируемом языке (Perl, Python, VBScript, JavaScript и др.). Обычные сценарии, работающие через интерфейс CGI или ISAPI на стороне сервера.
5. Теперь рассмотрим все это разнообразие подробнее.
6. CGI-программы представляют собой обычные исполняемые файлы, написанные на любом языке программирования и откомпилированные в машинный код процессора. Они не имеют интерфейса пользователя (как и все серверные программы), а работают с Web-сервером, получают от него входные данные и ему же пересылают результаты своей работы. Запускаются они самим Web-сервером, когда в них возникает нужда (когда необходимо обработать полученные от пользователя данные), и работают под управлением операционной системы серверного компьютера. При этом, если Web-серверу поступает одновременно несколько запросов на обработку данных от пользователей, он запускает соответствующее количество копий CGI-программы.
7. К достоинствам CGI-программ можно отнести легкость создания (многие среды разработки программ поддерживают создание таких приложений, в частности популярнейший Borland Delphi, начиная с версии 3) и простоту отладки. Также, поскольку CGI-приложения представляют собой независимые программы, они выполняются отдельно от Web-сервера (как говорят программисты и системные администраторы, выполняются в другом адресном пространстве). Это значит, что при сбое в CGI-программе завершается только она — сам Web-сервер остается "на плаву". А недостаток у CGI-программ всего один: большой расход системных ресурсов, поскольку для обработки каждого набора данных запускается отдельная копия серверной программы. И если Web-серверу поступит слишком много запросов на обработку данных, серверный компьютер может и зависнуть.

8. Расширения Web-сервера — более новая разновидность серверных программ. Они представляют собой обычные библиотеки DLL, в которых реализована вся логика серверной программы. Такие библиотеки как бы встраиваются в программу Web-сервера и работают как ее неотъемлемая часть. Поскольку библиотеки DLL работают только в среде Windows, для того чтобы создавать расширения в иных операционных системах, были придуманы и другие форматы. В частности, модули расширения сервера Apache не являются библиотеками DLL,

9. Именно в виде библиотек DLL создаются расширения Web-серверов Internet Information Server фирмы Microsoft и Netscape Web Server фирмы Netscape. В первом случае расширения имеют формат ISAPI(Internet Server Application Programming Interface — интерфейс программирования приложений интернет-сервера), а во втором — NSAPI (Netscape Server Application Programming Interface — интерфейс программирования приложений сервер^ Netscape). Формат модулей расширения Apache так и называется — модули Apache.

10. Достоинство у расширений Web-сервера одно: бережный расход системных ресурсов. Дело в том, что для обработки всех наборов данных пользователя запускается всего один экземпляр расширения, который отнимает существенно меньше ресурсов, чем уйма запущенных CGI-программ. Однако расширения труднее создавать и отлаживать, к тому же они не так безопасны.

11. Как CGI-программы. Поскольку они работают как часть Web-сервера, любая ошибка в расширении приведет к зависанию сервера.

12. Оба описанных выше вида серверных программ обладают одним огромным недостатком. Прежде чем они смогут работать, они должны быть написаны на языке программирования и откомпилированы в машинные коды процессора, что отнимает много времени, особенно при отладке. Конечно, откомпилированные программы работают быстрее интерпретируемых, т. е. тех, где каждая инструкция читается, расшифровывается и обрабатывается специальной программой-интерпретатором. Но у интерпретируемых программ есть и свои преимущества, главными из которых являются простота и быстрота написания. Две следующие разновидности серверных программ, которые будут описаны, как раз будут интерпретируемыми.

13. Как уже говорилось, активные серверные страницы — это обычные Web-страницы, включающие в себя особые серверные сценарии, выполняемые самим Web-сервером или специальной серверной программой (CGI-приложением или расширением Web-сервера). В частности, ASP (Active Server Pages — активные серверные страницы), поддерживаемые Microsoft Internet Information Server, и JSP (Java Server Pages — серверные страницы, написанные на JavaScript), поддерживаемые рядом других Web-серверов, работают именно таким образом. Серверные страницы ASP пишутся на языках JavaScript и VBScript, а JSP — только на JavaScript.

14. Достоинства активных серверных страниц вы уже знаете: легкость и быстрота написания и простота отладки. Кроме того, поскольку активные серверные страницы -- это обычные Web-страницы с "вкраплениями" программного кода, их написание легко освоят все, кто знаком с HTML. Недостаток: относительная медлительность и повышенные требования к системным ресурсам.

15. Серверные сценарии подобны активным серверным страницам тем, что являются интерпретируемыми, однако представляют собой "чистый" программный код, без HTML-"примесей". Интерпретатор практически всегда представляет собой CGI-программу, однако ничто не мешает разработать его в виде расширения Web-сервера. Сценарии обычно пишутся на языке программирования Perl, специально предназначенном для обработки текста; также используются языки Python, JavaScript, VBScript и даже (как говорят) язык командных файлов MS-DOS. Фактически писать сценарии можно на любом языке программирования, для которого есть интерпретатор.

16. Достоинства и недостатки серверных сценариев те же, что у активных серверных страниц. Однако сценарии потребляют исключительно много системных ресурсов, даже больше, чем CGI-приложения. Ведь для обработки каждого набора данных пользователя запускается своя копия интерпретатора, а интерпретатор, в свою очередь,

расходует много ресурсов на обработку сценария. И все же, несмотря на это, сценарии — самый популярный способ создания серверных программ.

Лекция 24

Принципы построения серверной части программного обеспечения

Се́рверное программное обеспечение (англ. Server от англ.to serve — служить) в информационных технологиях — программный компонент вычислительной системы, выполняющий сервисные (обслуживающие) функции по запросу клиента, предоставляя ему доступ к определённым ресурсам или услугам.

Для успешного применения технологии «клиент-сервер» должно использоваться соответствующее программное обеспечение, включающее клиентскую и серверную части. В частности, широко используемый пакет Microsoft Office представляет собой комплекс программ для клиентского компьютера. В его состав входят: текстовый процессор Word, табличный процессор Excel, система подготовки презентаций PowerPoint, система управления базами данных Access и программа управления информацией Outlook. В связи с успехом распространения этого пакета корпорация Microsoft решила собрать воедино комплекс программ для сервера – так появился пакет MS BackOffice. В состав названного пакета входят следующие компоненты:

- Windows NT Server – сетевая операционная система;
- System Management Server – система администрирования сети; SQL Server – сервер управления базами данных;
- SNA Server – сервер для соединения с хост-компьютерами;
- Exchange Server – сервер системы электронной почты;
- Internet Information Server – сервер для работы с Internet.

Windows NT/2000 Server способна обеспечить совместное использование файлов, печатающих устройств, предоставить услуги по соединению с рабочими станциями (клиентскими компьютерами) и другой сервис. Windows NT Server целесообразно использовать в случаях, когда предполагается наличие нескольких процессоров (обычно до четырех). Кроме того, Windows NT Server обеспечивает совместное использование ресурсов многими пользователями, возможность соединения с удаленными сетями через сервис удаленного доступа – RAS (Remote Access Service), а также через средства связи с сетями других фирм (Novell, Digital Pathworks и Apple). System Management Server (SMS) позволяет сетевому администратору централизованно управлять всей сетью. При этом обеспечивается возможность администрирования каждого компьютера, подключенного к сети, включая установленное на нем программное обеспечение. SMS предоставляет следующий сервис: управление инвентаризацией программного и аппаратного обеспечения; автоматизация установки и распространения программного обеспечения, включая его обновление; удаленное устранение неисправностей и предоставление полного контроля администратору за клавиатурой, мышью и экранами всех компьютеров в сети, работающих под управлением MS-DOS или Windows; управление сетевыми приложениями. SQL Server представляет собой систему управления реляционными базами данных, использующую принципы технологии «клиент-сервер». MS SQL Server поддерживает систему обработки транзакций, систему сохранения ссылочной целостности, механизм распределенных транзакций, тиражирование данных. SNA Server обеспечивает возможность связи с IBM AS/400 и мэйнфреймами IBM (ЕС ЭВМ). Этот продукт позволяет нескольким настольным ПЭВМ, работающим под управлением MS-DOS, Windows, Windows NT, Macintosh, Unix или OS/2, «видеть» хост-компьютеры. Exchange Server обеспечивает средства передачи и приема сообщений в информационной сети организации. Этот сервис включает электронную почту (E-mail) и обмен информационными сообщениями для рабочих групп. Microsoft Exchange Server построен на принципах технологии «клиент-сервер» и масштабируется в соответствии с возрастанием вычислительных возможностей сети. Internet Information Server обеспечивает возможность создания Web-, FTP- и Gopher-серверов для сети Internet, поддерживает управление ими с помощью встроенной программы Internet Service Manager.

Серверное ПО - это комплекс программных продуктов обслуживающих клиентские запросы. Для того, что бы добиться необходимых функциональных возможностей от сервера, требуется установить и настроить соответственное ПО. Компоненту установки и поддержки ПО для работы требуется наличие Windows 2000 Server, службы каталогов Active Directory, групповой политики и ОС Windows 2000 Professional.

Принципы построения серверной части программного обеспечения

1. Сохранение автономности сервера. Т.е. клиенты не должны ограничивать доступность сервера и нарушать его целостность.

2. Сохранение автономности клиентов. Т.е. функционирование клиентов не должно зависеть от того: подключены ли они к локальному или к удаленному серверу.

3. Поддержка независимости приложения от сервера. Т.е. клиент должен вести себя одинаково независимо от платформы.

4. Доступность специфических средств сервера. Клиент должен иметь возможность запрашивать специфические функции конкретного сервера, а они обычно бываю, за исключением простеньких серверов.

5. Поддержка доступа к реальным данным. Доступ к данным должен основываться на самих данных, а не на процедурах загрузки и выгрузки файлов. Т.е. открывать и закрывать файлы – это не наша задача, нас не волнует где – в каком файле лежат эти сведения.

6. Минимум дополнительных требований к рабочим станциям для доступа к серверу. Здесь речь идет о ресурсоемкости. Т.е. ПО сервера не должно быть ресурсоемким.

7. Возможность локального прототипирования. Удаленность данных не должна препятствовать возможности прототипирования клиентских приложений. «А иначе не настроишь». Сначала настраивают локальную станцию, потом удаленный режим и.т.д. и т.д. – все по частям.

8. Полнота вариантов соединения. Клиент не должен требовать дополнительного программного обеспечения для соединения с сервером.

9. Полнота пользовательского инструментария. По сути используемое ПО разработчика должно обеспечивать создания клиента в полном объеме.

10. Полнота среды разработки приложений. Имеется в виду, что должны быть сетевые средства доступа по сети. Потому что иначе по сети нет драйверов и средств удаленного доступа типа ODBS, GDBS, DDE, специализированные драйверы, то полноценной работы не будет.

11. Открытая среда включающих языков. Как правило, многие СУБД поддерживают многие встроенные языки, хорошо, если среда разработки имеет возможность задействовать что-то при необходимости конечно. Многие там имеют СИ, ODA, Паскаль, Prolog, PL1, и т.д. и т.д., встроенные языки к которым можно иметь доступ. Хорошо если среда разработки это позволяет.

12. Следование стандартам. Чем лучше дело обстоит на клиентской и серверной стороне со стандартами, тем меньше будет возникать проблем с интеропирабельностью (совместимость).

Лекция 25

Основные задачи, выполняемые серверными программами. Инструментальные программные средства создания программ, выполняемых на стороне сервера. Их характеристика и назначение.

Серверная программа служит для приема информации от Мониторной программы. Поскольку Наблюдающая станция и Мониторная программа вынуждены исполняться на близкорасположенных ПК из-за связи по serial-порту, а сервер, вообще говоря, может быть расположен достаточно далеко, то связь между Мониторной программой и данным приложением исполняется по UDP.

Основная задача Серверной программы - находиться в готовности установления связи с Мониторной программой и, принимая от нее актуальную информацию, формировать меняющийся файл в нужном каталоге. Программа работает в режиме

сервера, посылая UDP-бродкастинг для оповещения о своём местонахождении. Таким образом осуществляется динамическое установление связи с Мониторной программой. Программа также несёт функции согласования интервалов получаемой и записываемой информации, а также подготовки информации в нужном формате в случае вынужденного разрыва связи с Мониторной программой, либо последней с Наблюдающей станцией.

Основой разработки файл-серверных приложений для локальных сетей ПК является инструментальное окружение различных персональных СУБД: FoxPro, Clipper, Paradox, Clarion, dBase, Access и др. Такие инструменты, как правило, реализованы в виде диалоговой интегрированной среды, имеющей три уровня доступа:

• программирование и создание приложений на языке, сочетающем возможности языка 3GL с некоторыми возможностями языков четвертого поколения 4GL;

• создание и ведение структуры БД и индексов, а также интерактивная генерация макетного приложения и его компонентов (меню, форм или окон, отчетов, запросов и программных модулей);

• использование диалоговой среды и генераторов конечными пользователями для создания, ведения и просмотра БД, а также формирования несложных запросов и отчетов.

Диалоговые среды поддерживают как текстовый для DOS, так и графический для Windows интерфейсы пользователя. Внедрение графического интерфейса привело к развитию объектных свойств инструментов, средств визуальной генерации программ и событийного механизма приложений. Среди инструментальных средств СУБД для персональных компьютеров (ПК) преобладают интерпретирующие системы, хотя многие предоставляют и альтернативную возможность создания загрузочных модулей приложений. Однако СУБД для ПК привлекают простотой использования и доступностью, поэтому файл-серверные приложения еще будут применяться для рабочих групп.

В настоящее время широко используются визуальные объектно-ориентированные версии инструментальных средств и СУБД на ПК (MSAccess, VisualFoxPro, CA-VisualObjects и VisualdBase). Эти продукты направлены сугубо на создание Windows-приложений и содержат средства объектно-ориентированного диалога, событийно-управляемого программирования, визуального конструирования интерфейса пользователя и многие другие черты, присущие системам программирования 4GL и средствам быстрой разработки RAD. Кроме того, они поддерживают структурный язык запросов SQL, который характерен для приложений клиент-сервер.

Архитектура клиент-сервер спроектирована для решения проблем файл-серверных приложений путем разделения компонент приложения и размещения их там, где они будут функционировать более эффективно. Особенностью архитектуры клиент-сервер является использование выделенных серверов баз данных, понимающих запросы на языке структурированных запросов SQL и выполняющих поиск, сортировку и агрегирование информации на месте без излишней перекачки данных на рабочие станции. Клиенты серверов БД получают последовательно и порциями только результаты запросов. Для реализации серверов БД используется системное программное обеспечение (ПО) реляционных СУБД, понимающих язык запросов SQL, например: *Oracle, Informix, Sybase, MSSQLServer.*Многие из этих СУБД работают на различных аппаратных платформах и в средах разных ОС.

Объектами разработки в приложениях клиент-сервер помимо диалога и логики обработки являются прежде всего реляционная модель данных и связанный с ней набор SQL-операторов для типовых запросов для этой БД.

Большинство конфигураций клиент-сервер используют двухзвен-ную модель, состоящую из сервера и клиента, который обращается к услугам сервера. Для эффективной реализации такой схемы часто применяют неоднородную сеть.

Группу инструментальных средств для создания информационных приложений с архитектурой клиент-сервер можно разделить на следующие подгруппы:

• среды разработки приложений для серверов баз данных;

• независимые от СУБД инструменты для создания приложений клиент-сервер;

• средства поддержки распределенных информационных приложений.

Среды разработки приложений для серверов БД представляют собой системы программирования четвертого поколения 4GL или инструментальные средства быстрой разработки приложений RAD *(RapidApplication Development)*. Особенностями этой подгруппы средств являются:

• реализация удаленного доступа к СУБД по двухзвенной схеме клиент-сервер;

• связь клиентских приложений с серверами БД с помощью непроцедурного языка структурированных запросов SQL (кроме серверов Btrieve);

• обеспечение целостности БД, включая целостность транзакций;

• поддержка хранимых процедур на серверах БД;

• реализация клиентских и серверных триггеров-процедур;

• генерация элементов диалогового интерфейса и отчетов.

В качестве примера можно назвать инструменты Informix/4GL, Oracle*Forms и др.

Независимые инструментальные средства, ориентированные на многие платформы СУБД, представлены в виде средств быстрой разработки приложений RAD. Для таких средств создания приложений клиент-сервер характерны:

• возможность распределения приложения на клиентах и (или) серверах;

• создание приложений для разных серверов БД;

• поддержка спецификации *ODBC (Open DataBase Connectivity)* для доступа к различным серверам БД, включая СУБД для ПК;

• связь с мониторами транзакций для организации трехзвенной архитектуры приложений клиент-сервер;

• объектно-ориентированное программирование приложений;

• визуальный характер генерации приложения;

• ведение репозитория объектов и их свойств, что облегчает интеграцию со средствами автоматизации проектирования программ CASE;

• управление проектами и версиями приложений;

• интеграция приложения с электронной почтой и средствами офисной автоматизации.

Известными примерами независимых инструментальных средств разработки являются *PowerBuilder, JAM, Uniface* и др.

Средства поддержки распределенных приложений относятся к категории промежуточного программного обеспечения (*middleware)* для организации серверов приложений. Сюда входят разнообразные библиотеки и наборы инструментальных средств:

• интерфейсы доступа к базам данных *ODBC и IDAPI;*

• шлюзы для систем управления базами данных;

• протоколы и команды мониторов обработки транзакций;

• почтовые интерфейсы *MAPI, VIM, MHS, X.400н EDI;*

• средства обмена сообщениями *MOM;*

• протоколы связывания и встраивания объектов *OLE* и динамического обмена данными *DDE;*

• протоколы удаленного вызова процедур *RPC* и именованных конвейеров NaraedPipes;

• средства коммуникационного ввода-вывода *BSDSockets* и *WinSock.* Самостоятельную группу инструментальных средств составляют объектно-ориентированные базы данных, которые используют для хранения объектов реляционные БД или применяют специальные хранилища объектов.

Лекция 26
Спецификация CGI (Common Gateway Interface). CGI-скрипт. Шлюз CGI. Препроцессор.

CGI (от англ. *Common Gateway Interface* — «общий интерфейс шлюза») — стандарт интерфейса, используемого для связи внешней программы с веб-сервером. Программу, которая работает по такому интерфейсу совместно с веб-сервером, принято

называть шлюзом, хотя многие предпочитают названия «скрипт» (сценарий) или «CGI-программа».

Сам интерфейс разработан таким образом, чтобы можно было использовать любой язык программирования, который может работать со стандартными устройствами ввода-вывода. Такими возможностями обладают даже скрипты для встроенных командных интерпретаторов операционных систем, поэтому в простых случаях могут использоваться даже командные скрипты.

Все скрипты, как правило, помещают в каталог cgi (или cgi-bin) сервера, но это необязательно: скрипт может располагаться где угодно, но при этом большинство веб-серверов требуют специальной настройки. В веб-сервере Apache, например, такая настройка может производиться при помощи общего файла настроек httpd.conf или с помощью файла .htaccess в том каталоге, где содержится этот скрипт.

CGI является одним из наиболее распространённых средств создания динамических веб-страниц.

Спецификация CGI была разработана в Центре Суперкомпьютерных Приложений Унив-ерситета штата Иллинойс (NCSA). Работы над ней велись параллельно с Mosaic. С точки зрения общей архитектуры программного обеспечения World Wide Web, CGI определила все дальнейшее развитие системных средств. До появления этой спецификации все новые возможности реализовывались в виде модулей, включенных в библиотеку общих кодов ЦЕРН. Разработчики серверов должны были использовать эти коды для реализации программ или заменять их своими собственными аналогами. Это означало, что после компиляции сервера добавить в него новые возможности будет невозможно. CGI в корне изменила эту практику.

Главное назначение Common Gateway Interface - обеспечение единообразного потока данных между сервером и прикладной программой, которая запускается из-под сервера. CGI определяет протокол обмена данными между сервером и программой. Для тех, кто знаком с протоколом HTTP, может показаться, что CGI - это просто подмножество этого протокола. Однако это не так. Во-первых, CGI определяет порядок взаимодействия сервера с прикладной программой, в котором сервер выступает инициирующей стороной, во-вторых, CGI определяет механизм реального обмена данными и управляющими командами в этом взаимодействии, что не определено в HTTP. Естественно, что такие понятия, как метод доступа, переменные заголовка, MIME, типы данных, заимствованы из HTTP и делают спецификацию прозрачной для тех, кто знаком с самим протоколом.

При описании различных программ, которые вызываются сервером HTTP и реализованы в стандарте CGI, используют следующую терминологию:

CGI-скрипт - программа, написанная в соответствии со спецификацией Common Gateway Interface. CGI-скрипты могут быть написаны на любом языке программирования (C, C++, PASCAL, FORTRAN и т.п.) или командном языке (shell, cshell, командный язык MS-DOS, Perl и т.п.). Скрипт может быть написан даже на языке редактора EMAC в системах Unix.

Шлюз - это CGI-скрипт, который используется для обмена данными с другими информационными ресурсами Internet или приложениями-демонами. Обычная CGI-программа запускается сервером HTTP для выполнения некоторой работы, возвращает результаты серверу и завершает свое выполнение. Шлюз выполняется точно также, только, фактически, он инициирует взаимодействие в качестве клиента с третьей программой. Если эта третья программа является сервисом Internet, например, сервер Gopher, то шлюз становится клиентом Gopher, который посылает запрос по порту Gopher, а после получения ответа пересылает его серверу HTTP.

Собственно спецификация CGI описывает четыре набора механизмов обмена данными:

- через переменные окружения;
- через командную строку;
- через стандартный ввод;
- через стандартный вывод

Последовательность работы *CGI* состоит из следующих этапов:

1. Получение Web-сервером информации от клиента-браузера. Для передачи данных Web-серверу используются формы. Форма задается в HTML-документе при помощи тэгов <form> … </form> и состоит из набора полей ввода, отображаемых браузером в виде графических элементов управления: селекторных кнопок, опций, строк ввода текста, управляющих кнопок и т. д. Обычно одна из кнопок предназначена для завершения ввода. Когда пользователь заполнит всю форму, он нажимает эту кнопку, и данные из полей формы передаются программе *CGI*.
2. Анализ и обработка полученной информации. Данные, извлеченные из *HTML-формы*, передаются для обработки *CGI*-программе. Они не всегда могут быть обработаны *CGI*-программой самостоятельно. Например, они могут содержать запрос к базе данных. В этом случае *CGI*-программа на основании полученной информации формирует запрос к ядру СУБД, выполняющейся на том же или удаленном компьютере.
3. Создание нового HTML-документа и пересылка его браузеру. После обработки полученной информации *CGI*-программа создает динамический (виртуальный) HTML-документ, или формирует ссылку на уже существующий документ и передает результат браузеру.

CGI программа, имеет свою специфику, заключающуюся в том, что она, как правило, генерирует HTML-документ, посылаемый клиенту в виде ответа сервера. Ответ сервера, так же как и запрос клиента, имеет определенную структуру. Он состоит из следующих трех частей:

1. Строка состояния, содержащая три поля: номер версии протокола HTTP, код состояния и краткое описание состояния, например:
2. HTTP/1.0 200 OK # Запрос клиента обработан успешно

HTTP/1.0 404 Not Found # Документ по указанному адресу не существует

3. Заголовки ответа, содержащие информацию о сервере и о возвращаемом HTML-документе, например:
4. Date: Mon, 26 Jul 1999 18:37:07 GMT # Текущая дата и время
5. Server: Apache/1.3.6 # Имя и номер версии сервера

Content-type: text/html # Описывает медиа-тип содержимого

6. Содержимое ответа — HTML-документ, являющийся результатом выполнения *CGI*-программы.

CGI-программа передает результат своей работы — HTML-документ — серверу, который возвращает его клиенту. *Шлюз*осуществляет свой вывод в *стандартный поток вывода*. Этот вывод может представлять собой или документ, сгенерированный*шлюзом*, или инструкции серверу, где получить необходимый документ. При этом сервер не анализирует и не изменяет полученные данные, он может только дополнять их некоторыми заголовками, содержащими общую информацию (например, текущая дата и время) и информацию о самом себе (например, имя и версия сервера).

В зависимости от метода данные формы передаются в *CGI*-программу или через *стандартный ввод* (POST), или через*переменную среды QUERY_STRING* (GET). Помимо этих данных *CGI*-программе доступна и другая информация, поступившая от клиента в заголовках запроса или предоставленная Web-сервером.

Информация шлюзам передается в следующей форме:

имя=значение&имя1=значение1&..,

где имя- имя переменной (из оператора FORM, например), и значение - ее *реальное значение*. В зависимости от метода, который используется для запроса, эта строка появляется или как часть URL (в случае метода GET), или как содержимое HTTP запроса (метод POST). В последнем случае, эта информация будет послана шлюзу в стандартный поток ввода.

Данные формы поступают в *CGI*-программу в закодированном виде, поэтому в качестве первого шага обработки *CGI*-сценарий должен выполнить *декодирование* полученной информации.

Таким образом, *декодирование* данных сводится к следующей последовательности манипуляций со строкой:

1. замена каждой группы %hh, состоящей из шестнадцатеричного ASCII-кода hh с префиксом %, на соответствующий ASCII-символ;
2. замена символов + пробелами;
3. выделение отдельных пар имя=значение, разделенных ограничителем & ;
4. выделение из каждой пары имя— значение имени и значения соответствующего поля формы.

Perl

CGI-программа может быть написана на любом языке программирования, имеющем средства обмена данными между программами.

Для создания абсолютно мобильных программ *CGI* лучше всего воспользоваться языком Perl. Интерпретаторы этого языка созданы практически для всех операционных систем. Этот *язык высокого уровня* содержит многие функции, упрощающие создание программ *CGI*. В сети Internet доступны (причем бесплатно) версии интерпретатора Perl для различных платформ (в том числе и для Microsoft *Windows NT*), разнообразная документация и примеры программ.

Программа на языке Perl представляет собой *последовательность операторов*, которые интерпретатор языка выполняет при каждом запуске без преобразования исходного текста программы в выполняемый двоичный код. По этой причине *CGI*-программы называют также *CGI*-сценариями или *CGI-скриптами*.

PHP

Для динамического формирования документов HTML используется также технология PHP (*Hypertext Preprocessor*). PHP часто еще называют *препроцессором гипертекста* (*Hypertext Preprocessor*). По сути PHP серверный (выполняющийся на стороне сервера) мультиплатформный язык описания сценариев, встраиваемый непосредственно в HTML-код.

Основными сферами применения данной технологии являются создание *серверных приложений* и интерфейсов к базам данных.

Приложения PHP встраиваются в документы HTML в качестве сценариев, подобно сценариям JavaScript, однако, в отличие от JavaScript, сценарии PHP выполняются не *пользовательским агентом*, а специальной программой – проигрывателем сценариев PHP, которая является приложением *CGI* и запускается сервером. Пользовательский агент получает результат выполнения запрошенного им сценария и не может получить доступ к исходному коду самого сценария.

До того, как сервер пересылает опубликованный файл браузеру, его просматривает препроцессор-интерпретатор. Для этого файлы имеют специальное расширение .phtml или *php3*. Если страница содержит помимо HTML php-код, то он выполняется и результат отправляется браузеру.

Преимуществами PHP является бесплатность и *кроссплатформенность*. Недостаток – плохая масштабируемость. PHP непригоден для использования в сложных проектах. Это связано со следующими особенностями:

1. падение производительности при обработке больших скриптов;
2. PHP – интерпретируемый язык и уступает по скорости работы компилируемым технологиям;
3. в *php3* нет поддержки сессий, как, например, в ASP.

Лекция 27
Расширения ISAPI и их преимущества. Серверы ASP.

До сегодняшнего дня было немало методик работы с динамическим содержимым Web. Самой старой из них, которая поддерживается большинством HTTP-серверов, является общий шлюзовой интерфейс (CGI). С помощью переменных среды, CGI-программа принимает данные заголовка HTTP-запроса, отправляемые с Web-броузера. Кроме того, такая программа, пользуясь перенаправленным стандартным входным потоком, принимает данные запроса через Web-сервер, а затем генерирует динамический HTML для передачи клиенту через тот же Web-сервер, используя перенаправленный стандартный выходной поток. Конечно, применение переменных среды, стандартных

входного и выходного потоков является способом не слишком эффективным, зато работающим одинаково на большинстве Web-серверов, размещенных как в Unix, так и в Windows. К сожалению, для каждого клиентского запроса общий шлюзовой интерфейс (CGI) создает новый процесс, а это слишком дорого обходится с точки зрения производительности и использования ресурсов. Так что эффективно обработать большое количество клиентских запросов с помощью общего шлюзового интерфейса (CGI) не удается. К тому же, Web-сервер и CGI-процесс находятся в разных адресных пространствах. Поэтому общий шлюзовой интерфейс (CGI) страдает еще и от накладных расходов, вызываемых большим количеством взаимодействий между процессами.

Что касается интерфейса прикладного программирования Internet-сервера ISAPI, который представляет собой технологию на основе информационного сервера Internet (Internet Information Server, US), то он решает проблемы с производительностью и масштабированием, выполняя вместо CGI-процесса встроенную в процесс динамически подсоединяемую библиотеку (DLL) интерфейса прикладного программирования Internet-сервера (ISAPI), По требованию эта библиотека загружается в адресное пространство информационного сервера Internet (IIS) и совместно используется многими W/eb-клиентами. Интерфейс прикладного программирования Internet-сервера (ISAPI) содержит интерфейсы прикладного программирования (API), которые необходимы для доступа к клиентским запросам, и, в частности, ко входным параметрам и HTTP-заголовкам. Для чтения данных, находящихся в запросах, интерфейс прикладного программирования Internet-сервера (ISAPI) организует входной поток, а для отправки клиенту, в ответ на его запрос, динамически созданного HTML-кода — выходной поток. Такой подход намного эффективнее, чем использование общего шлюзового интерфейса CGI, потому что новый процесс не создается, и, следовательно, взаимодействие процессов на сервере не требуется. В среде Visual C++ 6.0 имеется ISAPI Extension Wizard (Мастер создания расширений интерфейса прикладного программирования Internet-сервера), предназначенный для создания на основе MFC расширений интерфейса прикладного программирования Internet-сервера. Кроме того, там имеются проекты фильтров для интерфейса прикладного программирования Internet-сервера (ISAPI). Для работы с интерфейсом прикладного программирования Internet-сервера (ISAPI) в библиотеке MFC предусмотрен CHttpServer, исполняющий роль оболочки в интерфейсе прикладного программирования Internet-служб, а также макрос для создания карты сообщений и макросы для различных преобразований в ходе грамматического разбора.

Фильтры и расширения интерфейса прикладного программирования Internet-сервера (ISAPI) прекрасно подходили для создания высокопроизводительных Web-приложений, однако требовали от программистов достаточно высокой квалификации. Этим специалистам нужно было не только знать C++, но также понимать и использовать организацию пула с помощью процессов, синхронизацию, обработку транзакций и безопасность. Помимо этого, программисту требовалось быть очень осторожным и перед размещением динамически подсоединяемой библиотеки (DLL) интерфейса прикладного программирования Internet-сервера (ISAPI), приходилось проводить большое количество испытанийСДёло в том, что любая достаточно серьезная ошибка в такой библиотеке, работающей в рамках того же процесса, что и информационный сервер Internet, привела бы к аварийному завершению всего процесса (то есть к аварийному завершению работы Web-сервера OS). И, наконец, большинство программистов возненавидело интерфейс прикладного программирования Internet-сервера (ISAPI). Причина здесь в том, что фильтры и расширения интерфейса прикладного программирования Internet-сервера (ISAPI) труднее поддаются отладке, чем другие программы, написанные на C++, так как необходимо подсоединяться к работающему процессу, в рамках которого выполняется информационный сервер Internet (П5).

ASP (англ. *Active Server Pages* — «активные серверные страницы») — технология, предложенная компаниейMicrosoft в 1996 году для создания Web-приложений. Эта технология основана на внедрении в обыкновенныевеб-страницы специальных элементов управления, допускающих программное управление.

По своей сути, ASP — это технология динамического создания страниц на стороне сервера, приблизившая проектирование и реализацию Web-приложений к той модели, по которой проектируются и реализуются обычные приложения.
Для реализации приложений ASP используются языки сценариев (VBScript или JScript). Также допускается применение COM-компонентов.
Технология ASP разработана для операционных систем из семейства Windows NT и функционирует под управлением веб-сервера Microsoft IIS.
Технология ASP получила своё развитие в виде ASP.NET — технологии создания веб-приложений, основанной уже на платформе Microsoft .NET.

Что касается технологии ASP, то для разработки на высоком уровне динамического Web-содержимого она является достаточно удобным средством. Эта технология использует интерфейс прикладного программирования Internet-сервера (ISAPI), но при этом не столь эффективна, как интерфейс прикладного программирования Internet-сервера (ISAPI) в чистом виде. ASP фактически реализована в виде заранее подготовленной, достаточно универсальной динамически подсоединяемой библиотеки (DLL) интерфейса прикладного программирования Internet-сервера (ISAPI), которая реализует машину сценариев. В свою очередь машина сценариев потом интерпретирует ASP-страницу. Такая страница похожа на обычную HTML-страницу, за исключением того, что содержит встроенные фрагменты ASP-кода, написанные в виде сценария. Этот код не компилируется, а интерпретируется машиной сценариев. Для интерпретации можно использовать любой язык подготовки сценариев, который установлен на Web-сервере, поддерживающем ASP. По умолчанию информационный сервер Internet (П5) автоматически поддерживает VBScript и JScript. Недостатком ASP является то, что интерпретируемый код, естественно, работает медленнее, чем выполняющий те же действия скомпилированный код, в котором используется интерфейс прикладного программирования Internet-сервера (ISAPI).

Другим недостатком ASP является то, что языки подготовки сценариев не обеспечивают типовую безопасность, что может привести к ошибкам времени выполнения, которые было бы лучше предотвратить еще во время компиляции. Ну и, наконец, языки подготовки сценариев не являются объектно-ориентированными и потому не могут служить в качестве языков программирования высокого^ уровня, используемых для программирования больших систем. А вот огромным преимуществом ASP является то, что языки подготовки сценариев обычно очень легкие для1 изучения. В частности, VBScript известен огромному числу людей. То, что языки подготовки сценариев, подобные VBScript, не являются объектно-ориентированными, частично компенсируется их способностью вызывать серверные COM-компоненты. Дело в том, что эти компоненты можно писать на мощных объектно-ориентированных языках, таких, как C++. Другое преимущество ASP перед интерфейсом прикладного программирования Internet-сервера (ISAPI) состоит в том, что ASP прекрасно подходит для работы с сервером транзакций корпорации Microsoft (Microsoft Transaction Server, MTS). Это дает возможность автоматизировать организацию поточной обработки, а также управление синхронизацией, обработкой транзакций и безопасностью.

Технология ASP.NET, к счастью, сохраняет все преимущества традиционной ASP и избавлена от большинства ее недостатков, в том числе и снижения производительности. Вместо интерпретатора языков подготовки сценариев ASP.NET использует компилируемые языки платформы .NET, такие, например, как C#, VB.NET и даже управляемый C++.

Таким образом, теперь уже можно выбирать из нескольких технологий. Это общий шлюзовой интерфейс (CGI), интерфейс прикладного программирования Internet-сервера (ISAPI), ASP и ASP.NET. Ну да, есть еще и библиотека шаблонных классов ATL. Так же, как и ASP.NET, библиотека шаблонных классов ATL дает возможность разрабатывать как Web-узлы, так и Web-службы. Впрочем, серверы и службы на основе библиотеки шаблонных классов ATL создаются с помощью библиотеки шаблонов для C++, разработанной на основе технологии ISAPI (интерфейс прикладного программирования Internet-сервера).

Лекция 28
Характеристика инструментального программного средства.

Инструментальное программное обеспечение – это пакет программ, предназначенных для автоматизации создания, редактирования, отладки, тестирования различных программных продуктов.

К этой категории относятся программы, предназначенные для разработки программного обеспечения:

- ассемблеры — компьютерные программы, осуществляющие преобразование программы в форме исходного текста на языке ассемблера в машинные команды в виде объектного кода.
- трансляторы — программы или технические средства, выполняющие трансляцию программы.
- компиляторы — Программы, переводящие текст программы на языке высокого уровня, в эквивалентную программу на машинном языке.
- интерпретаторы — Программы (иногда аппаратные средства), анализирующие команды или операторы программы и тут же выполняющие их
- компоновщики (редакторы связей) — программы, которые производят компоновку — принимают на вход один или несколько объектных модулей и собирают по ним исполнимый модуль.
- препроцессоры исходных текстов — это компьютерные программы, принимающие данные на входе и выдающие данные, предназначенные для входа другой программы, например, такой, как компилятор
- отла́дчик (debugger) является модулем среды разработки или отдельным приложением, предназначенным для поиска ошибок в программе.
- текстовые редакторы — компьютерные программы, предназначенные для создания и изменения текстовых файлов, а также их просмотра на экране, вывода на печать, поиска фрагментов текста и т. п.
- специализированные редакторы исходных текстов — текстовые редакторы для создания и редактирования исходного кода программ. Специализированный редактор исходных текстов может быть отдельным приложением, или быть встроен в интегрированную среду разработки (IDE).
- библиотеки подпрограмм — сборники подпрограмм или объектов, используемых для разработки программного обеспечения.
- редакторы графического интерфейса.

Перечисленные инструменты могут входить в состав интегрированных сред разработки

Виды инструментального ПО

- Текстовые редакторы- самостоятельная компьютерная программа или компонент программного комплекса (например, редактор исходного кода интегрированной среды разработки или окно ввода в браузере), предназначенная для создания и изменения текстовых данных в общем и текстовых файлов в частности.Текстовые редакторы предназначены для работы с текстовыми файлами в интерактивном режиме. Они позволяют просматривать содержимое текстовых файлов и производить над ними различные действия — вставку, удаление и копирование текста, контекстный поиск и замену, сортировку строк, просмотр кодов символов и конвертацию кодировок, печать и т. п.Часто интерактивные текстовые редакторы содержат дополнительную функциональность, призванную автоматизировать действия по редактированию (от записываемых последовательностей нажатий клавиш до полноценных встроенных языков программирования), или отображают текстовые данные специальным образом (например, с подсветкой синтаксиса).
- Интегрированные среды разработки- система программных средств, используемая программистамидля разработки программного обеспечения (ПО).
- SDK- — комплект средств разработки, который позволяет специалистам по программному обеспечению создавать приложения для определённого пакета

программ, программного обеспечения базовых средств разработки, аппаратной платформы, компьютерной системы, игровых консолей, операционных систем и прочих платформ.Программист, как правило, получает SDK непосредственно от разработчика целевой технологии или системы. Часто SDK распространяется черезИнтернет. Многие SDK распространяются бесплатно для того, чтобы побудить разработчиков использовать данную технологию или платформу.Поставщики SDK иногда подменяют слово «software» в словосочетании «software development kit» на более точное слово. Например, Microsoft и Applepредоставляют Driver Development Kit (DDK) для разработки драйверов устройств, PalmSource называет свой инструментарий для разработки PalmOS Development Kit (PDK), а Oracle — Java Development Kit (JDK).

- Компиляторы- программа, выполняющая компиляцию.Компиляция — трансляция программы, составленной на исходном языке высокого уровня, в эквивалентную программу на низкоуровневом языке, близком машинному коду (абсолютный код, объектный модуль, иногда на язык ассемблера). Входной информацией для компилятора (исходный код) является описание алгоритма или программа на проблемно-ориентированном языке, а на выходе компилятора — эквивалентное описание алгоритма на машинно-ориентированном языке (объектный код). Компилировать — проводить трансляцию машинной программы с проблемно-ориентированного языка на машинно-ориентированный язык
- Интерпретаторы -программа (разновидность транслятора), выполняющая интерпретацию. Интерпрета́ция — пооператорный (покомандный, построчный) анализ, обработка и тут же выполнение исходной программы или запроса (в отличие откомпиляции, при которой программа транслируется без её выполнения)
- Линковщики- в сфере разработки программного обеспечения, компоновщик (также редактор связей или ли́нкер, от англ. link editor, linker) — это инструментальная программа, которая производит компоновку («линковку»): принимает на вход один или несколько объектных модулей и собирает по ним исполнимый модуль. Изначально, до появления динамических библиотек, загрузчики могли выполнять некоторые функции компоновщика, однако сейчас, чаще всего, загрузка программ выделяется в отдельный процесс
- Парсеры и генераторы парсеров (см. Javacc)- Синтакси́ческий ана́лиз (жарг. па́рсинг) в лингвистике и информатике — процесс сопоставления линейной последовательности лексем (слов, токенов)естественного или формального языка с его формальной грамматикой. Результатом обычно является дерево разбора (синтаксическое дерево). Обычно применяется совместно с лексическим анализом.

Синтаксический анализатор (жарг. па́рсер) — это программа или часть программы, выполняющая синтаксический анализ.

Пример разбора выражения в дерево

В ходе синтаксического анализа исходный текст преобразуется в структуру данных, обычно — в дерево, которое отражает синтаксическую структуру входной последовательности и хорошо подходит для дальнейшей обработки.

- Ассемблеры- (от англ. assembler — сборщик) — компилятор исходного текста программы, написанной на языке ассемблера, в программу на машинном языке. Как и сам язык, ассемблеры, как правило, специфичны для конкретной архитектуры, операционной системы и варианта синтаксиса языка. Вместе с тем существуют мультиплатформенные или вовсе универсальные (точнее, ограниченно-универсальные, потому что на языке низкого уровня нельзя написать аппаратно-независимые программы) ассемблеры, которые могут работать на разных платформах и операционных системах. Среди последних можно также выделить группу кросс-ассемблеров, способных собирать машинный код и исполняемые модули (файлы) для других архитектур и ОС.

- Отладчики (деба́ггер, англ. debugger) — компьютерная программа, предназначенная для поиска ошибок в других программах, ядрах операционных систем, SQL-запросах и других видах кода. Отладчик позволяет выполнять трассировку, отслеживать, устанавливать или изменять значения переменныхв процессе выполнения кода, устанавливать и удалять контрольные точки или условия остановки и т.д.
- Профилировщики
- Генераторы документации- программа или пакет программ, позволяющая получать документацию, предназначенную для программистов(документация на API) и/или для конечных пользователей системы, по особым образом комментированному исходному коду и, в некоторых случаях, поисполняемым модулям (полученным на выходе компилятора). Обычно генератор анализирует исходный код программы, выделяя синтаксические конструкции, соответствующие значимым объектам программы (типам, классам и их членам/свойствам/методам, процедурам/функциям и т. п.). В ходе анализа также используется мета-информация об объектах программы, представленная в виде документирующих комментариев. На основе всей собранной информации формируется готовая документация, как правило, в одном из общепринятых форматов — HTML, HTMLHelp, PDF, RTF и других.
- Средства анализа покрытия кода- Покры́тие ко́да — мера, используемая при тестировании программного обеспечения. Она показывает процент, насколько исходный код программы был протестирован.
- Средства непрерывной интеграции- Непрерывная интеграция (CI, англ. Continuous Integration) — это практика разработки программного обеспечения, которая заключается в выполнении частых автоматизированных сборок проекта для скорейшего выявления и решения интеграционных проблем. В обычном проекте, где над разными частями системы разработчики трудятся независимо, стадия интеграции является заключительной. Она может непредсказуемо задержать окончание работ. Переход к непрерывной интеграции позволяет снизить трудоёмкость интеграции и сделать её более предсказуемой за счет наиболее раннего обнаружения и устранения ошибок и противоречий
- Средства автоматизированного тестирования
- Системы управления версиями- (от англ. Version Control System, VCS или Revision Control System) — программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы. Однако они могут с успехом применяться и в других областях, в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов. В частности, системы управления версиями применяются в САПР, обычно в составе систем управления данными об изделии (PDM). Управление версиями используется в инструментах конфигурационного управления (Software Configuration Management Tools).
- и др.

Трансляция программы (компиляция и интерпретация)

С помощью языка программирования создается не готовая программа, а только ее текст, описывающий разработанный алгоритм. **Текст алгоритма задачи, записанный на любом языке программирования называется исходным модулем.** Специальная программа - транслятор переводит исходный модуль в последовательность команд ЭВМ. Имеются два основных типа таких программ - трансляторов: компиляторы и интерпретаторы. **Компилятор** транслирует весь текст исходного модуля в машинный код, который называется **объектным модулем** за один непрерывный процесс. Компилятор выдает промежуточный объектный код - двоичный файл с расширением .OBJ. Объектный модуль еще не может выполняться, т.к. он может содержать неразрешенные ссылки на другие модули или программы, а также перемещаемый код. К нему еще нужно добавить машинный код подпрограмм, реализующий стандартные

функции (например, математические). Эти функции содержатся в стандартных библиотеках - файлах с расширением **.LIB**. Поэтому объектный модуль обрабатывается специальной программой – редактором связей. Редактор связи разрешает все внешние ссылки и создает загрузочный модуль. Далее начинает работу программа Загрузчик, она определяет для загрузочного модуля абсолютные адреса в ОП. Только после всех этих действий программ может выполняться. Часто функции редактора и загрузчика выполняет одна программа – редактирующий загрузчик. Итак, объектный код обрабатывается специальной программой - редактором связей или сборщиком, который выполняет связывание объектных модулей и машинного кода стандартных функций, находя их в стандартных библиотеках, и формирует на выходе работоспособное приложение - исполнимый код.

Итак, чтобы получить работающую программу, надо текст программы, называемый исходным модулем, перевести в объектный модуль, пригодный для последующего редактирования и выполнения на ЭВМ.

Рассмотрим подробнее работу программы-компилятора

Программа-компилятор автоматически переводит исходный текст в машинный код, который потом используется отдельно от исходного текста. В общем случае работа компилятора состоит из 4-ч фаз.

1. Лексический анализ. На основе исходного модуля идентифицируются различные символы и классифицируются по категориям: ключевые слова, числовые значения, идентификаторы переменных и т.д.

2. Синтаксический анализ. При этом определяются синтаксические отношения ключевых слов и строится структурный каркас программы.

3. Генерация объектного кода, который соответствует структуре программы.

4. Оптимизация объектного модуля с целью повышения его эффективности (по объему и быстродействию).

Компиляторы делают исходную программу компактной, эффективной, работающей в сотни раз быстрее программы, выполненной с помощью интерпретатора.

Исполнимый код - это законченная программа, которую можно запустить на любом компьютере, где установлена ОС, для которой эта программа создавалась. Итоговый файл имеет расширение **.EXE** или **.COM**.

Программа-интерпретатор сразу выполняет команды языка, указанные в тексте программы. Интерпретатор берет очередной оператор языка из текста программы, анализирует его и если все правильно, сразу же исполняет его. Только после успешного выполнения текущего оператора интерпретатор перейдет к следующему оператору. При выполнении одного оператора многократно, интерпретатор каждый раз работает с ним так, словно встретил этот оператор впервые. Программы с большим объемом повторяющихся операторов будут работать медленно. Интерпретатор удобен при изучении инструментов программирования, т.к. позволяет понять принцип работы отдельного оператора языка.

В отличии от компилятора интерпретатор не создает объектный код, а выполняет исходный модуль программы в режиме «оператор за оператором», по ходу работы он превращает каждый оператор ЯВУ в машинные команды.

Лекция 29
Функциональные возможности программного средства.

Язык PHP был разработан как инструмент для решения чисто практических задач. Его создатель, Расмус Лердорф, хотел знать, сколько человек читают его online-резюме, и написал для этого простенькую CGI-оболочку на языке Perl, т.е. это был набор Perl-скриптов, предназначенных исключительно для определенной цели – сбора статистики посещений.

Для справки. CGI (Common Gateway Interface – общий интерфейс шлюзов) является стандартом, который предназначен для создания серверных приложений, работающих по протоколу HTTP. Такие приложения (их называют шлюзами или CGI-программами)

запускаются сервером в режиме реального времени. Сервер передает запросы пользователя CGI-программе, которая их обрабатывает и возвращает результат своей работы на экран пользователя. Таким образом, посетитель получает динамическую информацию, которая может изменяться в результате влияния различных факторов. Сам шлюз (скрипт CGI) может быть написан на различных языках программирования – Си/C++, Fortran, Perl, TCL, UNIX Shell, Visual Basic, Python и др.

Вскоре выяснилось, что оболочка обладает небольшой производительностью, и пришлось переписать ее заново, но уже на языке Си. После этого исходники были выложены на всеобщее обозрение для исправления ошибок и дополнения. Пользователи сервера, где располагался сайт с первой версией PHP, заинтересовались инструментом, появились желающие его использовать. Так что скоро PHP превратился в самостоятельный проект, и в начале 1995 года вышла первая известная версия продукта, называвшаяся Personal Home Page Tools (средства для персональной домашней страницы). Средства эти были более чем скромными: анализатор кода, понимающий всего лишь несколько специальных команд, и набор утилит, полезных для создания гостевой книги, счетчика посещений, чата и т.п.

К середине 1995 года после основательной переработки появилась вторая версия продукта, названная PHP/FI (Personal Home Page / Forms Interpreter – персональная домашняя страница/ интерпретатор форм). Она включала набор базовых возможностей сегодняшнего PHP, возможность автоматически обрабатывать html-формы и встраиваться в html-коды. Синтаксис PHP/FI сильно напоминал синтаксис Perl, но был более простым.

В 1997 вышла вторая версия Си-реализации PHP – PHP/FI 2.0. К тому моменту PHP использовали уже несколько тысяч людей по всему миру, примерно с 50 тыс. доменов, что составляло около 1% всего числа доменов Internet. Число разработчиков PHP увеличилось до нескольких человек, но, несмотря на это, PHP/FI 2.0 все еще оставался крупным проектом одного человека. Официально PHP/FI 2.0 вышел только в ноябре 1997 года, просуществовав до этого в основном в бета-версиях. Вскоре после выхода его заменили альфа-версии PHP 3.0.

PHP 3.0 была первой версией, напоминающей PHP, каким мы знаем его сегодня. Он очень сильно отличался от PHP/FI 2.0 и появился опять же как инструмент для решения конкретной прикладной задачи. Его создатели, Энди Гутманс (Andi Gutmans) и Зив Сураски (Zeev Suraski), в 1997 году переписали заново код PHP/FI, поскольку он показался им непригодным для разработки приложения электронной коммерции, над которым они работали. Для того чтобы получить помощь в реализации проекта от разработчиков PHP/FI, Гутманс и Сураски решили объединиться с ними и объявить PHP3 официальным преемником PHP/FI. После объединения разработка PHP/FI была полностью прекращена.

Одной из сильных сторон PHP 3.0 была возможность расширения ядра. Именно свойство расширяемости PHP 3.0 привлекло внимание множества разработчиков, желающих добавить свой модуль расширения. Кроме того, PHP 3.0 предоставляла широкие возможности для взаимодействия с базами данных, различными протоколами и API. Немаловажным шагом к успеху оказалась разработка нового, намного более мощного и полного синтаксиса с поддержкой ООП. С момента появления PHP 3.0 изменилась не только функциональность и внутреннее устройство языка, но и его название. В аббревиатуре PHP больше не было упоминания о персональном использовании, PHP стало сокращением (рекурсивным акронимом) от PHP: Hypertext Preprocessor, что значит «PHP: препроцессор гипертекста».

К концу 1998 года число пользователей PHP возросло до десятков тысяч. Сотни тысяч web-сайтов сообщали о том, что они работают с использованием этого языка. Почти на 10% серверов Internet был установлен PHP 3.0.

Официально PHP 3.0 вышел в июне 1998 года, после 9 месяцев публичного тестирования. А уже к зиме Энди Гутманс и Зив Сураски начали переработку ядра PHP. В их задачи входило увеличение производительности работы сложных приложений и улучшение модульности кода, лежащего в основе PHP.

Новое ядро было названо «Zend Engine» (от имен создателей: Zeev и Andi) и впервые представлено в середине 1999 года. PHP 4.0, основанный на этом ядре и принесший с собой набор дополнительных функций, официально вышел в мае 2000 года, почти через два года после своего предшественника, PHP 3.0. Помимо улучшения производительности, PHP 4.0 имел еще несколько ключевых нововведений, таких как поддержка сессий, буферизация вывода, более безопасные способы обработки вводимой пользователем информации и несколько новых языковых конструкций.

В настоящее время ведутся работы по улучшению Zend Engine и внедрению нововведений в PHP 5.0, первые бета-версии которого уже вышли в свет. Одно из существенных изменений произошло в объектной модели языка, ее основательно подлатали и добавили много новых возможностей.

Сегодня PHP используется сотнями тысяч разработчиков. Несколько миллионов сайтов написаны на PHP, что составляет более 20% доменов Internet.

Возможности PHP

«PHP может все», – заявляют его создатели. В первую очередь PHP используется для создания скриптов, работающих на стороне сервера, для этого его, собственно, и придумали. PHP способен решать те же задачи, что и любые другие CGI-скрипты, в том числе обрабатывать данные html-форм, динамически генерировать html страницы и т.п. Но есть и другие области, где может использоваться PHP. Всего выделяют три основные области применения PHP.

- Первая область, как уже говорилось, – это создание приложений (скриптов), которые исполняются на стороне сервера. PHP наиболее широко используется именно для создания такого рода скриптов. Для того чтобы работать таким образом, понадобится PHP-парсер (т.е. обработчик php-скриптов) и web-сервер для обработки скрипта, браузер для просмотра результатов работы скрипта, ну, и, конечно, какой-либо текстовый редактор для написания самого php-кода. Парсер PHP распространяется в виде CGI-программы или серверного модуля. Как установить его и web-сервер на свой компьютер, мы рассмотрим немного позднее. В этом курсе мы будем обсуждать, как правило, создание именно серверных приложений, как пример использования языка PHP.

- Вторая область – это создание скриптов, выполняющихся в командной строке. То есть с помощью PHP можно создавать такие скрипты, которые будут исполняться, вне зависимости от web-сервера и браузера, на конкретной машине. Для такой работы потребуется лишь парсер PHP (в этом случае его называют интерпретатором командной строки (cli, command line interpreter)). Этот способ работы подходит, например, для скриптов, которые должны выполняться регулярно с помощью различных планировщиков задач или для решения задач простой обработки текста.

- И последняя область – это создание GUI-приложений (графических интерфейсов), выполняющихся на стороне клиента. В принципе это не самый лучший способ использовать PHP, особенно для начинающих, но если вы уже досконально изучили PHP, то такие возможности языка могут оказаться весьма полезны. Для применения PHP в этой области потребуется специальный инструмент – PHP-GTK, который является расширением PHP.

Итак, область применения PHP достаточно обширна и разнообразна. Тем не менее существует множество других языков программирования, способных решать похожие задачи. Почему стоит изучать PHP? Что это нам дает? Во-первых, PHP очень прост в изучении. Достаточно ознакомиться лишь с основными правилами синтаксиса и принципами его работы, и можно начинать писать собственные программы, причем браться за такие задачи, решение которых на другом языке требовало бы серьезной подготовки.

Во-вторых, PHP поддерживается почти на всех известных платформах, почти во всех операционных системах и на самых разных серверах. Это тоже очень важно. Вряд ли кому-то захочется переходить, например, от работы под Windows к работе под Linux или от сервера IIS к серверу Apache только для того, чтобы изучить еще один язык программирования.

В PHP сочетаются две самые популярные парадигмы программирования – объектная и процедурная. В PHP4 более полно поддерживается процедурное программирование, но есть возможность писать программы и в объектном стиле. Уже в первых пробных версиях PHP5 большинство недочетов в реализации объектно-ориентированной модели языка, существующих в PHP4, устранены. Таким образом, можно выбрать наиболее привычный стиль работы.

Если говорить о возможностях сегодняшнего PHP, то они выходят далеко за рамки тех, что были реализованы в его первых версиях. С помощью PHP можно создавать изображения, PDF-файлы, флэш-ролики, в него включена поддержка большого числа современных баз данных, встроены функции для работы с текстовыми данными любых форматов, включая XML, и функции для работы с файловой системой. PHP поддерживает взаимодействие с различными сервисами посредством соответствующих протоколов, таких как протокол управления доступом к директориям LDAP, протокол работы с сетевым оборудованием SNMP, протоколы передачи сообщений IMAP, NNTP и POP3, протокол передачи гипертекста HTTP и т.д.

Обращая внимание на взаимодействие между различными языками, следует упомянуть о поддержке объектов Java и возможности их использования в качестве объектов PHP. Для доступа к удаленным объектам можно использовать расширение CORBA.

Для работы с текстовой информацией PHP унаследовал (с небольшими изменениями) механизмы работы с регулярными выражениями из языка Perl и UNIX-систем. Для обработки XML-документов можно использовать как стандарты DOM и SAX, так и API для XSLT-трансформаций.

Для создания приложений электронной коммерции существует ряд полезных функций, таких как функции осуществления платежей Cybercash, CyberMUT, VeriSign Payflow Pro и CCVS.

Лекция 30
Основы языка программного средства

В настоящее время количество приложений с использованием систем управления реляционными базами данных (СУБД) неуклонно растет. Особенно на этом фоне выделяются разработки под Интернет.

Современные информационные системы, такие как *динамические WEB-сайты,* используют СУБД для управления контентом (информационным наполнением) и обеспечения интерфейса взаимодействия с пользователями. Динамические Web-сайты, как правило, основаны на шаблонных страницах (в частности, HTML-формата), в которые вставляется (когда пользователь запрашивает соответствующие страницы через WEB-браузер) постоянно меняющееся информационное наполнение, извлекаемое из СУБД.

Отметим, что HTML (от англ. HyperText Markup Language — «язык разметки гипертекста») — стандартный язык разметки документов во Всемирной паутине. Большинство WEB-страниц создаются при помощи языка HTML (или XHTML).

Динамические WEB-сайты, как правило, создаются с использованием различных скриптовых языков программирования и технологий, среди которых, наиболее распространенными являются:

- **PHP** (англ. PHP: Hypertext Preprocessor — «PHP: препроцессор гипертекста») — скриптовый язык программирования общего назначения, интенсивно применяющийся для разработки WEB-приложений, в том числе взаимодействующих с СУБД;
- **ASP.NET** — технология создания WEB-приложений и WEB-сервисов от компании Microsoft. Она является составной частью платформы Microsoft .NET. . Разработчики могут писать код для ASP.NET, используя практически любые языки программирования, в том числе, и входящие в комплект .NET Framework (C#, Visual Basic.NET, и JScript .NET). ASP.NET имеет преимущество в скорости по сравнению со скриптовыми технологиями, так как при первом обращении код компилируется и

помещается в специальный кэш, и впоследствии только исполняется, не требуя затрат времени на парсинг, оптимизацию, и т. д.

Существуют и другие языки программирования и технологии создания динамических WEB-приложений, взаимодействующих с СУБД, например, язык программирования Java Server Pages (JSP), технология Java 2 Enterprise Edition и JDBC, технология создания расширений ISAPI и приложений CGI на языке программирования C++ и др. Такие технологии имеют определенные достоинства и недостатки, связанные с быстродействием, функциональными возможностями, и др. Вместе с тем, в процессе обучения программированию приложений баз данных для WEB целесообразно ориентироваться на наиболее популярные инструменты, такие как PHP, и ASP.NET (VBScript .NET). При этом следует отметить, что скриптовый язык программирования PHP является более простым в использовании и ориентирован на проектирование малых и средних динамических WEB-сайтов (таких как, форумы, блоги, интернет-магазины и др.), а технология ASP.NET предназначена для проектирования больших информационных систем (например, Интернет-порталов) и предполагает активное использование методов объектно-ориентированного программирования (ООП) и визуальных средств разработки, в частности, Microsoft Visual Studio. Также отметим, что технология PHP наилучшим образом адаптирована для работы с СУБД **MySQL,** а ASP.NET для взаимодействия с СУБД **Microsoft SQL Server** (посредством специальной технологии ADO.NET) .

Как было отмечено ранее, PHP (англ. PHP: Hypertext Preprocessor — «PHP: препроцессор гипертекста») — скриптовый язык программирования общего назначения, интенсивно применяющийся для разработки WEB-приложений, в том числе взаимодействующих с СУБД.

В отличие, в частности, от сценариев JavaScript и VBScript, выполняемых на стороне «клиента», PHP сценарии выполняются на стороне «сервера» и позволяют работать с различными СУБД, например MS SQL Server, Oracle, серверной файловой системой, почтовыми сервисами и др.

Обратите внимание. Для работы с PHP (ASP, JSP и др.) (в отличие от JavaScript) нужен установленный и специально настроенный WEB-сервер, например, Internet Information Server (IIS). К примеру, если у Вас установлена операционная система Windows XP/Vista и т.п., то Вы можете скачать дистрибутив PHP с сайта: www.php.net (например, для пятой версии: http://ru.php.net/get/php-5.2.12-Win32.zip/from/a/mirror).

Обратите внимание. PHP является свободно распространяемым продуктом, класса Open Source (открытого исходного кода).

Отметим, что файл, использующий PHP-сценарии, должен, как правило, иметь расширение «*.php».

ASP.NET — технология создания WEB-приложений и WEB-сервисов от компании Microsoft. Она является составной частью платформы Microsoft .NET и развитием более старой технологии Microsoft ASP. На данный момент последней версией этой технологии является **ASP.NET 4.0b**.

ASP.NET внешне во многом сохраняет схожесть с более старой технологией ASP, что позволяет разработчикам относительно легко перейти на ASP.NET. В то же время внутреннее устройство ASP.NET существенно отличается от ASP, поскольку она основана на платформе .NET и, следовательно, использует все новые возможности, предоставляемые этой платформой.

Хотя ASP.NET берёт своё название от старой технологии Microsoft ASP, она значительно от неё отличается. Microsoft полностью перестроила ASP.NET, основываясь на Common Language Runtime (CLR), который является основой всех приложений Microsoft .NET. Разработчики могут писать код для ASP.NET, используя практически любые языки программирования, в том числе, и входящие в комплект .NET Framework (C#, Visual Basic.NET, и JScript .NET). ASP.NET имеет преимущество в скорости по сравнению со скриптовыми технологиями, так как при первом обращении код компилируется и помещается в специальный кэш, и впоследствии только исполняется, не требуя затрат времени на парсинг, оптимизацию, и т. д.

Основа всего в Web-приложении – это страница. Пользователь, пользуясь браузером, перемещается между страницами, периодически возвращаясь к уже просмотренным ранее страницам, вводя какие-то данные в HTML формы и получая некоторый результат. В ASP.NET страница чаще всего представляет собой Web-форму, содержащую различные элементы управления, реагирующую на события, создаваемые пользователем.

ASP.NET 1.x позволяет разделять код логики от кода представления, то есть помещать код программной логики страницы в файл .cs или .vb, отдельно от кода собственно страницы, размещаемом в .aspx файле. Эта технология называется Code-Behind. Таким образом, дизайн страницы может быть изменен не затрагивая кода страницы, что позволяет разделить ответственность за внешний вид и работу страницы между дизайнером и программистом. Для этого в .aspx файле можно задать параметры директивы Page.

<%@ Page Language="c#" Src="User.aspx.cs" %>

Но для поддержки редактирования с помощью Microsoft Visual Studio .NET в ASP.NET странице необходимо указать класс, соответствующей данной странице и файл, в котором находится код этого класса. Для этого директива Page преобразуется с использованием ключевых слов Codebenind и Inherits.

<%@ Page Language="c#" Codebehind="TheProject.User" Inherits="User.aspx.cs" %>

В ASP.NET 2.0 используется иной механизм разделения кода. В директиве Page при этом необходимо использовать другие ключевые слова: CodeFile и Inherits.

<%@ Page Language="c#" CodeFile="TheProject.User" Inherits="User.aspx.cs" %>

В этом случае код класса программной логики страницы будет размещен в файле указанном в атрибуте CodeFile. Надо отметить, что Visual Studio использует разделяемые классы:

public partial class Default : System.Web.UI.Page{ protected void Page_Load(object sender, EventArgs e) { }}

Поэтому разработчик может поместить код класса в нескольких файлах, но подобное рассредоточение кода делает приложение весьма громоздким и трудным в поддержке и разработке.

Используемая в **Visual Studio** модель Code-Behind обладает несколькими весьма существенными недостатками. Прежде всего, используя Visual Studio разработчику необходимо компилировать проект перед публикацией, поскольку ASP.NET компилирует страницы, только если указан атрибут Src, не используемый Visual Studio. При этом, поскольку среда ASP.NET обнаруживает изменение даты создания сборки, после каждой замены старой сборки в каталоге bin происходит перезапуск домена приложения, что выливается во временную «заторможенность» в работе приложения.

Visual Studio использует новые ключевые слова, поддерживаемые средой выполнения ASP.NET 2.0, а среда выполнения, в свою очередь, использует новую технику компиляции страниц. Это позволяет решить проблему замены сборки на более новую.

Несмотря на это, Visual Studio по-прежнему позволяет отказаться от разделения кода и поместить код программной логики в самом файле страницы, и использовать теги <script runat="server"></script>. Более того, по умолчанию Visual Studio создает именно страницы без разделения кода.

В ASP.NET 2.0 среда выполнения также анализирует директивы Page, осуществляет поиск сборки соответствующей классу логики страницы, после чего создается класс страницы. В отличие от ASP.NET 1.x, родительским классом для класса страницы является System.Web.UI.Page, поскольку создаваемый динамический класс является собственно классом страницы (используются разделяемые классы для класса страницы и класса программной логики), а не потомком класса программной логики. Поэтому, если в ASP.NET 1.x класс Web-формы мог называться также как и сама Web-форма.

<form id="frmDefault" runat="server"></form>…public class frmDefault : System.Web.UI.Page{ protected void Page_Load(object sender, EventArgs e) { }}

В ASP.NET 2.0 это недопустимо, поскольку элемент управления System.Web.UI.Form является элементом класса. Основным преимуществом ASP.NET является то, что в случае отсутствия сборки, необходимой для выполнения страницы, происходит компиляция только файла программной логики страницы, без перекомпиляции всей сборки.Поскольку существует динамическая компиляция, то необходимо обеспечить возможность создавать код, общий для всех страниц приложения. Для этой цели в ASP.NET 2.0 существуют специальные директории, являющиеся дочерними директориями корневой директории приложения, одна из которых App_Code, служит для хранения файлов, содержащих общий код для всех страниц. При выполнении динамической компиляции, код из директории App_Code компилируется и становится доступным для всех страниц Web-приложения. При этом Visual Studio поддерживает код, находящийся в директории App_Code, поэтому работает подсветка синтаксиса и IntelliSense.

Обратите внимание. Технология ASP.NET базируется на технологии ASP (хотя и меет существенные отличия от последней).

Лекция 31
Синтаксис языка программного средства

Синтаксис PHP подобен синтаксису языка Си. Некоторые элементы, такие как ассоциативные массивы и циклforeach, заимствованы из Perl.

Для работы программы не требуется описывать какие-либо переменные, используемые модули и т. п. Любая программа может начинаться непосредственно с оператора PHP.

Простейшая программа Hello world на PHP выглядит следующим образом:

```
<?php
  echo 'Hello, world!';
?>
```

Также возможен более короткий вариант вывода строки:

```
<?= 'Hello, world!' ?>
```

Открывающий тег вида <?= используется для сокращённой записи конструкций используемых для вывода строки.

PHP исполняет код, находящийся внутри ограничителей, таких как <?php ?>. Всё, что находится вне ограничителей, выводится без изменений. В основном это используется для вставки PHP-кода в HTML-документ, например, так:

```
<html>
  <head>
    <title>
      Тестируем PHP
    </title>
  </head>
  <body>
    <?php
      echo 'Hello, world!';
    ?>
  </body>
</html>
```

Помимо ограничителей <?php ?>, допускается использование дополнительных вариантов, таких как <? ?> и <script language="php"> </script>. Кроме того, до версии 6.0 допускается использование ограничителей языка программирования ASP <% %> (конструкции <? ?> и <% %> могут быть выключены в конфигурационном файле php.ini).

Имена переменных начинаются с символа $, тип переменной объявлять не нужно. Имена переменных, функций и классов чувствительны к регистру. Константы также чувствительны к регистру. Переменные обрабатываются в строках, заключённых в

двойные кавычки, и heredoc-строках (строках, созданных при помощи оператора <<<). Переменные в строках, заключенных в одинарные кавычки, не обрабатываются.

PHP рассматривает переход на новую строку как пробел, так же как HTML и другие языки со свободным форматом. Инструкции разделяются с помощью*точки с запятой* (;), за исключением некоторых случаев, после объявления конструкции if/else и циклов.

Переменные в функцию можно передавать как по значению, так и по ссылке (используется знак &).

PHP поддерживает три типа комментариев: в стиле языка Си (ограниченные /* */), C++ (начинающиеся с // и идущие до конца строки) и оболочки UNIX (с # до конца строки).

Типы данных

PHP является языком программирования с динамической типизацией, не требующим указания типа при объявлении переменных, равно как и самого объявления переменных. Преобразования между скалярными типами зачастую осуществляются неявно без дополнительных усилий (впрочем, PHP предоставляет широкие возможности и для явного преобразования типов).

К скалярным типам данных относятся:

- целый тип (integer),
- вещественный тип данных (float, double),
- логический тип (boolean),
- строковый тип (string),
- и специальный тип NULL.

К нескалярным типам относятся:

- «ресурс» (resource),
- массив (array),
- объект (object),

К псевдотипам относятся:

- mixed один или несколько необязательных параметров,
- number число (integer либо float)
- callback (string или анонимная функция)
- void отсутствие параметров

Диапазон целых чисел (integer) в PHP зависит от платформы (обычно, это диапазон 32-битных знаковых целых чисел, то есть, от −2 147 483 648 до 2 147 483 647). Числа можно задавать в десятичной, восьмеричной и шестнадцатеричной системах счисления. Диапазон вещественных чисел (double) также зависит от платформы (для 32-битной архитектуры диапазон позволяет оперировать числами от $\pm 1.7\times10^{-308}$ до $\pm 1.7\times10^{+308}$).

PHP предоставляет разработчикам логический тип (boolean), способный принимать только два значения TRUE («истина») и FALSE («ложь»). При преобразовании в логический тип число 0, пустая строка, ноль в строке «0», NULL и пустой массив считаются равными FALSE. Все остальные значения автоматически преобразуются в TRUE.

Специальный тип NULL предназначен для переменных без определённого значения. Единственным значением данного типа является константа NULL. Тип NULL принимают неинициализированные переменные, переменные инициализированные константой NULL, а также переменные, удалённые при помощи конструкции unset().

Ссылки на внешние ресурсы имеют тип «ресурс» (resource). Переменные данного типа, как правило, представляют собой дескриптор, позволяющий управлять внешними объектами, такими как файлы, динамические изображения, результирующие таблицы базы данных и т. п.

Массивы (array) поддерживают числовые и строковые ключи и являются гетерогенными. Массивы могут содержать значения любых типов, включая другие массивы. Порядок элементов и их ключей сохраняется. Не совсем корректно называть php-массивы массивами, на самом деле это, скорее всего, упорядоченный хеш. Возможно неожиданное поведение при использовании цикла for со счетчиком

вместо foreach. Так, например, при сортировке массива с численными индексами функциями из стандартной библиотеки, сортируются и ключи тоже.

Указатель на функцию в PHP может быть представлен замыканием или псевдотипом callback. Замыкание доступно с версии 5.3 и в коде выглядит как простое определение функции, в которую явно можно утянуть значения из контекста, например:

function($args..$argsN) **use**($ctxVar,$ctxVar1) { definition ; }

callback тип может быть представлен:

- строкой (интерпретируется как название функции);
- массивом где нулевой и первый элемент — строки (интерпретируется как название статической функции класса);
- массивом где нулевой элемент — объект, а первый — строка (интерпретируется как метод у объекта).

Для проверки является ли значение вызываемым следует использовать is_callable($var)

Обращение к переменным и функциям[править | править вики-текст]

Обращение к переменным осуществляется с помощью символа $, за которым следует имя переменной. Данная конструкция может быть применена также для создания динамических переменных и функций.[16] Например:

```
$a = 'I am a'; // Запись значения в переменную $a
echo $a; // Вывод переменной $a

$b = 'a';
echo $$b; // Вывод переменной $a (дополнительный $ перед переменной $b)

echo ${'a'}; // Вывод переменной $a

function_name(); // Вызов функции function_name
$c = 'function_name';
$c(); // Вызов функции function_name

$d = 'Class_name';
$obj = new Class_name; // Создание объекта класса Class_name
$obj = new $d(); // Создание объекта класса Class_name

$obj->b; // Обращение к полю b объекта
$obj->c(); // Вызов метода c() объекта

$obj->$b; // Обращение к полю a объекта, так как $b = 'a'
$obj->$c(); // Вызов метода function_name() объекта, так как $c = 'function_name'
```

В PHP echo и print не являются функциями[17] (хотя print имеет возвращаемое значение), а являются синтаксическими единицами. При их использовании можно опустить скобки.

Суперглобальные массивы

Суперглобальными массивами (англ. *Superglobal arrays*) в PHP называются предопределённые массивы, имеющие глобальную область видимости без использования директивы global. Большая часть этих массивов содержит входные данные запроса пользователя (параметры GET-запроса, поля форм при посылке методом POST, куки и т. п.).

Все суперглобальные массивы, кроме $GLOBALS и $_REQUEST, имеют устаревшие аналоги с длинными именами, которые доступны вплоть до версии 5.3.x (начиная с 5.4.0 были удалены). Таким образом, обращения $_GET['year'] и $HTTP_GET_VARS['year'] идентичны (за исключением области видимости: массивы с «длинными» именами не являются суперглобальными).

$GLOBALS

Массив всех глобальных переменных (в том числе и пользовательских).

$_SERVER (аналог для устаревшего — $HTTP_SERVER_VARS)

Содержит переменные окружения, которые операционная система передаёт серверу.

$_ENV (уст. $HTTP_ENV_VARS)

Текущие переменные среды (англ. *Environment variables*). Их набор специфичен для платформы, на которой выполняется скрипт.

$_GET (уст. $HTTP_GET_VARS)

Содержит параметры GET-запроса, переданные в URI после знака вопроса «?».

$_POST (уст. $HTTP_POST_VARS)

Ассоциативный массив значений полей HTML-формы при отправке методом POST. Индексы элементов соответствуют значению свойства nameобъектов(кнопки, формы, радио-кнопки, флажки и т. д.) HTML-формы.

$_FILES (уст. $HTTP_POST_FILES)

Ассоциативный массив со сведениями об отправленных методом POST файлах. Каждый элемент имеет индекс, идентичный значению атрибута «name» в форме, и, в свою очередь, также является массивом со следующими элементами:

- ['name'] — исходное имя файла на компьютере пользователя.
- ['type'] — указанный агентом пользователя MIME-тип файла. PHP не проверяет его, и поэтому нет никаких гарантий, что указанный тип соответствует действительности.
- ['size'] — размер файла в байтах.
- ['tmp_name'] — полный путь к файлу во временной папке. Файл необходимо переместить оттуда функцией move_uploaded_file. Загруженные файлы из временной папки PHP удаляет самостоятельно.
- ['error'] — код ошибки. Если файл удачно загрузился, то этот элемент будет равен 0 (UPLOAD_ERR_OK).

$_COOKIE (уст. $HTTP_COOKIE_VARS)

Ассоциативный массив с переданными агентом пользователя значениями куки.

$_REQUEST

Содержит элементы из массивов $_GET, $_POST, $_COOKIE. С версии PHP 4.1 включает $_FILES.

$_SESSION (уст. $HTTP_SESSION_VARS)

Содержит данные сессии.

Лекция 32
Элементы и выражения языка.

Оператор if

Это один из самых важных операторов многих языков, включая PHP. Он позволяет выполнять фрагменты кода в зависимости от условия. Структуру оператора if можно представить следующим образом:

if (выражение) блок_выполнения

Здесь выражение есть любое правильное PHP-выражение (т.е. все, что имеет значение). В процессе обработки скрипта выражение преобразуется к логическому типу. Если в результате преобразования значение выражения истинно (True), то выполняется блок_выполнения. В противном случае блок_выполнения игнорируется. Если блок_выполнения содержит несколько команд, то он должен быть заключен в фигурные скобки { }.

Правила преобразования выражения к логическому типу:

- логическое False
- целый ноль (0)
- действительный ноль (0.0)

- пустая строка и строка "0"
- массив без элементов
- объект без переменных (подробно об объектах будет рассказано в одной из следующих лекций)
- специальный тип NULL
- Все остальные значения преобразуются в TRUE.

```
<?
$names = array("Иван","Петр","Семен");
if ($names[0]=="Иван") {
echo "Привет, Ваня!";
$num = 1;
$account = 2000;
}
if ($num) echo "Иван первый в списке!";
$bax = 30;
if ($account > 100*$bax+3)
echo "Эта строчка не появится
на экране, так как условие не выполнено";
?>
```

Оператор else

Мы рассмотрели только одну, основную часть оператора if. Существует несколько расширений этого оператора. Оператор else расширяет if на случай, если проверяемое в if выражение является неверным, и позволяет выполнить какие-либо действия при таких условиях.

Структуру оператора if, расширенного с помощью оператора else, можно представить следующим образом:

```
if (выражение) блок_выполнения
else блок_выполнения1
```

Эту конструкцию if...else можно интерпретировать примерно так: если выполнено условие (т.е. выражение=true), то выполняем действия из блока_выполнения, иначе – действия из блока_выполнения1. Использовать оператор else не обязательно.

Посмотрим, как можно изменить предыдущий пример, учитывая необходимость совершения действий и в случае невыполнения условия.

```
<?
$names = array("Иван","Петр","Семен");
if ($names[0]=="Иван") {
echo "Привет, Ваня!";
$num = 1;
$account = 2000;
} else {
echo "Привет, $names[0].
А мы ждали Ваню :(";
}
if ($num) echo "Иван первый в списке!";
else echo "Иван НЕ первый в списке?!";
$bax = 30;
if ($account > 100*$bax+3)
echo "Эта строка не появится на экране,
так как условие не выполнено";
else echo "Зато появится эта строка!";
?>
```

Оператор elseif

Еще один способ расширения условного оператора if – использование оператора elseif. elseif – это комбинация else и if. Как и else, он расширяет if для выполнения различных действий в том случае, если условие, проверяемое в if, неверно. Но в отличие от else, альтернативные действия будут выполнены, только если elseif-условие является верным. Структуру оператора if, расширенного с помощью операторов else и elseif, можно представить следующим образом:

```
if (выражение) блок_выполнения
elseif(выражение1) блок_выполнения1
...
else блок_выполненияN
```

Операторов elseif может быть сразу несколько в одном if-блоке. Elseif-утверждение будет выполнено, только если предшествующее if-условие является False, все предшествующие elseif-условия являются False, а данное elseif-условие – True.

```
<?
$names = array("Иван","Петр","Семен");
if ($names[0]=="Иван") {
// если первое имя в массиве Иван
echo "Привет, Ваня!";
}elseif ($names[0] == "Петр"){
// если первое имя
// не Иван, а Петр
echo "Привет, Петя!";
}elseif ($names[0] == "Семен"){
// если первое имя не
// Иван, не Петр, а Семен
echo "Привет, Сеня!";
}else {
// если первое имя не Иван,
// не Петр и не Семен
echo "Привет, $names[0]. А ты кто такой?";
}
?>
```

Альтернативный синтаксис

PHP предлагает альтернативный синтаксис для некоторых своих управляющих структур, а именно для if, while, for, foreach и switch. В каждом случае открывающую скобку нужно заменить на двоеточие (:), а закрывающую – на endif;, endwhile; и т.д. соответственно.

Например, синтаксис оператора if можно записать таким образом:

```
if(выражение): блок_выполнения endif;
```

Смысл остается тем же: если условие, записанное в круглых скобках оператора if, оказалось истиной, будет выполняться весь код, от двоеточия «:» до команды endif;. Использование такого синтаксиса полезно при встраивании php в html-код.

```
<?php
$names = array("Иван","Петр","Семен");
if ($names[0]=="Иван"):
?>
Привет, Ваня!
<?php endif ?>
```

Если используются конструкции else и elseif, то также можно задействовать альтернативный синтаксис:

```
<?php
if ($a == 5):
print "a равно 5";
```

```
print "...";
elseif ($a == 6):
print "a равно 6";
print "!!!";
else:
print "a не равно ни 5, ни 6";
endif;
?>
```

Оператор switch

Еще одна конструкция, позволяющая проверять условие и выполнять в зависимости от этого различные действия, – это switch. На русский язык название данного оператора можно перевести как «переключатель». И смысл у него именно такой. В зависимости от того, какое значение имеет переменная, он переключается между различными блоками действия. switch очень похож на оператор if...elseif...else или набор операторов if. Структуру switch можно записать следующим образом:

```
switch (выражение или переменная){
case значение1:
блок_действий1
break;
case значение2:
блок_действий2
break;
...
default:
блок_действий_по_умолчанию
}
```

В отличие от if, здесь значение выражения не приводится к логическому типу, а просто сравнивается со значениями, перечисленными после ключевых слов case (значение1, значение 2 и т.д.). Если значение выражения совпало с каким-то вариантом, то выполняется соответствующий блок_действий – от двоеточия после совпавшего значения до конца switch или до первого оператора break, если таковой найдется. Если значение выражения не совпало ни с одним из вариантов, то выполняются действия по умолчанию (блок_действий_по_умолчанию), находящиеся после ключевого слова default. Выражение в switch вычисляется только один раз, а в операторе elseif – каждый раз, поэтому, если выражение достаточно сложное, то switch работает быстрее.

Для конструкции switch, как и для if, возможен альтернативный синтаксис, где открывающая switch фигурная скобка заменяется двоеточием, а закрывающая – endswitch; соответственно.

Циклы

В PHP существует несколько конструкций, позволяющих выполнять повторяющиеся действия в зависимости от условия. Это циклы while, do..while, foreach и for. Рассмотрим их более подробно.

while

Структура:

```
while (выражение) { блок_выполнения }
```

либо

```
while (выражение): блок_выполнения endwhile;
```

while – простой цикл. Он предписывает PHP выполнять команды блока_выполнения до тех пор, пока выражение вычисляется как True (здесь, как и в if, происходит приведение выражения к логическому типу). Значение выражения проверяется каждый раз в начале цикла, так что, даже если его значение изменилось в процессе выполнения блока_выполнения, цикл не будет остановлен до конца итерации (т.е. пока все команды блока_выполнения не будут исполнены).

```
<?
//эта программа напечатает все четные цифры
$i = 1;
while ($i < 10) {
if ($i % 2 == 0) print $i;
// печатаем цифру, если она четная
$i++;
// и увеличиваем $i на единицу
}
?>
```

do... while

Циклы do..while очень похожи на циклы while, с той лишь разницей, что истинность выражения проверяется в конце цикла, а не в начале. Благодаря этому блок_выполнения цикл do...while гарантированно выполняется хотя бы один раз.

Операторы передачи управления

Иногда в случае особых обстоятельств требуется немедленно завершить работу цикла и передать управление первой инструкции программы, расположенной за последней инструкцией цикла. Для этого используют операторы break и continue.

Break

Оператор break заканчивает выполнение текущего цикла, будь то for, foreach, while, do..while или switch. break может использоваться с числовым аргументом, который говорит, работу скольких управляющих структур, содержащих его, нужно завершить.

```
<?php
$i=1;
while ($i) {
$n = rand(1,10);
// генерируем произвольное число
// от 1 до 10
echo "$i:$n ";
// выводим номер итерации и
// сгенерированное число
if ($n==5) break;
/* Если было сгенерировано число 5,
то прекращаем работу цикла. В этом случае
все, что находится после этой строчки
внутри цикла, не будет выполнено */
echo "Цикл работает <br>";
$i++;
}
echo "<br>Число итераций цикла $i ";
?>
```

Результатом работы этого скрипта будет примерно следующее:

1:7 Цикл работает
2:2 Цикл работает
3:5
Число итераций цикла 3

Если после оператора break указать число, то прервется именно такое количество содержащих этот оператор циклов. В приведенном выше примере это неактуально, поскольку вложенных циклов нет.

Операторы включения

include

Оператор include позволяет включать код, содержащийся в указанном файле, и выполнять его столько раз, сколько программа встречает этот оператор. Включение может производиться любым из перечисленных способов:

```
include 'имя_файла';
include $file_name;
include ("имя_файла");
```

Заметим, что использование оператора include эквивалентно простой вставке содержательной части файла params.inc в код программы include.php. Может быть, тогда можно было в params.inc записать простой текст без всяких тегов, указывающих на то, что это php-код? Нельзя! Дело в том, что в момент вставки файла происходит переключение из режима обработки PHP в режим HTML. Поэтому код внутри включаемого файла, который нужно обработать как PHP-скрипт, должен быть заключен в соответствующие теги.

Лекция 33
Функции. Работа с формами. (PHP, ASP, Java- сервлеты и т.п.)

В этой лекции будут рассматриваться вопросы создания и использования функций в PHP. Говоря «функции», мы не имеем в виду все существующие в PHP функции, а лишь функции, определяемые пользователем. Мы рассмотрим способы задания таких функций, методы передачи аргументов, использование аргументов со значением по умолчанию и значения, возвращаемые функцией.

В качестве примера создадим web-интерфейс для генерации html-формы. То есть пользователь выбирает, не прибегая к программированию, какие элементы формы нужно создать, и характеристики этих элементов, а наша программа генерирует нужную форму.

Функции, определяемые пользователем

Для чего нужны функции? Чтобы ответить на этот вопрос, нужно понять, что вообще представляют собой функции. В программировании, как и в математике, функция есть отображение множества ее аргументов на множество ее значений. То есть функция для каждого набора значений аргумента возвращает какие-то значения, являющиеся результатом ее работы. Зачем нужны функции, попытаемся объяснить на примере. Классический пример функции в программировании – это функция, вычисляющая значение факториала числа. То есть мы задаем ей число, а она возвращает нам его факториал. При этом не нужно для каждого числа, факториал которого мы хотим получить, повторять один и тот же код – достаточно просто вызвать функцию с аргументом, равным этому числу.

Таким образом, когда мы осуществляем действия, в которых прослеживается зависимость от каких-то данных, и при этом, возможно, нам понадобится выполнять такие же действия, но с другими исходными данными, удобно использовать механизм функций – оформить блок действий в виде тела функции, а меняющиеся данные – в качестве ее параметров.

Посмотрим, как в общем виде выглядит задание (объявление) функции. Функция может быть определена с помощью следующего синтаксиса:

```
function Имя_функции (параметр1, параметр2,
... параметрN){
Блок_действий
return "значение возвращаемое функцией";
}
```

Если прямо так написать в php-программе, то работать ничего не будет.

Во-первых, Имя_функции и имена параметров функции (параметр1, параметр2 и т.д.) должны соответствовать правилам наименования в PHP (и русских символов в них лучше не использовать). Имена функций нечувствительны к регистру.

Во-вторых, параметры функции – это переменные языка, поэтому перед названием каждого из них должен стоять знак $. Никаких троеточий ставить в списке параметров нельзя.

В-третьих, вместо слов блок_действий в теле функции должен находиться любой правильный PHP-код (не обязательно зависящий от параметров). И наконец, после ключевого слова return должно идти корректное php-выражение (что-либо, что имеет значение). Кроме того, у функции может и не быть параметров, как и возвращаемого значения. Пример правильного объявления функции – функция вычисления факториала, приведенная выше.

Как происходит вызов функции? Указывается имя функции и в круглых скобках список значений ее параметров, если таковые имеются:

```
<?php
Имя_функции ("значение_для_параметра1",
"значение_для_параметра2",...);
// пример вызова функции – вызов функции
// вычисления факториала приведен выше,
// там для вычисления факториала числа 3
// мы писали: fact(3);
// где fact – имя вызываемой функции,
// а 3 – значение ее параметра с именем $n
?>
```

Когда можно вызывать функцию? Казалось бы, странный вопрос. Функцию можно вызвать после ее определения, т.е. в любой строке программы ниже блока function f_name(){...}. В PHP3 это было действительно так. Но уже в PHP4 такого требования нет. Все дело в том, как интерпретатор обрабатывает получаемый код. Единственное исключение составляют функции, определяемые условно (внутри условных операторов или других функций). Когда функция определяется таким образом, ее определение должно предшествовать ее вызову.

Если функция однажды определена в программе, то переопределить или удалить ее позже нельзя. Несмотря на то, что имена функций нечувствительны к регистру, лучше вызывать функцию по тому же имени, каким она была задана в определении.

Аргументы функций

У каждой функции может быть, как мы уже говорили, список аргументов. С помощью этих аргументов в функцию передается различная информация (например, значение числа, факториал которого надо подсчитать). Каждый аргумент представляет собой переменную или константу.

С помощью аргументов данные в функцию можно передавать тремя различными способами. Это передача аргументов по значению (используется по умолчанию), по ссылке и задание значения аргументов по умолчанию. Рассмотрим эти способы подробнее.

Когда аргумент передается в функцию по значению, изменение значения аргумента внутри функции не влияет на его значение вне функции. Чтобы позволить функции изменять ее аргументы, их нужно передавать по ссылке. Для этого в определении функции перед именем аргумента следует написать знак амперсанд «&».

```
<?php
// напишем функцию, которая бы добавляла
// к строке слово checked
function add_label(&$data_str){
$data_str .= "checked";
}
$str = "<input type=radio name=article ";
// пусть имеется такая строка
echo $str .="><br>";
// выведет элемент формы –
```

```
// не отмеченную радио кнопку
add_label($str);
// вызовем функцию
echo $str ."><br>";
// это выведет уже отмеченную
// радио кнопку
?>
```

В функции можно определять значения аргументов, используемые по умолчанию. Само значение по умолчанию должно быть константным выражением, а не переменной и не представителем класса или вызовом другой функции.

У нас есть функция, создающая информационное сообщение, подпись к которому меняется в зависимости от значения переданного ей параметра. Если значение параметра не задано, то используется подпись "Оргкомитет".

```
<?php
function Message($sign="Оргкомитет."){
// здесь параметр sign имеет по умолчанию значение "Оргкомитет"
echo "Следующее собрание состоится завтра.<br>";
echo "$sign<br>";
}
Message();
// вызываем функцию без параметра.
// В этом случае подпись – это Оргкомитет
Message("С уважением, Вася");
// В этом случае подпись
// будет "С уважением, Вася."
?>
```

Если у функции несколько параметров, то те аргументы, для которых задаются значения по умолчанию, должны быть записаны после всех остальных аргументов в определении функции. В противном случае появится ошибка, если эти аргументы будут опущены при вызове функции.

Например, мы хотим внести описание статьи в каталог. Пользователь должен ввести такие характеристики статьи, как ее название, автор и краткое описание. Если пользователь не вводит имя автора статьи, считаем, что это Иванов Иван.

Списки аргументов переменной длины

В PHP4 можно создавать функции с переменным числом аргументов. То есть мы создаем функцию, не зная заранее, со сколькими аргументами ее вызовут. Для написания такой функции никакого специального синтаксиса не требуется. Все делается с помощью встроенных функций func_num_args(), func_get_arg(), func_get_args().

Функция func_num_args() возвращает число аргументов, переданных в текущую функцию. Эта функция может использоваться только внутри определения пользовательской функции. Если она появится вне функции, то интерпретатор выдаст предупреждение.

Использование переменных внутри функции

Глобальные переменные

Чтобы использовать внутри функции переменные, заданные вне нее, эти переменные нужно объявить как глобальные. Для этого в теле функции следует перечислить их имена после ключевого слова global:

Когда переменная объявляется как глобальная, фактически создается ссылка на глобальную переменную. Поэтому такая запись эквивалентна следующей (массив GLOBALS содержит все переменные, глобальные относительно текущей области видимости):

Статические переменные

Чтобы использовать переменные только внутри функции, при этом сохраняя их значения и после выхода из функции, нужно объявить эти переменные как статические. Статические переменные видны только внутри функции и не теряют своего значения, если выполнение программы выходит за пределы функции. Объявление таких переменных производится с помощью ключевого слова static:

Возвращаемые значения

Все функции, приведенные выше в качестве примеров, выполняли какие-либо действия. Кроме подобных действий, любая функция может возвращать как результат своей работы какое-нибудь значение. Это делается с помощью утверждения return. Возвращаемое значение может быть любого типа, включая списки и объекты. Когда интерпретатор встречает команду return в теле функции, он немедленно прекращает ее исполнение и переходит на ту строку, из которой была вызвана функция.

Когда функция возвращает несколько значений для их обработки в программе, удобно использовать языковую конструкцию list(), которая позволяет одним действием присвоить значения сразу нескольким переменным. Например, в предыдущем примере, оставив без изменения функцию, обработать возвращаемые ей значения можно было так:

Возвращение ссылки

В результате своей работы функция также может возвращать ссылку на какую-либо переменную. Это может пригодиться, если требуется использовать функцию для того, чтобы определить, какой переменной должна быть присвоена ссылка. Чтобы получить из функции ссылку, нужно при объявлении перед ее именем написать знак амперсанд (&) и каждый раз при вызове функции перед ее именем тоже писать амперсанд (&). Обычно функция возвращает ссылку на какую-либо глобальную переменную (или ее часть – ссылку на элемент глобального массива), ссылку на статическую переменную (или ее часть) или ссылку на один из аргументов, если он был также передан по ссылке.

Переменные функции

PHP поддерживает концепцию переменных функций. Это значит, что если имя переменной заканчивается круглыми скобками, то PHP ищет функцию с таким же именем и пытается ее выполнить.

В этом примере функция Show_text просто выводит строку текста. Казалось бы, зачем для этого создавать отдельную функцию, если существует специальная функция echo(). Дело в том, что такие функции, как echo(), print(), unset(), include() и т.п. нельзя использовать в качестве переменных функций.

Внутренние (встроенные) функции

Говоря о функциях, определяемых пользователем, все же нельзя не сказать пару слов о встроенных функциях. С некоторыми из встроенных функций, такими как echo(), print(), date(), include(), мы уже познакомились. На самом деле все перечисленные функции, кроме date(), являются языковыми конструкциями. Они входят в ядро PHP и не требуют никаких дополнительных настроек и модулей. Функция date() тоже входит в состав ядра PHP и не требует настроек. Но есть и функции, для работы с которыми нужно установить различные библиотеки и подключить соответствующий модуль. Например, для использования функций работы с базой данных MySql следует скомпилировать PHP с поддержкой этого расширения. В последнее время наиболее распространенные расширения и соответственно их функции изначально включают в состав PHP так, чтобы с ними можно работать без каких бы то ни было дополнительных настроек интерпретатора.

Решение задачи

Напомним, в чем состоит задача. Мы хотим написать интерфейс, который позволял бы создавать html-формы. Пользователь выбирает, какие элементы и в каком количестве

нужно создать, придумывает им названия, а наша программа сама генерирует требуемую форму.

Разобьем задачу на несколько подзадач: выбор типов элементов ввода и их количества, создание названий элементов ввода и обработка полученных данных, т.е. непосредственно генерация формы. Первая задача достаточно проста: нужно написать соответствующую форму, например подобную приведенной ниже (task_form.html):

```
<form action="ask_names.php">
Создать элемент "строка ввода текста": <input
type=checkbox name=types[]
value=string><br>
Количество элементов: <input type=text
name=numbers[string]
size=3><br>
<br>
Создать элемент "текстовая область": <input
type=checkbox
name=types[] value=text><br>
Количество элементов: <input type=text
name=numbers[text]
size=3><br>
<input type=submit value="Создать">
</form>
```

Когда мы пишем в имени элемента формы, например types[], это значит, что его имя – следующий элемент массива types. То есть у нас первый элемент формы ("строка ввода текста") будет иметь имя types[0], а второй (текстовая область) – types[1]. В браузере task_form.html будет выглядеть примерно так:

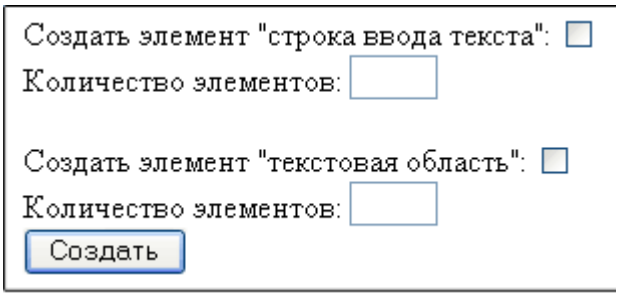


Рис. 1.1. Форма для выбора создаваемых элементов и их количества

После отправки данных этой формы мы получим информацию о том, какие элементы и сколько элементов каждого типа нужно создать. Следующий скрипт запрашивает названия для этих элементов:

Допустим, нужно создать два элемента типа «текстовая строка» и один элемент типа «текстовая область», как и отмечено в форме выше. Тогда скрипт ask_names.php обработает ее таким образом, что мы получим такую форму:

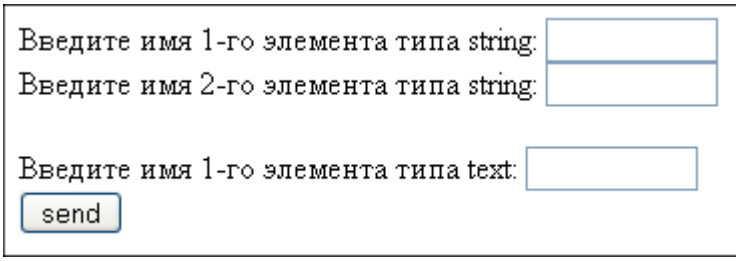


Рис. 1.2. Форма для ввода названий создаваемых элементов

Введем в эту форму, например, строки «Название», «Автор» и «Краткое содержание». Эти данные будет обрабатывать скрипт task.php.

Результатом работы этого скрипта с входными данными, приведенными выше, будет следующая форма:

Рис. 1.3. Пример формы, сгенерированной нашей программой

Лекция 34
Работа с базами данных.

База данных – это совокупность связанных данных, организованных по определенным правилам, предусматривающим общие принципы описания, хранения и манипулирования, независимая от прикладных программ. База данных является информационной моделью предметной области. Обращение к базам данных осуществляется с помощью системы управления базами данных (**СУБД**). СУБД обеспечивает поддержку создания баз данных, централизованного управления и организации доступа к ним различных пользователей.

Итак, мы пришли к выводу, что хранить данные независимо от программ, так, что они связаны между собой и организованы по определенным правилам, целесообразно. Но вопрос, как хранить данные, по каким правилам они должны быть организованы, остался открытым. Способов существует множество (кстати, называются они моделями представления или хранения данных). Наиболее популярные – объектная и реляционная модели данных.

Автором **реляционной модели** считается Э. Кодд, который первым предложил использовать для обработки данных аппарат теории множеств (объединение, пересечение, разность, декартово произведение) и показал, что любое представление данных сводится к совокупности двумерных таблиц особого вида, известного в математике как отношение.

Таким образом, реляционная база данных представляет собой набор таблиц (точно таких же, как приведенная выше), связанных между собой. Строка в таблице соответствует сущности реального мира (в приведенном выше примере это информация о человеке).

Примеры реляционных СУБД: MySql, PostgreSql.

В основу **объектной модели** положена концепция объектно-ориентированного программирования, в которой данные представляются в виде набора объектов и классов, связанных между собой родственными отношениями, а работа с объектами осуществляется с помощью скрытых (инкапсулированных) в них методов.

Примеры объектных СУБД: Cache, GemStone (от Servio Corporation), ONTOS (ONTOS).

В последнее время производители СУБД стремятся соединить два этих подхода и проповедуют объектно-реляционную модель представления данных. Примеры таких СУБД – IBM DB2 for Common Servers, Oracle8.

Поскольку мы собираемся работать с Mysql, то будем обсуждать аспекты работы только с реляционными базами данных. Нам осталось рассмотреть еще два важных понятия из этой области: ключи и индексирование, после чего мы сможем приступить к изучению языка запросов SQL.

Ключи

Для начала давайте подумаем над таким вопросом: какую информацию нужно дать о человеке, чтобы собеседник точно сказал, что это именно тот человек, сомнений быть не может, второго такого нет? Сообщить фамилию, очевидно, недостаточно, поскольку существуют однофамильцы. Если собеседник человек, то мы можем приблизительно объяснить, о ком речь, например вспомнить поступок, который совершил тот человек, или

еще как-то. Компьютер же такого объяснения не поймет, ему нужны четкие правила, как определить, о ком идет речь. В системах управления базами данных для решения такой задачи ввели понятие первичного ключа.

Первичный ключ (primary key, PK) – минимальный набор полей, уникально идентифицирующий запись в таблице. Значит, первичный ключ – это в первую очередь набор полей таблицы, во-вторых, каждый набор значений этих полей должен определять единственную запись (строку) в таблице и, в-третьих, этот набор полей должен быть минимальным из всех обладающих таким же свойством. Поскольку первичный ключ определяет только одну уникальную запись, то никакие две записи таблицы не могут иметь одинаковых значений первичного ключа.

Например, в нашей таблице (см. выше) ФИО и адрес позволяют однозначно выделить запись о человеке. Если же говорить в общем, без связи с решаемой задачей, то такие знания не позволяют точно указать на единственного человека, поскольку существуют однофамильцы, живущие в разных городах по одному адресу. Все дело в границах, которые мы сами себе задаем. Если считаем, что знания ФИО, телефона и адреса без указания города для наших целей достаточно, то все замечательно, тогда поля ФИО и адрес могут образовывать первичный ключ. В любом случае проблема создания первичного ключа ложится на плечи того, кто проектирует базу данных (разрабатывает структуру хранения данных). Решением этой проблемы может стать либо выделение характеристик, которые естественным образом определяют запись в таблице (задание так называемого логического, или естественного, PK), либо создание дополнительного поля, предназначенного именно для однозначной идентификации записей в таблице (задание так называемого суррогатного, или искусственного, PK). Примером логического первичного ключа является номер паспорта в базе данных о паспортных данных жителей или ФИО и адрес в телефонной книге (таблица выше). Для задания суррогатного первичного ключа в нашу таблицу можно добавить поле id (идентификатор), значением которого будет целое число, уникальное для каждой строки таблицы. Использование таких суррогатных ключей имеет смысл, если естественный первичный ключ представляет собой большой набор полей или его выделение нетривиально.

Кроме однозначной идентификации записи, первичные ключи используются для организации связей с другими таблицами.

Например, у нас есть три таблицы: содержащая информацию об исторических личностях (Persons), содержащая информацию об их изобретениях (Artifacts) и содержащая изображения как личностей, так и артефактов (Images) (рис 10.1).

Первичным ключом во всех этих таблицах является поле id (идентификатор). В таблице Artifacts есть поле author, в котором записан идентификатор, присвоенный автору изобретения в таблице Persons. Каждое значение этого поля является **внешним ключом**для первичного ключа таблицы Persons. Кроме того, в таблицах Persons и Artifacts есть поле photo, которое ссылается на изображение в таблице Images. Эти поля также являются внешними ключами для первичного ключа таблицы Images и устанавливают однозначную логическую связь Persons-Images и Artifacts-Images. То есть если значение внешнего ключа photo в таблице личности равно 10, то это значит, что фотография этой личности имеет id=10 в таблице изображений. Таким образом, **внешние ключи** используются для организации связей между таблицами базы данных (родительскими и дочерними) и для поддержания ограничений ссылочной целостности данных.

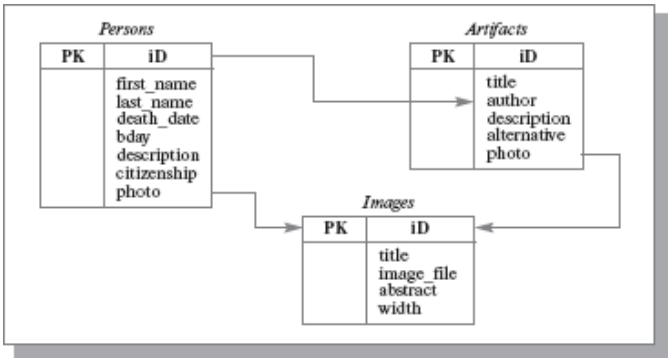


Рис. 1.1. Пример использования первичных ключей для организации связей с другими таблицами

Индексирование

Одна из основных задач, возникающих при работе с базами данных, – это задача поиска. При этом, поскольку информации в базе данных, как правило, содержится много, перед программистами встает задача не просто поиска, а эффективного поиска, т.е. поиска за сравнительно небольшое время и с достаточной точностью. Для этого (для оптимизации производительности запросов) производят **индексирование** некоторых полей таблицы. Использовать индексы полезно для быстрого поиска строк с указанным значением одного столбца. Без индекса чтение таблицы осуществляется по всей таблице, начиная с первой записи, пока не будут найдены соответствующие строки. Чем больше таблица, тем больше накладные расходы. Если же таблица содержит индекс по рассматриваемым столбцам, то база данных может быстро определить позицию для поиска в середине файла данных без просмотра всех данных. Это происходит потому, что база данных помещает проиндексированные поля поближе в памяти, так, чтобы можно было побыстрее найти их значения. Для таблицы, содержащей 1000 строк, это будет как минимум в 100 раз быстрее по сравнению с последовательным перебором всех записей. Однако в случае, когда необходим доступ почти ко всем 1000 строкам, быстрее будет последовательное чтение, так как при этом не требуется операций поиска по диску. Так что иногда индексы бывают только помехой. Например, если копируется большой объем данных в таблицу, то лучше не иметь никаких индексов. Однако в некоторых случаях требуется задействовать сразу несколько индексов (например, для обработки запросов к часто используемым таблицам).

Если говорить о MySQL, то там существует три вида индексов: PRIMARY, UNIQUE, и INDEX, а слово ключ (KEY) используется как синоним слова индекс (INDEX). Все индексы хранятся в памяти в виде B-деревьев.

PRIMARY – уникальный индекс (ключ) с ограничением, что все индексированные им поля не могут иметь пустого значения (т.е. они NOT NULL). Таблица может иметь только один первичный индекс, но он может состоять из нескольких полей.

UNIQUE – ключ (индекс), задающий поля, которые могут иметь только уникальные значения.

INDEX – обычный индекс (как мы описали выше). В MySqL, кроме того, можно индексировать строковые поля по заданному числу символов от начала строки.

СУБД MySQL

MySQL – это реляционная система управления базами данных. То есть данные в ее базах хранятся в виде логически связанных между собой таблиц, доступ к которым осуществляется с помощью языка запросов SQL. MySQL – свободно распространяемая система, т.е. платить за ее применение не нужно. Кроме того, это достаточно быстрая, надежная и, главное, простая в использовании СУБД, вполне подходящая для не слишком глобальных проектов.

Работать с MySQL можно не только в текстовом режиме, но и в графическом. Существует очень популярный визуальный интерфейс (кстати, написанный на PHP) для работы с этой СУБД. Называется он PhpMyAdmin. Этот интерфейс позволяет значительно упростить работу с базами данных в MySQL.

В текстовом режиме работа с базой данных выглядит просто как ввод команд в командную строку (рис 10.2), а результаты выборок возвращаются в виде своеобразных таблиц, поля в которых налезают друг на друга, если данные не помещаются на экран (рис 10.3).

```
mysql> show databases;
```

Рис. 1.2. Работа с MySQL в коммандной строке. Команда show databases — вывести все имеющиеся базы данных

PhpMyAdmin позволяет пользоваться всеми достоинствами браузера, включая прокрутку изображения, если оно не умещается на экран. Многие из базовых SQL-функций работы с данными в PhpMyAdmin сведены к интуитивно понятным интерфейсам и действиям, напоминающим переход по ссылкам в Internet. Но тем не менее стоит все же поработать и в текстовом режиме.

```
mysql> show databases;
+----------+
| Database |
+----------+
| book     |
| mysql    |
| test     |
+----------+
3 rows in set (0.01 sec)

mysql>
```

Рис. 1.3. Работа с MySQL в коммандной строке. Результат обработки команды show databases

Перед тем как переходить к детальному изучению языка SQL, несколько слов об установке MySQL и подготовке к работе. Если вы не собираетесь заниматься администрированием сервера, то информация, приведенная ниже, пригодится вам только для общего развития. Итак, устанавливается MySQL очень просто – автоматически, пару раз нажмите OK, и все. После этого вы можете зайти в директорию, где лежат файлы типа mysql.exe, mysqld.exe и т.п. (у нас под Windows XP это c:\mysql\bin) Последний файл запускает Mysql-сервер. В некоторых системах сервер запускается в виде сервиса. После запуска сервера следует запустить mysql-клиент, запустив программу mysql.exe.

Все данные о пользователях MySQL хранит в таблице user в специальной базе данных mysql, доступ к которой имеет только администратор сервера. Поэтому, чтобы изменить какой-либо пароль, нужно изменить эту таблицу. Пароль задается с помощью функции PASSWORD, которая кодирует введенные данные. Кроме изменения пароля администратора, нужно еще удалить всех пользователей, не имеющих логина (команда DELETE). Команда Flush Privileges заставляет вступить в действие изменения, произошедшие в системной базе данных (mysql).

екция 35

Взаимосвязь БД и серверных приложений

В дистрибутив PHP входит расширение, содержащее встроенные функции для работы с базой данных MySQL. В этой лекции мы познакомимся с некоторыми основными функциями для работы с MySQL, которые потребуются для решения задач построения web-интерфейсов с целью отображения и наполнения базы данных. Возникает вопрос, зачем строить такие интерфейсы? Для того чтобы вносить информацию в базу данных и просматривать ее содержимое могли люди, не знакомые с языком запросов SQL. При работе с web-интерфейсом для добавления информации в базу данных человеку нужно просто ввести эти данные в html-форму и отправить их на сервер, а наш скрипт сделает все остальное. А для просмотра содержимого таблиц достаточно просто щелкнуть по ссылке и зайти на нужную страницу.

Для наглядности будем строить эти интерфейсы для таблицы Artifacts, в которой содержится информация об экспонатах виртуального музея информатики. В предыдущей лекции мы уже приводили структуру этой коллекции, а также ее связи с коллекциями описания персон (Persons) и изображений (Images). Напомним, что каждый экспонат в коллекции Artifacts описывается с помощью следующих характеристик:

- название (title);
- автор (author);
- описание (description);
- альтернативное название (alternative);
- изображение (photo).

Название и альтернативное название являются строками менее чем 255 символов длиной (т.е. имеют тип VARCHAR(255)), описание - текстовое поле (имеет тип TEXT), а в полях "автор" и "изображение" содержатся идентификаторы автора из коллекции Persons и изображения экспоната из коллекции Images соответственно.

Построение интерфейса для добавления информации

Итак, у нас есть какая-то таблица в базе данных. Чтобы построить интерфейс для добавления информации в эту таблицу, нужно ее структуру (т.е. набор ее полей) отобразить в html-форму.

Разобьем эту задачу на следующие подзадачи:

- установка соединения с БД;
- выбор рабочей БД;
- получение списка полей таблицы;
- отображение полей в html-форму.

После этого данные, введенные в форму, нужно записать в базу данных. Рассмотрим все эти задачи по порядку.

Установка соединения

Итак, первое, что нужно сделать, - это установить соединение с базой данных. Воспользуемся функцией mysql_connect.

Данная функция устанавливает соединение с сервером MySQL и возвращает указатель на это соединение или FALSE в случае неудачи. Для отсутствующих параметров устанавливаются следующие значения по умолчанию:

server = 'localhost:3306'

username = имя пользователя владельца

процесса сервера

password = пустой пароль

Если функция вызывается дважды с одними и теми же параметрами, то новое соединение не устанавливается, а возвращается ссылка на старое соединение. Чтобы этого избежать, используют параметр new_link, который заставляет в любом случае открыть еще одно соединение.

Параметр client_flags - это комбинация следующих констант: MYSQL_CLIENT_COMPRESS (использовать протокол сжатия), MYSQL_CLIENT_IGNORE_SPACE (позволяет вставлять пробелы после имен функций), MYSQL_CLIENT_INTERACTIVE (ждать interactive_timeout секунд - вместо wait_timeout - до закрытия соединения).

Параметр new_link появился в PHP 4.2.0, а параметр client_flags - в PHP 4.3.0.

Соединение с сервером закрывается при завершении исполнения скрипта, если оно до этого не было закрыто с помощью функции mysql_close().

Итак, устанавливаем соединение с базой данных на локальном сервере для пользователя nina с паролем "123":

```
<?
$conn = mysql_connect(
"localhost", "nina","123")
or die("Невозможно установить
соединение: ". mysql_error());
echo "Соединение установлено";
mysql_close($conn);
?>
```

Действие mysql_connect равносильно команде

```
shell>mysql -u nina -p123
```

Выбор базы данных

После установки соединения нужно выбрать базу данных, с которой будем работать. Наши данные хранятся в базе данных book. В MySQL выбор базы данных осуществляется с помощью команды use:

mysql>use book;

В PHP для этого существует функция mysql_select_db.

Эта функция возвращает TRUE в случае успешного выбора базы данных и FALSE - в противном случае.

Сделаем базу данных book рабочей:

```
<?
$conn = mysql_connect(
"localhost","nina","123")
or die("Невозможно установить
соединение: ". mysql_error());
echo "Соединение установлено";
mysql_select_db("book");
?>
```

Получение списка полей таблицы

Теперь можно заняться собственно решением задачи. Как получить список полей таблицы? Очень просто. В PHP и на этот случай есть своя команда - mysql_list_fields.

Эта функция возвращает список полей в таблице table_name в базе данных database_name. Получается, что выбирать базу данных нам было необязательно, но это пригодится позже. Как можно заметить, результат работы этой функции - переменная типа ресурс. То есть это не совсем то, что мы хотели получить. Это ссылка, которую можно использовать для получения информации о полях таблицы, включая их названия, типы и флаги.

Функция mysql_field_name возвращает имя поля, полученного в результате выполнения запроса. Функция mysql_field_len возвращает длину поля. Функция mysql_field_type возвращает тип поля, а функция mysql_field_flags возвращает список флагов поля, записанных через пробел. Типы поля могут быть int, real, string, blob и т.д. Флаги могут быть not_null, primary_key, unique_key, blob, auto_increment и т.д.

Здесь result - это идентификатор результата запроса (например, запроса, отправленного функциями mysql_list_fields или mysql_query (о ней будет рассказано позднее)), a field_offset - порядковый номер поля в результате.

Вообще говоря, то, что возвращают функции типа mysql_list_fields или mysql_query, представляет собой таблицу, а точнее, указатель на нее. Чтобы получить из этой таблицы конкретные значения, нужно задействовать специальные функции, которые построчно читают эту таблицу. К таким функциям и относятся mysql_field_name и т.п. Чтобы перебрать все строки в таблице результата выполнения запроса, нужно знать число строк в этой таблице. Команда mysql_num_rows(ресурс result) возвращает число строк во множестве результатов result.

А теперь попробуем получить список полей таблицы Artifacts (коллекция экспонатов).

```
<?
$conn = mysql_connect(
"localhost","nina","123")
or die("Невозможно установить
соединение: ". mysql_error());
echo "Соединение установлено";
mysql_select_db("book");
$list_f = mysql_list_fields (
"book","Artifacts",$conn);
$n = mysql_num_fields($list_f);
for($i=0;$i<$n; $i++){
```

```
$type = mysql_field_type($list_f, $i);
$name_f = mysql_field_name($list_f,$i);
$len = mysql_field_len($list_f, $i);
$flags_str = mysql_field_flags (
$list_f, $i);
echo "<br>Имя поля: ". $name_f;
echo "<br>Тип поля: ". $type;
echo "<br>Длина поля: ". $len;
echo "<br>Строка флагов поля: ".
$flags_str . "<hr>";
}
?>
```

В результате должно получиться примерно вот что (если в таблице всего два поля, конечно):

Имя поля: id
Тип поля: int
Длина поля: 11
Строка флагов поля:
not_null primary_key auto_increment
Имя поля: title
Тип поля: string
Длина поля: 255
Строка флагов поля:

Отображение списка полей в html-форму

Теперь немножко подкорректируем предыдущий пример. Будем не просто выводить информацию о поле, а отображать его в подходящий элемент html-формы. Так, элементы типа BLOB переведем в textarea (заметим, что поле description, которое мы создавали с типом TEXT, отображается как имеющее тип BLOB), числа и строки отобразим в текстовые строки ввода <input type=text>, а элемент, имеющий метку автоинкремента, вообще не будем отображать, поскольку его значение устанавливается автоматически.

Все это решается довольно просто, за исключением выделения из списка флагов флага auto_increment. Для этого нужно воспользоваться функцией explode.

Синтаксис explode:

массив explode(строка separator,
строка string [, int limit])

Эта функция разбивает строку string на части с помощью разделителя separator и возвращает массив полученных строк.

В нашем случае в качестве разделителя нужно взять пробел " ", а в качестве исходной строки для разбиения - строку флагов поля.

Итак, создадим форму для ввода данных в таблицу Artifacts:

Запись данных в базу данных

Итак, форма создана. Теперь нужно сделать самое главное - отправить данные из этой формы в нашу базу данных. Как вы уже знаете, для того чтобы записать данные в таблицу, используется команда INSERT языка SQL. Например:

```
mysql> INSERT INTO Artifacts
SET title='Петров';
```

Возникает вопрос, как можно воспользоваться такой командой (или любой другой командой SQL) в PHP скрипте. Для этого существует функция mysql_query().

mysql_query() посылает SQL-запрос активной базе данных MySQL сервера, который определяется с помощью указателя link_identifier (это ссылка на какое-то соединение с сервером MySQL). Если параметр link_identifier опущен, используется последнее открытое соединение. Если открытые соединения отсутствуют, функция пытается соединиться с СУБД, аналогично функции mysql_connect() без параметров. Результат запроса буферизируется.

Замечание: строка запроса НЕ должна заканчиваться точкой с запятой.

Только для запросов SELECT, SHOW, EXPLAIN, DESCRIBE, mysql_query() возвращает указатель на результат запроса, или FALSE, если запрос не был выполнен. В остальных случаях mysql_query() возвращает TRUE, если запрос выполнен успешно, и FALSE - в случае ошибки. Значение, не равное FALSE, говорит о том, что запрос был выполнен успешно. Оно не говорит о количестве затронутых или возвращенных рядов. Вполне возможна ситуация, когда успешный запрос не затронет ни одного ряда. mysql_query() также считается ошибочным и вернет FALSE, если у пользователя недостаточно прав для работы с указанной в запросе таблицей.

Итак, теперь мы знаем, как отправить запрос на вставку строк в базу данных. Заметим, что в предыдущем примере элементы формы мы назвали именами полей таблицы. Поэтому они будут доступны в скрипте insert.php, обрабатывающем данные формы, как переменные вида

$_POST['имя_поля'].

Итак, задачу добавления данных с помощью web-интерфейса мы решили. Однако тут есть одна тонкость. При решении мы не учитывали тот факт, что значения некоторых полей (author, photo) должны браться из других таблиц (Persons, Images). Поскольку MySQL с внешними ключами не работает, этот момент остается на совести разработчиков системы, т.е. на нашей совести. Нужно дописать программу таким образом, чтобы была возможность вводить в такие поля правильные значения. Но мы делать этого не будем, поскольку задача лекции состоит в том, чтобы познакомить читателя с элементами технологии, а не в том, чтобы создать работающую систему. Кроме того, имеющихся у читателя знаний вполне достаточно, чтобы решить эту проблему самостоятельно. Мы же обратимся к другой задаче - отображение данных, хранящихся в базе данных СУБД MySQL.

Отображение данных, хранящихся в MySQL

Чтобы отобразить какие-то данные в браузер с помощью PHP, нужно сначала получить эти данные в виде переменных PHP. При работе с MySQL без посредника (такого, как PHP) выборка данных производится с помощью команды SELECT языка SQL:

mysql> SELECT * FROM Artifacts;

В предыдущей главе мы говорили, что любой запрос, в том числе и на выборку, можно отправить на сервер с помощью функции mysql_query(); Там у нас стояла немного другая задача - получить данные из формы и отправить их с помощью запроса на вставку в базу данных. Результатом работы mysql_query() там могло быть только одно из выражений, TRUE или FALSE. Теперь же требуется отправить запрос на выбор всех полей, а результат отобразить в браузере. И здесь результат - это целая таблица значений, а точнее, указатель на эту таблицу. Так что нужны какие-то аналоги функции mysql_field_name(), только чтобы они извлекали из результата запроса не имя, а значение поля. Таких функций в PHP несколько. Наиболее популярные - mysql_result() и mysql_fetch_array().

mysql_result() возвращает значение одной ячейки результата запроса. Аргумент field может быть порядковым номером поля в результате, именем поля или именем поля с именем таблицы через точку tablename.fieldname. Если для имени поля в запросе применялся алиас ('select foo as bar from...'), используйте его вместо реального имени поля.

Работая с большими результатами запросов, следует задействовать одну из функций, обрабатывающих сразу целый ряд результата (например, mysql_fetch_row(), mysql_fetch_array() и т.д.). Так как эти функции возвращают значение нескольких ячеек сразу, они НАМНОГО быстрее mysql_result(). Кроме того, нужно учесть, что указание численного смещения (номера поля) работает намного быстрее, чем указание колонки или колонки и таблицы через точку.

Вызовы функции mysql_result() не должны смешиваться с другими функциями, работающими с результатом запроса.

Эта функция обрабатывает ряд результата запроса, возвращая массив (ассоциативный, численный или оба) с обработанным рядом результата запроса, или FALSE, если рядов больше нет.

mysql_fetch_array() - это расширенная версия функции mysql_fetch_row(). Помимо хранения значений в массиве с численными индексами, функция возвращает значения в массиве с индексами по названию колонок.

Если несколько колонок в результате будут иметь одинаковые названия, будет возвращена последняя колонка. Чтобы получить доступ к первым, следует использовать численные индексы массива или алиасы в запросе. В случае алиасов именно их вы не сможете использовать настоящие имена колонок, как, например, не сможете использовать "photo" в описанном ниже примере.

```
select Artifacts.photo as art_image,
Persons.photo as pers_image
from Artifacts, Persons
```

Важно заметить, что mysql_fetch_array() работает НЕ медленнее, чем mysql_fetch_row(), и предоставляет более удобный доступ к данным.

Второй опциональный аргумент result_type в функции mysql_fetch_array() является константой и может принимать следующие значения: MYSQL_ASSOC, MYSQL_NUM и MYSQL_BOTH. Эта возможность добавлена в PHP 3.0.7. Значением по умолчанию является: MYSQL_BOTH.

Используя MYSQL_BOTH, получим массив, состоящий как из ассоциативных индексов, так и из численных. MYSQL_ASSOC вернет только ассоциативные соответствия, а MYSQL_NUM - только численные.

Замечание: имена полей, возвращаемые этой функцией, регистрозависимы.

Теперь отобразим данные из Artifacts в виде таблицы в браузере:

Сделаем то же самое с помощью mysql_fetch_array():

В этой лекции мы решили две задачи: добавление данных в базу данных и их отображение в браузере с помощью языка PHP. Для этого мы рассмотрели ряд функций, которые позволяют отправлять SQL-запросы к базе данных и обрабатывать полученные ответы. Используя приведенную здесь технологию, можно решить целый ряд похожих задач, таких как задачи изменения и удаления данных, задачи манипулирования таблицами базы данных (т.е. их создание, изменение и удаление) и т.п. Все это типовые задачи, возникающие при разработке систем управления данными, и умение их решать, как и умение работать с базами данных в целом, очень важно для web-программиста.

Лекция 36
Проектирование Web-служб.

При создании сайта весь процесс разработки разбивается на несколько этапов, выполнение каждого из которых соблюдается в строгой последовательности. Это проектирование, дизайн, html-верстка, программирование, наполнение материалами, тестирование и запуск. В этой статьи я подробно рассмотрю все этапы создания сайта.

1Проектирование

Этап проектирования сайта – это фундамент будущего проекта. От правильности его реализации будет зависеть успешность функционирования и продвижения сайта.

Для того, чтобы верно спроектировать сайт, вебмастер должен четко знать что из себя будет представлять будущий сайт, как он должен выглядеть, какие задачи должен выполнять, какой функционал должен присутствовать на сайте.

Проектирование делится на несколько последовательных шагов:

1. Заполнение брифинга
2. Составление технического задания (далее – ТЗ)
3. Заключение договора

Брифинг на создание сайта – это анкета с вопросами, на которые отвечает заказчик, чтобы разработчик смог понять объем предстоящих работ, оценить затраты по времени и стоимость проекта. Более подробно читайте в статье «Брифинг на создание сайта».

Техническое задание составляется либо самим заказчиком, если у него имеется такой опыт, либо вебмастером. Для крупных проектов составление ТЗ может

оплачиваться отдельно, т.к. это довольно сложный и длительный процесс. Необходимо полностью, до мелочей расписать весь функционал сайта. Чаще всего заказчик сам до конца не знает, что он хочет увидеть в итоге, поэтому разработчику приходиться все продумывать самостоятельно. После составления ТЗ оно утверждается заказчиком. Для простых сайтов составление и утверждение ТЗ не является обязательным. Более подробно читайте в статье «Техническое задание на создание сайта».

Заключение договора – это юридические гарантии как для заказчика, так и для исполнителя. В договоре указываются сроки, стоимость, порядок выполнения, сдачи, приема готовой работы и оплаты. Также указана ответственность сторон в случае нарушения условий договора и форс-мажорные обстоятельства. Хотя договор не является 100% гарантией защиты, но все же заставляет придерживаться правил. В идеале в составлении договора должны быть заинтересованы обе стороны, т.к. заказчик получает гарантии выполнения проекта в оговоренные сроки, а исполнитель получает гарантии оплаты, а также защищен от претензий со стороны заказчика на выполнение задач, которые не были указаны в ТЗ и договоре. Но на практике договор заключается не всегда, т.к. при выполнении небольших проектов стороны не хотят тратить время на бумажную волокиту.

Работы по проектированию обычно оценивают в 10% от стоимости сайта.

Составление техзадания занимает не более 1 дня для простых сайтов и может занимать до 3-х дней для сложных.

2 Дизайн

После подготовительных работ на этапе проектирования приступают к разработке **дизайна сайта**. Дизайн создается на основе ТЗ.

Следует обратить внимание на следующие моменты:

1. Выбор цветовой гаммы. Если у заказчика уже есть фирменный стиль, то дизайн рисуется на его основе. Если фирменного стиля нет, то необходимо продумать цветовую гамму всего сайта. Лучше использовать на сайте не более 2-3 цветов. Уже доказано, что цвет имеет прямое влияние на восприятие человеком и поэтому для сайтов определенной тематики лучше применять рекомендуемые цвета.

2. Подбор шрифтов. Шрифт сайта подбирается под стиль сайта. Заголовки страниц и разделов можно сделать нестандартным шрифтом. Шрифт основного текста страниц лучше использовать стандартный, легкочитаемый.

3. Ширина сайта. Бывают сайты фиксированной ширины и так называемые «резиновые сайты», когда сайт растягивается на всю ширину экрана, независимо от его размера. Также в последнее время в связи с ростом мобильных устройств растет количество адаптированных сайтов, которые меняют структуру сайта в зависимости от размера экрана устройства. Все это нужно учитывать при создании дизайна сайта.

4. Шапка сайта. Верхняя часть сайта часто называется шапкой, где обычно слева-направо располагается логотип, название компании, картинка шапки, часто контактные данные.

5. Меню сайта. Меню бывает горизонтальное и вертикальное. Горизонтальное меню обычно размещается под шапкой. Вертикальное меню располагается слева или справа.

6. Контент. Центральная часть содержит основной контент. Может делиться на отдельные блоки как по горизонтали так и по вертикали. По горизонтали обычно бывают двухколоночные и трехколоночные сайты.

7. Футер или подвал сайта. Так называется самая нижняя часть сайта. Здесь часто дублируют меню сайта, ставят копирайтинг, контактные данные и ссылку на разработчика.

Отрисованный дизайн разлаживают по слоям и обычно сохраняют в формате .psd, который удобно использовать при верстке сайта.

Создание уникального дизайна сайта обычно оценивают в 40% от стоимости сайта. Но можно использовать готовые PSD или HTML шаблоны, тогда стоимости этого этапа разработки значительно падает.

По времени создание дизайна сайта может занимать от 1 до 4-х дней.

3

Html-верстка

Верстка сайта – это процесс формирования (сборки) рабочей страницы из различных элементов на языке разметки HTML, XHTML или XML по нарисованному дизайну для дальнейшего корректного отображения в различных браузерах. Когда сайт отображается одинаково корректно во всех браузерах, его называют кроссбраузерным.

Верстка бывает табличная и блочная. Полная верстка при помощи таблиц уже отошла в прошлое и используется лишь частично при формировании табличных данных. Верстка при помощи блоков и таблиц стилей CSS считается более профессиональной, использующей концепцию семантичной верстки.

Как уже было сказано выше, сайты могут быть с фиксированной шириной, резиновые и адаптивные. Соответственно отличается и верстка таких сайтов. Самой сложной считается адаптивная верстка, т.к. она требует тщательной проработки нескольких макетов для разных размеров экрана.

Одним из показателей качественной верстки является ее валидность – отсутствие ошибок на сверстанных страницах и соответствие их стандартам организации The World Wide Web Consortium (W3C).

Сложность верстки зависит от сложности дизайна и занимает от 1 дня для простых проектов до 7 дней для сложных.

4 Программирование

Программирование сайта – это самый сложный процесс в разработке сайта, требующий наличия знаний некоторых языков программирования. Чаще всего используется серверный язык программирования PHP в связке с базой данных MySQL, а также скриптовый язык Javascript и различные библиотеки, например JQuery.

Программирование сайта подразумевает подключение на страницы сайта вывода текстовой информации из базы данных, подключения различных скриптов, а также администраторской панели для управления контентом на сайте.

При использовании самописной системы управления все модули и компоненты разрабатываются программистом вручную или используя фреймворки, при использовании готовых CMS (систем управления контентом), таких как, например, WordPress, Joomla или Drupal программист собирает систему управления как конструктор из готовых отдельных модулей и плагинов.

Стоимость программирования сайта в среднем может быть оценено в 50% от общей стоимости без учета наполнения. По времени программирование сайта может занимать от 1 дня до месяца (для сложных проектов).

5 Наполнение материалами

Контент сайта – это информационное содержимое сайта (тексты, таблицы, фотографии, видеозаписи). После создания сайта он фактически представляет из себя пустой каркас и для того, чтобы заинтересовать посетителей, его необходимо наполнить информацией.

От грамотности представленной информации будет зависеть, заинтересует ли сайт посетителей и потенциальных клиентов. Текстовое наполнение сайт – это важный инструмент рекламы и продаж.

Для составления таких текстов необходимо иметь специальные навыки, поэтому существует такая профессия, как **копирайтер**.

Существуют следующие термины:

Копирайтинг – это составление уникальных текстов для сайта.

Рерайтинг – это переработка уже имеющихся текстов для придания им уникальности.

Копипастинг – это использование чужих текстов на своем сайте. Категорически не рекомендую заниматься копипастингом, т.к. в поисковые машины воспримут этот текст как плагиат и понизят страницу в результатах выдачи, а автор текста может подать на вас в суд.

СЕО-копирайтинг и СЕО-рерайтинг – это составление опимизированного текста с использованием в нем наиболее значимых ключевых слов с определенной плотностью и частотой. Это делается с целью повышения позиций сайта в поисковой выдаче по

ключевым словам. Иными словами, сайты с СЕО-текстами будут занимать более высокие позиции в выдаче по сравнению с сайтами, использующими обычные тексты. Следовательно такие сайты будут иметь большую посещаемость и будут приносить своим владельцам больше продаж.

Далее я приведу некоторые рекомендации для страниц сайта.

Главная страница

На главной странице обычно показывают самые ходовые товары и услуги, также указывают общую информацию о компании, роде ее деятельности. Именно главная страница важна для поисковиков, поэтому текст должен быть тщательно продуман и должен включать наиболее значимые ключевые слова с определенной плотностью и частотой.

Особое значение имеет заголовок страницы, где в порядке важности указываются ключевые слова, использующиеся в тексте.

О нас (О Компании)

На этой странице нужно описать историю создания и развития компании, можно перечислить основные товары и услуги, расписать преимущества компании. Иногда имеет смысл показать фотографии сотрудников или управляющих.

Услуги (Товары)

На этой странице располагается перечень всех товаров и услуг. Обычно с этой страницы можно перейти к подробному описанию товара или услуги и сделать заказ.

Контакты

На странице располагаются контактные данные компании: номера телефонов, факса, адрес компании, электронная почта, скайп. Также на странице можно разместить контактную форму и интерактивную карту с указанием на ней месторасположения компании.

Выше я перечислил те страницы, которые включены в большинство стандартных бизнес-сайтов. Но на сайтах могут быть и другие страницы, например, новости, статьи, фотоальбомы, портфолио и т.п.

Обычно наполнение сайта обсуждается индивидуально и может не включаться в договор о создании сайта.

6 Тестирование

Тестирование сайта проводится с целью обнаружения и устранения различных багов, ошибок и неточностей в работе сайта.

Симулируется использованием сайта клиентами. Сайт заполняется тестовыми товарами и услугами и проверяется в работе.

Может быть проверена работа сайта при различном разрешении монитора, а также его кроссбраузерность в основных популярных браузерах.

7 Запуск

Запуск сайта подразумевает следующие шаги:

1. Загрузка сайта на хостинг и обеспечение его полной работоспособности.
2. Подписание Акта о сдаче-приемке работы.
3. Полная оплата всех произведенных работ по разработке сайта.
4. Передача заказчику всех исходных файлов сайта, а также всех данных для доступа к хостингу и административной части сайта.
5. Тестирование сайта клиентом в течение определенного срока, обычно 3-5 дней. В этот период все недочеты и ошибки по работе сайта в соответствии с ТЗ, должны быть выполнены разработчиком без дополнительной оплаты.

При необходимости заказчик и разработчик могут заключить договор на выполнение других работ, например, оптимизация и продвижение сайта, администрирование сайта.

Надеюсь, данная статья помогла Вам разобраться в основных этапах создания сайта. Буду рад увидеть ваши отзывы к этой статье.

Лекция 37
Отладка контента и размещение сайта на веб-сервере

В большинстве случаев невозможно избежать прямого контакта с сервером, даже если речь идет о простой загрузке файлов. По этой причине все дизайнеры должны обладать базовыми знаниями о серверах и их работе. По меньшей мере, это поможет более четко общаться с администратором сервера. Если есть разрешение для более широкого доступа к серверу, можно решать определенные задачи самостоятельно, без посторонней помощи.

Сервер – это любое управляющее компьютером программное обеспечение, которое дает ему возможность выполнять запросы на документы или другие данные. Программы, которые запрашивают и отображают документы (такие как браузер), называются клиентами. Термины "на стороне сервера" и "на стороне клиента", используемые, например, при работе с картами-изображениями, относятся к той машине, которая руководит процессом. Функции на стороне клиента выполняются на машине пользователя, функции на стороне сервера – на удаленной машине.

Web-серверы отвечают на запросы браузеров (клиентских программ), находят заданные файлы (или выполняют сценарий CGI) и возвращают документ или результаты сценария. Web-браузеры и серверы общаются по протоколу Hypertext Transfer Protocol (HTTP, протокол передачи гипертекста).

Программное обеспечение серверов. Большинство серверов работают на платформе Unix. Именно поэтому в мире Web по-прежнему используется терминология системы Unix. В процессе работы понадобится выучить несколько Unix-команд. Однако процент серверов Windows NT, Windows 95 и даже MacOS постоянно увеличивается. Некоторые серверные пакеты предлагают графический интерфейс в качестве альтернативы управлению из командной строки Unix.

Вот некоторые известные серверы: NCSA Server, Apache, CERN, Netscape Servers, Internet Information Server (IIS).

Сегодня большинство серверов (приблизительно 70%) работают на Apache или его предшественнике NCSA. Конкретный тип сервера не влияет на большую часть того, что делает дизайнер, например, на создание графики или разработку базовых HTML-файлов. Конечно, он будет влиять на более совершенные методы создания Web-сайтов, такие как Server Side Includes (серверные включения), добавление типов MIME и Web-страницы, управляемые базами данных.

Корневой каталог. Когда браузер запрашивает документ, сервер определяет местонахождение документа, начиная с корневого каталога документа. Это каталог, который был сконфигурирован для хранения всех документов, совместно используемых посредством Web. Он не обязательно появляется в URL, который указывает на документ, поэтому важно знать, какой каталог является корневым при загрузке файлов.

Индексные файлы. Прямой слэш (/) в конце URL означает, что URL обращается к каталогу, а не к файлу. По умолчанию серверы отображают содержание каталога, указанного в URL. Большинство серверов, однако, сконфигурированы так, чтобы отображать особый файл вместо списка каталога, этот файл называется индексным. Индексные файлы обычно имеют имя index.html, но на некоторых серверах они могут называться welcome.html или default.html.

Если сервер сконфигурирован на нахождение индексного файла и не обнаруживает его, вместо него может быть отображено содержание каталога, но это делает файлы уязвимыми для посторонних. По этой причине неплохо всегда называть одну из страниц (обычно главную) в каждом каталоге index.html (или другим условленным именем).

Заголовки отклика HTTP. Как только сервер определяет месторасположение файла, он посылает содержимое этого файла обратно браузеру вместе с некоторыми заголовками отклика HTTP (response headers). Эти заголовки обеспечивают браузер информацией о прибывающем файле, включая его тип данных (также известный как "тип содержимого" или "тип MIME"). Обычно сервер определяет формат по расширению файла; например, файл с расширением .gif распознается как файл изображения.

Браузер читает информацию в заголовке и определяет, что делать с файлом. Он

может отобразить файл в окне либо запустить соответствующее вспомогательное или встраиваемое приложение (plug-ins).

Сценарии CGI. Вместо того чтобы обращаться к файлам HTML, URL может потребовать запустить программу CGI. CGI расшифровывается как Common Gateway Interface (общий шлюзовой интерфейс). Он позволяет Web-серверу общаться с другими программами (сценариями CGI), которые работают на сервере. Сценарии CGI обычно пишут на языках Perl, C или C++.

Сценарии CGI используют для выполнения разнообразных функций, таких как поиск, управление картами-изображениями на стороне сервера, игры. Однако наиболее типичное использование сценариев – обработка форм.

Большинство администраторов серверов придерживаются правила хранить сценарии CGI в специальном каталоге, озаглавленном cgi-bin (сокращенно от CGI-binaries). Когда они хранятся в одном каталоге, администраторам проще управлять сервером и обеспечивать его безопасность. Если сценарий CGI запрашивается браузером, сервер исполняет функцию и возвращает браузеру динамическое содержимое.

Использование SSI. SSI – Server Side Includes или, по-русски, – включения на стороне сервера. SSI – это директивы, вставляемые прямо в HTML-код и служащие для передачи указаний Web-серверу. Встречая такие директивы, которые называются SSI-вставками, Web-сервер интерпретирует их и выполняет соответствующие действия. Например: вставка HTML-фрагмента из другого файла, динамическое формирование страничек в зависимости от некоторых переменных (например, типа браузера) и другие не менее приятные вещи.

Преимущества SSI проявляются, когда нам нужно поддерживать достаточно большой по объему сайт, имеющий определенную структуру и повторяющиеся элементы кода на всех страничках. При применении серверных включений сайт удобно рассматривать как состоящий из отдельных блоков, каждый из которых отвечает за свою часть странички. Эти блоки практически неизменны и повторяются от страницы к странице. В эти блоки можно вынести такие элементы странички как: главное меню, рекламные вставки, повторяющиеся элементы оформления страничек и т.д. Физически эти блоки представляют собой просто HTML-файлы, содержащие часть кода, нужную для выполнения их задачи.

Для того, чтобы сервер знал, что страничка не обычная, а содержит SSI-директивы, она имеет специальное расширение: *.shtml или *.shtm, наличие которого и заставляет Web-сервер предварительно обрабатывать странички. Вообще-то, расширение может быть любое – в зависимости от конфигурации Web-сервера, но в основном применяется именно *.shtml.

Полная страничка формируется Web-сервером на лету, собирая код странички из таких вот блоков. Для того, чтобы указать серверу, какой блок нужно вставить и в каком месте странички, используется специальная форма записи в виде комментария. Например, вот такая:

<!--#command param="value" –->,

где # – признак начала SSI-вставки; command – SSI-команда; param – параметры SSI-команды.

Первое преимущество SSI с точки зрения дизайнера заключается в том, что при таком подходе Web-мастеру, занимающемуся поддержкой сайта, можно не бояться случайно испортить дизайн. Элементы сложной верстки скрыты за счет использования SSI, и поддержка содержимого страничек становится гораздо более легким и приятным делом.

Второе, не менее важное преимущество, – это возможность мгновенной замены дизайна сайта, не требующая переделывания страничек с информационным содержанием сайта. Для смены дизайна достаточно переписать SSI-вставки, формирующие внешний вид сайта.

Unix. На компьютерах IBM PC и Macintosh большинство функций можно выполнить, используя инструментарий с графическим интерфейсом. Тем не менее, иногда невозможно найти замену для старой сессии Telnet. Telnet – протокол "эмуляции

терминала", позволяющий вам регистрироваться в другой компьютерной системе или сети, такой как Internet. Этот термин также используют применительно к любому приложению, служащему для общения с использованием протокола Telnet. Программа Telnet предоставит текстовое терминальное окно другой системы, в котором можно вводить инструкции в командной строке.

Регистрация. Когда с помощью программы Telnet открывается сеанс связи с удаленным сервером, появляется приглашение зарегистрироваться в системе. Прежде чем получить доступ к серверу, системному администратору нужно установить для пользователя учетную запись (account) Unix.

При успешной регистрации, вы получите подсказку системы Unix %, (или иногда #) в зависимости от разновидности Unix, на которой работает сервер. Это знак, что система готова. С этого момента используется программа под названием shell. Она интерпретирует печатаемые вами команды и вызывает запрашиваемые программы. Перемещение в домашний каталог будет автоматическим.

Чтобы закончить сессию Telnet достаточно набрать logout или exit.

Структуры каталогов. Поскольку Web является порождением окружения Unix, он следует многим его соглашениям. Например, чтобы в гиперссылке правильно указать путь, URL, требуется понимание организации каталогов на платформе Unix.

Каталоги ("место хранения файлов") организованы в виде иерархической структуры. Самый верхний каталог известен как корневой и обозначается прямым слэшем (/). Корневой может содержать несколько каталогов, каждый из которых может иметь свои подкаталоги, и т. д. Говорят, подкаталог является "потомком" каталога, в состав которого он входит (последний называют "родителем").

Имя пути (path name) – это нотация, используемая для указания конкретного файла или каталога. Оно указывает, какой путь нужно пройти, чтобы добраться до желаемой цели. Есть два типа путей: абсолютный (из корневого каталога) и относительный (из каталога в котором находишься).

Соглашения об именах файлов. Для того чтобы файлы удачно путешествовали по сети, нужно назвать их в соответствии с установленными соглашениями об именах файлов:

– запрещено использование пробелов в именах файлов. Хотя это абсолютно приемлемо для локальных файлов в MacOS и Windows 95/98/NT, пробелы не распознаются другими системами;

– нежелательно использование в именах файлов специальных символов, таких как ?, %, #. Лучше ограничиться буквами, цифрами, символом подчеркивания (вместо пробелов), дефисами и точками;

– выбор расширения должен быть правильным. Документы HTML требуют расширения .html (или htm для сервера Windows). Графические файлы GIF имеют расширение .gif, а файлы JPEG – .jpg или .jpeg. Файлы с неправильным расширением браузер не распознает в качестве файлов, разрешенных для передачи в Web;

– имена файлов HTML чувствительны к регистру. Хотя это и не обязательно, но лучше использовать только буквы нижнего регистра. Это может облегчить запоминание имен файлов.

Загрузка документов (FTP). Наиболее частой транзакцией, с которой Web-дизайнер обращается к серверу, является загрузка на сервер HTML-документов, изображений или мультимедийных файлов. По сети файлы передаются между компьютерами с помощью метода, называемого FTP (File Transfer Protocol, протокол передачи файлов). При работе на Unix в сессии Telnet можно запустить программу ftp и передавать файлы с огромным числом аргументов командной строки.

На компьютерах IBM PC и Macintosh имеется ряд программ FTP с графическими интерфейсами, которые избавляют от необходимости передавать файлы с использованием командной строки Unix. Более того, в большинстве случаев функции FTP встроены непосредственно в WYSIWYG HTML-редакторы, такие как GoLive Cyberstudio, Claris HomePage и Dreamweaver. На компьютерах Macintosh достаточно популярны специальные программы, которые осуществляют передачу файлов методом "перетаскивания". На PC

есть немало простых программ FTP, таких как WS__FTP и AceFTP.

Браузеры Netscape Navigator и Internet Explorer также функционируют как простые FTP-клиенты, предоставляя возможность как загружать, так и выгружать файлы, используя интерфейс "drag-and-drop".

Процесс FTP. Независимо от того, какой инструментарий используется, базовые принципы и процессы остаются неизменными.

1. Запуск программы FTP, соединение с сервером. Нужно ввести точное имя сервера, регистрационное имя и пароль.

2. Находим каталог, в который нужно скопировать файлы. Можно также создать новый каталог или удалить существующие файлы и каталоги на сервере, используя средства управления FTP-программы.

3. Задаем режим передачи. Главное во время передачи – решить, передавать данные в двоичном или ASCII –режиме. ASCII-файлы состоят из буквенно-цифровых символов. Некоторые FTP-программы рассматривают файлы ASCII как "текстовые". Документы HTML следует передавать как ASCII или текст.

Двоичные файлы состоят из откомпилированных данных (единиц и нулей), их примерами являются выполняемые программы, изображения, фильмы и т.д. Некоторые программы рассматривают двоичный режим как "необработанные данные" ("raw data") или "изображение" ("Image"). Все графические (.gif или .jpeg) и мультимедийные файлы должны передаваться как двоичные или "Raw Data". В Fetch (MacOS) вы можете обнаружить параметр MacBinary, который передает файл полностью, вместе с ветвью ресурсов (часть файла, содержащая пиктограммы рабочего стола и другие специфические для компьютера Macintosh данные). Этот вариант следует использовать только при передаче с одного компьютера Macintosh на другой. Ветвь ресурсов отделяется от мультимедийных файлов, созданных на компьютерах Macintosh, при передаче в двоичном режиме.

Некоторые FTP-программы имеют также параметр Auto, который позволяет вам передавать полностью весь каталог, содержащий файлы обоих типов. Программа проверяет каждый файл и определяет, следует ли передавать его в текстовом или двоичном режиме. Эта функция не во всех программах надежна на 100%, поэтому надо использовать ее с осторожностью, пока не будет уверенности, что результат правильный.

4. Передача файлов на сервер. Стандартный протокол FTP использует термины "поместить" ("put") для обозначения передачи файлов с компьютера пользователя на сервер и "получить" ("get") для обозначения загрузки файла с сервера на компьютер, поэтому они могут использоваться и в программе FTP. Одновременно можно загружать несколько файлов.

5. Разъединение. Когда передача закончена, связь с сервером завершается. Перед этим можно протестировать переданные в браузер файлы, чтобы убедиться, что передача прошла удачно.

Установка права доступа. При передаче файлов на Web-сервер, нужно убедиться, что все права доступа к файлам установлены таким образом, что каждый может их прочитать. Право доступа контролирует, кто может читать, записывать (редактировать) или выполнять файл (если это программа). Права доступа требуется установить для владельца файла, группы файла или для всех пользователей. Устанавливать права доступа может только автор файла.

Установка права доступа программой FTP. Некоторые FTP-про-граммы позволяют устанавливать право доступа по умолчанию в диалоговом окне. Для достижения большинства целей Web пользователю нужно обеспечить полный доступ и ограничить всех других пользователей только возможностью чтения. Может понадобиться согласие администратора сервера с такими установками.

Типы файлов (MIME-типы). Серверы добавляют к каждому документу заголовок, который сообщает браузеру, какой тип файла он отправляет. Основываясь на этой информации, браузер определяет, что делать с файлом: или отобразить его содержимое в окне, или запустить соответствующее встраиваемое или вспомогательное приложение.

Система для сообщения мультимедийных типов файлов очень напоминает MIME (Multipurpose Internet Mail Extension, многоцелевые расширения почтовой службы в Интернете), которая была изначально разработана для вложений в письма электронной почты. Чтобы успешно сообщать тип файла браузеру, нужно чтобы сервер был сконфигурирован на распознавание любого MIME-типа. Если нужно доставлять данные, выходящие за рамки стандартных HTML-файлов и изображений (например, видео Shockwave Flash или аудиофайлы), следует уточнить у администратора сервера, способен ли сервер поддерживать этот MIME-тип. Большинство распространенных форматов встроены в текущие версии программного обеспечения сервера. Если нет, то администратор может легко их установить, следует лишь предоставить ему необходимую информацию.

Точный синтаксис для конфигурирования MIME-типов варьируется в разных серверных программах. Тем не менее всем требуется одна и та же базовая информация: тип, подтип и расширение. Типы – это наиболее общие категории файлов. Они включают текст, изображение, аудио, видео, приложение и т.д. В каждой категории есть ряд подтипов. Например, файловый тип image (изображение) включает подтипы gif, jpeg и т.д. Расширение файла используется сервером для определения типа файла и его подтипа. Не все расширения стандартизированы.