Учреждение образования
«БЕЛОРУССКАЯ ГОСУДАРСТВЕННАЯ АКАДЕМИЯ СВЯЗИ»
Кафедра ПОСТ

КОНСПЕКТ ЛЕКЦИЙ
ПО ДИСЦИПЛИНЕ
«ПРОГРАММИРОВАНИЕ ДЛЯ ИНТЕРНЕТ»

для учащихся дневной формы получения образования
по специальности
2-40 01 31 – Тестирование программного обеспечения

Составитель: Лущик Н. И.

Утвержден на заседании кафедры ПОСТ
протокол № 1 от 30.08.2018 г.

Зав. кафедрой ПОСТ _____ РыбакВ.А.

Минск,
2018 г.

СОДЕРЖАНИЕ

РАЗДЕЛ 1 ОСНОВЫ СОЗДАНИЯ WEB-ПРОЕКТОВ

Тема 1.1 Web-проект. Этапы создания

Web-проект – проект по созданию различных информационных систем в Интернете на основе веб-технологий.

Веб-разработка (от англ. Web development) - процесс создания веб-сайта или веб-приложения. Процесс создания веб-сайта включает в себя веб-дизайн, вёрстку веб-страниц, веб-программирование, а также конфигурирование веб-сервера.

Основные этапы создания веб-сайта:

- Определение целей веб-сайта и его позиционирование;
- Создание Технического Задания (ТЗ) на разработку веб-сайта;
- Создание дизайн-макета веб-сайта;
- Верстка сайта;
- Программирование сайта;
- Наполнение сайта информацией;
- Расположение сайта в сети Интернет.
- Тестирование сайта.

1. Определение целей веб-сайта и его позиционирование

На этом этапе необходимо определить, для чего нужен сайт, т.е. какие задачи он должен решать: предоставить общее представление о компании или многосторонне осветить какую-либо сторону человеческой деятельности, увеличить продажи по традиционным каналам или организовать веб-торговлю, провести рекламную или маркетинговую кампанию.

Цели веб-сайта, в большинстве случаев, должны ставиться заказчиком, а затем, вместе с исполнителем они уточняются и корректируются.

Это один из самых важных этапов не только создания веб-сайта как такового, но важнейший этап интернет-маркетинга.

Если заказчик не понимает, для чего ему нужен веб-сайт, с 99% вероятностью он будет недоволен работой исполнителя и будет считать, что деньги, потраченные на создание веб-сайта, просто потеряны.

После определения целей сайта надо совершенно четко и как можно подробнее представить и описать целевую аудиторию сайта, т.к. это влияет на то, в каком виде будет представлена информация.

Определение целевой аудитории веб-сайта - это не менее важный, чем определение целей веб-сайта, этап. Не всегда целевая аудитория компании из офлайн будет прямо проецироваться на онлайн. Зная целевую аудиторию и аудиторию российской части Интернета можно сделать некие предположения о том, кто будет являться основными посетителями веб-сайта.

Определение и как можно более подробное описание целевой аудитории сайта дает возможность разработать правильный дизайн для проекта, а также выбрать правильное направление для написания текстов. Очень важно говорить с аудиторией на понятном ей языке.

На заключительной стадии этого этапа примерно определяют, по каким поисковым запросам сайт должен появляться в результатах поискового запроса и посмотреть интернет-проекты конкурентов будущего веб-проекта.

Определение поисковых запросов или другими словами составление семантического ядра процесс достаточно сложный и в идеале должен делаться профессионалами. Однако, составить приблизительный список запросов под силу каждому руководители или владельцу бизнеса. Для этого просто необходимо выписать все те слова и фразы, которые вы лично используете для поиска конкурентов и опросить менеджеров отдела продаж – какие вопросы возникают у клиентов вашей компании наиболее часто.

Дополнительную информацию о положении вещей в Интернете на рынке даст обзор сайтов конкурентов. Необходимо иметь ввиду, что совсем необязательно конкуренты из реальной жизни будут являться таковыми в Интернете. Просмотрите некоторое количество сайтов, которые расположены в ТОП10 ведущих поисковых машин, обратите внимание на графическое решение, на тексты, на поисковые запросы.

2. Создание Технического Задания (ТЗ) на разработку веб-сайта

В ТЗ необходимо как можно более подробно описать:

– цели создания сайта и его целевую аудиторию;

– структуру веб-сайта и количество страниц в каждом разделе;

– работу динамических модулей;

– пожелания по дизайну (цвета, использование фирменного стиля, соотношение графика/текст ;

– используемые технологии (HTML, PHP и проч.);

– порядок предоставления, обработки или создания графической и текстовой информации;

– технические требования к сайту.

ТЗ является основным документом, на основе которого осуществляются все последующие этапы разработки веб-сайта.

Очень часто создается ситуация, при которой заказчик ждет ТЗ от исполнителя, а исполнитель от заказчика. Создание веб-сайта – это сотрудничество. В одиночку, ни заказчик, ни исполнитель не могут составить ТЗ. Как правило, заказчик должен четко описать цели веб-сайта, его целевую аудиторию и пожелания по функционалу. Далее за дело берется исполнитель, который техническим языком описывает, как будет работать будущий сайт.

3. Создание дизайн-макета веб-сайта

На этом этапе дизайнер в специальной графической программе создает дизайн страниц будущего веб-сайта с прорисовкой всех графических (банеров, кнопок, фотографий) и текстовых элементов. Дизайнер создает дизайн веб-страниц с учетом пожеланий заказчика и задания, прописанного в ТЗ.

Современные мониторы имеют различные разрешения. В разговоре о веб-сайтах важной является ширина. Размер монитора по ширине в пикселях может быть 800, 1024, 1280,1600 и даже больше. Более того, пользователь может просматривать сайт в полуоткрытом окне.

Фиксированный дизайн сайта предполагает одну единую ширину сайта для всех разрешений экрана и для всех ширин окон, т.е. если ширина сайта 1000 пикселейй, то на экране 800 пикселей будет появляться горизонтальная прокрутка, а на экране 1280 пикселей – пустые поля слева и справа (или только с одной стороны)

Такое построение сайта позволяет четко управлять композицией сайта и однозначно знать, где будет каждый элемент навигации на каждом экране. Этот тип сайта позволяет использовать более насыщенные графические элементы.

Резиновый дизайн, в отличие от фиксированного «подстраивается» под ширину экрана. Это приводит к тому, что сайт без горизонтальной прокрутки увидит большинство пользователей, но на разных экранах композиция сайта будет непредсказуемой, что сильно не по душе дизайнерам.

4. Верстка сайта

После того, как Заказчик утвердил дизайн-макет (в письменном виде или по электронной почте) за работу принимается верстальщик – это тот человек, который переводит дизайн-макет на язык, понятный компьютеру с использованием языка HTML.

5. Программирование сайта.

Очень часто этап программирования и верстки объединяют в один. На мелких и средний проектах оба действия в состоянии выполнить один человек. На крупных проектах в силу специфики работ эти этапы разделяют.

На этапе программирования (как правило, с использованием JavaScript, PHP, Perl, ASP и баз данных) происходит создание всех страниц сайта, определяется порядок работы меню, расставляются гиперссылки, создается динамика на сайте, программируются такие составляющие, как гостевая книга, форум, новостная лента и т.д.

Если сайт должен иметь администраторский интерфейс то он создается именно на этапе программирования.

Очень важно на этапе программирования определить, на какой системе администрирования (Content Management Site) будет работать ваш сайт. На сегодня на рынке существует великое множество систем администрирования.

В результате работы верстальщика и программиста получается сайт без информационного наполнения. Физически сайт в таком виде представляет из себя набор файлов.

Сроки выполнения работ по верстке и программированию зависят от сложности проекта и могут составлять от недели до 2-х месяцев.

6. Наполнение сайта информацией.

На этом этапе информация, предоставленная Заказчиком, размещается на сайте, т.е. путем перевода в специальный формат текст и графика располагаются на сайте на определенных страницах, и эта информация становится доступной для просмотра.

Срок исполнения опять же зависит от сложности проекта, объема информации, который надо расположить на сайте, и того вида, в котором Заказчик ее представил. Если информация представлена в электронном вмде, а графика не требует доплнительной обработки (например, изменение размера, добавление дополнительных элементов и проч.), то наполнение сайта происходит достаточно быстро. Если необходима дополнительная работа, как например, поиск или набор текста, сканирование фотографий или создание рисунков, то этап наполнения сайта может стать одним из самых длительных.

7. Расположение сайта в сети Интернет.

Данный этап необязательно выполняется после проведения всех вышеперечисленных работ. Он может проводиться параллельно с любым из этапов. Он заключается в том, что файлы сайта располагаются на хостинге.

8. Тестирование сайта.

Этот этап можно осуществить как до, так и после размещения сайта по его «родному» адресу. На этом этапе выявляются все ошибки и недочеты в программировании и написании текстов. Срок тестирования зависит от сложности проекта, но, как правило, не превышает 1 месяца.

И вот, когда тестирование закончено, наступает момент размещения сайта. Вопреки расхожему мнению, после того как сайт выложен, работа с ним не заканчивается. Если цель – превратить сайт в инструмент маркетинга, то необходимо будет:

- выкладывать новые материалы
- продвигать сайт

опрашивать посетителей и добавлять новую необходимую им функциональность

РАЗДЕЛ 2 ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА И ТЕХНОЛОГИИ

Тема 2.1 HTML как основа web-программирования

HTML (HyperText Markup Language – «язык гипертекстовой разметки») - веб-технология, без которой невозможно создавать даже самые простые интернет-страницы. Все сайты создаются по правилам HTML.

В общем можно сказать, что HTML - это набор правил, согласно которым строится структура интернет-документов. С помощью этих правил можно придать страницам желаемые параметры - такие, как расположение элементов на странице, их цвет, цвет фона, ссылки, название, изображения и т.д.

Основное понятие в HTML - это тег. Тег задает нужную информацию, согласно которой будет показываться веб-страница.

Тэги это метки, которые вы используете для указания браузеру, как он должен показывать ваш web-сайт. Все тэги имеют одинаковый формат: они начинаются знаком "<" и заканчиваются знаком sign ">". Обычно имеются два тэга - открывающий: <html> и закрывающий: </html>. Это парные теги. Бывают ещё одинарные.

Различие в том, что в закрывающем имеется слэш "/". Всё содержимое, помещённое между открывающим и закрывающим тэгами, является содержимым тэга.

Для того, чтобы прямо указать браузеру клиента, как показывать элемент, задаваемый тегом, существуют атрибуты. Например, есть тег <h1 align="center">, задающий заголовок 1-го уровня. При этом у него существует атрибут align со значением "center", задающий выравнивание текста заголовка по центру (можно также задать выравнивание по левому краю или по правому).

Структура HTML-документа:

HTML-документ заключается в теги <html> и </html>. Между этими тегами располагаются два раздела: раздел заголовка (элемент head) и раздел тела документа (элемент body для простого документа либо элемент frameset, задающий набор кадров). Все указанные элементы имеют начальный и конечный тег.

Секция заголовка содержит описание параметров, используемых при отображении документа, но не отражающихся непосредственно в окне браузера. Секция тела документа содержит текст, предназначенный для отображения браузером и элементы, указывающие на способ форматирования текста, определяющие графическое оформление документа, задающие параметры гиперссылок и так далее.

```
<html>
<head>
<title>codebra</title>
</head>
```

<body>

Это просто
 текст.

</body>

</html>

Тег DOCTYPE, согласно спецификации HTML, объявляет тип документа. То есть он объявляет валидатору, какую версию документа использовать. Тег DOCTYPE должен находиться в первой строке каждого документа HTML. Он очень важен для правильного отображения страниц сайта браузером.

Синтаксис

<!DOCTYPE [Элемент верхнего уровня] [Публичность] "[Регистрация]//[Организация]//[Тип] [Имя]//[Язык]" "[URL]">

Параметры

Элемент верхнегоуровня — указывает элемент верхнего уровня в документе, для HTML это тег <html>.

Публичность — объект является публичным (значение PUBLIC) или системным ресурсом (значение SYSTEM), например, таким как локальный файл. Для HTML/XHTML указывается значение PUBLIC.

Регистрация — сообщает, что разработчик DTD зарегистрирован в международной организации по стандартизации (International Organization for Standardization, ISO). Принимает одно из двух значений: плюс (+) — разработчик зарегистрирован в ISO и - (минус) — разработчик не зарегистрирован. Для W3C значение ставится «-».

Организация — уникальное название организации, разработавшей DTD. Официально HTML/XHTML публикует W3C, это название и пишется в <!DOCTYPE>.

Тип — тип описываемого документа. Для HTML/XHTML значение указывается DTD.

Имя — уникальное имя документа для описания DTD.

Язык — язык, на котором написан текст для описания объекта. Содержит две буквы, пишется в верхнем регистре. Для документа HTML/XHTML указывается английский язык (EN).

URL — адрес документа с DTD.

Большинству браузеров безразлично, в каком регистре введены буквы тэгов. <HTML>, <html> или <HtMl> обычно даёт одинаковый результат. Однако корректным будет нижний регистр. Поэтому привыкайте печатать тэги в нижнем регистре.

Атрибуты тела документа:

– <body bgcolor=?> Устанавливает цвет фона документа, используя значение цвета в стандарте RGB - пример: FFFF00 - желтый цвет.

– <body text=?> Устанавливает цвет текста документа, используя значение цвета в стандарте RGB - пример: 00ff00 - зеленый цвет.

– <body link=?> Устанавливает цвет гиперссылок, используя значение цвета в стандарте RGB - пример: 00FF00 - зеленый цвет.

– <body vlink=?> Устанавливает цвет гиперссылок, на которыхх вы уже побывали, используя значение цвета в виде стандарте RGB - пример: 333333 - темно-серый цвет.

– <body alink=?> Устанавливает цвет гиперссылок при нажатии.

название ссылки

Атрибут href задаёт значение адреса документа, на который указывает ссылка.

– filename — имя файла или адрес Internet, на который необходимо сослаться.

– название ссылки — название гипертекстовой ссылки, которое будет отображаться в браузере, то есть показываться тем, кто зашёл на страницу.

– target — задаёт значение окна или фрейма, в котором будет открыт документ, на который указывает ссылка. Возможные значения атрибута:

– _top — открытие документа в текущем окне;

– _blank — открытие документа в новом окне;

– _self — открытие документа в текущем фрейме;

– _parent — открытие документа в родительском фрейме.

Значение по умолчанию: _self.

Фрейм (от англ. frame — рамка) — отдельный, законченный HTML-документ, который вместе с другими HTML-документами может быть отображён в окне браузера.

Фреймы по своей сути очень похожи на ячейки таблицы, однако более универсальны. Фреймы разбивают веб-страницу на отдельные миникадры, расположенные на одном экране, которые являются независимыми друг от друга. Каждое окно может иметь собственный адрес. При нажатии на любую из ссылок, расположенных в одном фрейме, можно продолжать видеть страницы в других окнах.

Фреймы часто использовались для навигации по веб-сайту. При этом навигационная страница располагается в одном окне, а страницы с текстом — в другом.

В настоящее время использование фреймов для публичных сайтов не рекомендовано. Главным образом это связано с принципом работы поисковых машин, которые приводят пользователя к HTML-документу, являющемуся согласно задумке лишь одним из фреймов того, что автору сайта хотелось бы представить.

Гиперссылки

 ТЕКСТ Создает гиперссылку на другие документы или часть текущего документа. Здесь URL адрес ссылки, ТЕКСТ - текст ссылки.

< img src="imgURL" > Создает гиперссылку на рисунок, находящийся по адресу imgURL.

"URL" = "links/main.htm" Адрес документа main.htm, находящегося в локальной папке links данного компьютера.

"URL" = http://www.rambler.ru Ссылка на ресурс, находящийся на удаленном компьютере. В адресе присутствуют: программа связи с удаленным компьютером http (HyperText Transfer Protocol, разделители :// и интернет (IP) адрес искомого ресурса (в данном случае поискового сервера Rambler).

 Создает гиперссылку вызова почтовой программы для написания письма по указанному адресу.

 Отмечает часть текста как место перехода по гипперссылке в документе.

 Создает гиперссылку на помечанную часть текущего документа.

Теги форматирования текста

<pre></pre> Обрамляет предварительно отформатированный текст.

<h1></h1> Создает самый большой заголовок (как отдельный абзац)

<h6></h6> Создает самый маленький заголовок.

 Создает жирный текст

 Создает наклонный текст

  Устанавливает размер текста в пределах от 1 до 7.

  Устанавливает цвет текста, используя значение цвета в виде RRGGBB.

Форматирование

<p> Создает новый параграф

<p align=?> Выравнивает параграф относительно одной из сторон документа, значения: left, right, justify или center

Вставляет перевод строки.

<dl></dl>Создает список определений.

<dt> Определяет каждый из терминов списка

<dd> Описывает каждое определение

 Создает нумерованный список

Определяет каждый элемент списка и присваивает номер

 Создает ненумерованный список

 Предваряет каждый элемент списка и добавляет кружок или квадратик.

 Тег, используемый для форматирования больших блоков текста HTML документа.

Графические элементы

<img src="name"> Добавляет изображение в HTML документ

<img src="name" align=?> Выравнивает изображение к одной из сторон документа, принимает значения: left, right, center; bottom, top, middle

<img src="name" border=?> Устанавливает толщину рамки вокруг изображения

<hr> Добавляет в HTML документ горизонтальную линию.

<hr size=?> Устанавливает высоту(толщину) линии

<hr width=?> Устанавливает ширину линии, можно указать ширину в пикселях или процентах.

<hr noshade> Создает линию без тени.

<hr color=?> Задает линии определенный цвет. Значение RRGGBB.

Таблицы

<table></table> Создает таблицу.

<tr></tr> Определяет строку в таблице.

<td></td> Определяет отдельную ячейку в таблице.

<th></th> Определяет заголовок таблицы (нормальная ячейка с отцентрованным жирным текстом)

Атрибуты таблицы

<table border=#> Задает толщину рамки таблицы.

<table cellspacing=#> Задает расстояние между ячейками таблицы.

<table cellpadding=#> Задает расстояние между содержимым ячейки и ее рамкой.

<table width=#> Устанавливает ширину таблицы в пикселах или процентах от ширины документа.

<tr align=?> или <td align=?> Устанавливает выравнивание ячеек в таблице, принимает значения: left, center, или right.

<tr valign=?> или <td valign=?> Устанавливает вертикальное выравнивание для ячеек таблицы, принимает значения : top, middle, или bottom.

<td colspan=#> Указывает кол-во столбцев которое объединено в одной ячейке. (по умолчанию=1)

<td rowspan=#> Указывает кол-во строк которое объединено в одной ячейке. (по умолчанию=1)

<td nowrap> Не позволяет программе просмотра делать перевод строки в ячейке таблицы.

Тема 2.2 CSS как основа web-программирования

"CSS" расшифровывается как "cascading style sheets", что в переводе с английского означает "таблицы каскадных стилей". Эта технология заметно упростила жизнь вебмастерам, поскольку позволила отделить оформление веб-страниц от основного кода. По сути это язык веб-программирования, определяющий отображение HTML-страниц. С помощью команд CSS можно задавать такие параметры страницы, как цвет, шрифт, отступ, задавать поля и многое еще. Язык HTML в каких-то случаях может использоваться не совсем верно для оформления страниц. Такие страницы неправильно отбражаются, но CSS сделал возможным редактировать

оформление куда точнее и эффективнее. К тому же, CSS поддерживается на сегодня всеми браузерами, а это очень важно для правильной передачи информации.

CSS отличается от HTML тем, что HTML позволяет создавать нужную структуру страницы, а посредством CSS уже готовая структура и содержимое веб-страниц подвергается форматированию.

Стили являются удобным, практичным и эффективным инструментом при вёрстке веб-страниц и оформления текста, ссылок, изображений и других элементов. Несмотря на явные плюсы применения стилей, рассмотрим все преимущества CSS, в том числе и незаметные на первый взгляд.

Достоинства:

1. Разграничение кода и оформления

Идея о том, чтобы код HTML был свободен от элементов оформления вроде установки цвета, размера шрифта и других параметров, стара как мир. В идеале, веб-страница должна содержать только теги логического форматирования, а вид элементов задаётся через стили. При подобном разделении работа над дизайном и версткой сайта может вестись параллельно.

2. Разное оформление для разных устройств

С помощью стилей можно определить вид веб-страницы для разных устройств вывода: монитора, принтера, смартфона, планшета и др. Например, на экране монитора отображать страницу в одном оформлении, а при её печати — в другом. Эта возможность также позволяет скрывать или показывать некоторые элементы документа при отображении на разных устройствах.

3. Расширенные по сравнению с HTML способы оформления элементов

В отличие от HTML стили имеют гораздо больше возможностей по оформлению элементов веб-страниц. Простыми средствами можно изменить цвет фона элемента, добавить рамку, установить шрифт, определить размеры, положение и многое другое.

4. Ускорение загрузки сайта

При хранении стилей в отдельном файле, он кэшируется и при повторном обращении к нему извлекается из кэша браузера. За счёт кэширования и того, что стили хранятся в отдельном файле, уменьшается код веб-страниц и снижается время загрузки документов.

Кэшем называется специальное место на локальном компьютере пользователя, куда браузер сохраняет файлы при первом обращении к сайту. При следующем обращении к сайту эти файлы уже не скачиваются по сети, а берутся с локального диска. Такой подход позволяет существенно повысить скорость загрузки веб-страниц.

5. Единое стилевое оформление множества документов

Сайт — это не просто набор связанных между собой документов, но и одинаковое расположение основных блоков, и их вид. Применение единообразного оформления заголовков, основного текста и других элементов создает преемственность между страницами и облегчает

пользователям работу с сайтом и его восприятие в целом. Разработчикам же использование стилей существенно упрощает проектирование дизайна.

6. Централизованное хранение

Стили, как правило, хранятся в одном или нескольких специальных файлах, ссылка на которые указывается во всех документах сайта. Благодаря этому удобно править стиль в одном месте, при этом оформление элементов автоматически меняется на всех страницах, которые связаны с указанным файлом. Вместо того чтобы модифицировать десятки HTML-файлов, достаточно отредактировать один файл со стилем и оформление нужных документов сразу же поменяется.

Для добавления стилей на веб-страницу существует несколько способов, которые различаются своими возможностями и назначением.

Связанные стили

При использовании связанных стилей описание селекторов и их значений располагается в отдельном файле, как правило, с расширением css, а для связывания документа с этим файлом применяется тег <link>. Данный тег помещается в контейнер <head>.

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Стили</title>
<link rel="stylesheet" href="http://htmlbook.ru/mysite.css">
</head>
<body>
<h1>Заголовок</h1>
<p>Текст</p>
</body> </html>
```

Файл со стилем

```
H1 {
color: #000080;
font-size: 200%;
font-family: Arial, Verdana, sans-serif;
text-align: center;
}
P {
padding-left: 20px;
```

```
}
```

Как видно из данного примера, файл со стилем не хранит никаких данных, кроме синтаксиса CSS. В свою очередь и HTML-документ содержит только ссылку на файл со стилем, т. е. таким способом в полной мере реализуется принцип разделения кода и оформления сайта. Поэтому использование связанных стилей является наиболее универсальным и удобным методом добавления стиля на сайт. Ведь стили хранятся в одном файле, а в HTML-документах указывается только ссылка на него.

Глобальные стили

При использовании глобальных стилей свойства CSS описываются в самом документе и располагаются в заголовке веб-страницы. По своей гибкости и возможностям этот способ добавления стиля уступает предыдущему, но также позволяет хранить стили в одном месте, в данном случае прямо на той же странице с помощью контейнера <style>.

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Глобальные стили</title>
<style>
H1 {
font-size: 120%;
font-family: Verdana, Arial, Helvetica, sans-serif;
color: #333366;
}
</style>
</head>
<body>
<h1>Hello, world!</h1>
</body>
</html>
```

Внутренние стили

Внутренний или встроенный стиль является по существу расширением для одиночного тега используемого на текущей веб-странице. Для определения стиля используется атрибут style, а его значением выступает набор стилевых правил

Использование внутреннего стиля

```
<!DOCTYPE HTML>
```

```
<html>
<head>
<meta charset="utf-8">
<title>Внутренние стили</title>
</head>
<body>
<p style="font-size: 120%; font-family: monospace; color: #cd66cc">Пример текста</p>
</body>
</html>
```

В данном примере стиль тега <p> задаётся с помощью атрибута style, в котором через точку с запятой перечисляются стилевые свойства

Внутренние стили рекомендуется применять на сайте ограниченно или вообще отказаться от их использования. Дело в том, что добавление таких стилей увеличивает общий объём файлов, что ведет к повышению времени их загрузки в браузере, и усложняет редактирование документов для разработчиков.

Все описанные методы использования CSS могут применяться как самостоятельно, так и в сочетании друг с другом. В этом случае необходимо помнить об их иерархии. Первым имеет приоритет внутренний стиль, затем глобальный стиль и в последнюю очередь связанный стиль.

В примере применяется сразу два метода добавления стиля в документ.

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Подключение стиля</title>
<style>
H1 {
font-size: 120%;
font-family: Arial, Helvetica, sans-serif;
color: green;
}
</style>
</head>
<body>
<h1 style="font-size: 36px; font-family: Times, serif; color: red">Заголовок 1</h1>
<h1>Заголовок 2</h1>
</body>
</html>
```

В данном примере первый заголовок задаётся красным цветом размером 36 пикселов с помощью внутреннего стиля, а следующий — зелёным цветом через таблицу глобальных стилей

Синтаксис CSS:

Как уже было отмечено ранее, стилевые правила записываются в своём формате, отличном от HTML. Основным понятием выступает селектор – это некоторое имя стиля, для которого добавляются параметры форматирования. В качестве селектора выступают теги, классы и идентификаторы. Общий способ записи имеет следующий вид (рисунок 1).

Рисунок 1 – Способ записи селектора, класса и идентификатора

Вначале пишется имя селектора, например, TABLE, это означает, что все стилевые параметры будут применяться к тегу <table>, затем идут фигурные скобки, в которых записывается стилевое свойство, а его значение указывается после двоеточия. Стилевые свойства разделяются между собой точкой с запятой, в конце этот символ можно опустить.

CSS не чувствителен к регистру, переносу строк, пробелам и символам табуляции, поэтому форма записи зависит от желания разработчика.

Для селектора допускается добавлять каждое стилевое свойство и его значение по отдельности.

Пример. Расширенная форма записи

td { background: olive; }

td { color: white; }

td { border: 1px; }

Однако такая запись не очень удобна. Приходится повторять несколько раз один и тот же селектор, да и легко запутаться в их количестве. Поэтому пишите все свойства для каждого селектора вместе. Указанный набор записей в таком случае получит следующий вид.

Пример: Компактная форма записи

td {

background: olive;

color: white;

border: 1px;

}

Эта форма записи более наглядная и удобная в использовании.

Если для селектора вначале задаётся свойство с одним значением, а затем то же свойство, но уже с другим значением, то применяться будет то значение, которое в коде установлено ниже.

Пример Разные значения у одного свойства

p { color: green; }

p { color: red; }

В данном примере для селектора p цвет текста вначале установлен зелёным, а затем красным. Поскольку значение red расположено ниже, то оно в итоге и будет применяться к тексту.

На самом деле такой записи лучше вообще избегать и удалять повторяющиеся значения. Но подобное может произойти случайно, например, в случае подключения разных стилевых файлов, в которых содержатся одинаковые селекторы.

У каждого свойства может быть только соответствующее его функции значение. Например, для color, который устанавливает цвет текста, в качестве значений недопустимо использовать числа.

Комментарии

Комментарии нужны, чтобы делать пояснения по поводу использования того или иного стилевого свойства, выделять разделы или писать свои заметки. Комментарии позволяют легко вспоминать логику и структуру селекторов, и повышают разборчивость кода.

Вместе с тем, добавление текста увеличивает объём документов, что отрицательно сказывается на времени их загрузки. Поэтому комментарии обычно применяют в отладочных или учебных целях, а при выкладывании сайта в сеть их стирают.

Чтобы пометить, что текст является комментарием, применяют следующую конструкцию /* ... */ .

Пример Комментарии в CSS-файле

```
/*
Стиль для сайта htmlbook.ru
Сделан для ознакомительных целей
*/
div {
width: 200px; /* Ширина блока */
margin: 10px; /* Поля вокруг элемента */
float: left; /* Обтекание по правому краю */
}
```

Как следует из данного примера, комментарии можно добавлять в любое место CSS-документа, а также писать текст комментария в несколько строк. Вложенные комментарии недопустимы.

Классы

Классы применяют, когда необходимо определить стиль для индивидуального элемента веб-страницы или задать разные стили для одного тега. При использовании совместно с тегами синтаксис для классов будет следующий.

Тег.Имя класса { свойство1: значение; свойство2: значение; ... }

Внутри стиля вначале пишется желаемый тег, а затем, через точку пользовательское имя класса. Имена классов должны начинаться с латинского символа и могут содержать в себе символ дефиса (-) и подчеркивания (_). Использование русских букв в именах классов недопустимо. Чтобы указать в коде HTML, что тег используется с определённым классом, к тегу добавляется атрибут class="Имя класса"

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Классы</title>
<style>
P { /* Обычный абзац */
text-align: justify; /* Выравнивание текста по ширине */
}
P.cite { /* Абзац с классом cite */
color: navy; /* Цвет текста */
margin-left: 20px; /* Отступ слева */
border-left: 1px solid navy; /* Граница слева от текста */
padding-left: 15px; /* Расстояние от линии до текста */
}
</style>
</head>
<body>
<p> </p>
<p class="cite"> </p>
</body>
</html>
```

Тема 2.3 Расширяемый язык разметки XML

XML (eXtensible Markup Language) – это язык для текстового выражения информации в стандартном виде. Сам по себе он не имеет операторов и не выполняет никаких вычислений. Таким образом, XML – это метаязык, главной задачей которого есть описание новых языков документа.

Одной из самых важных особенностей XML-документов является то, что их можно легко обрабатывать программно. При этом полностью сохраняется возможность визуального представления документа. Для этого достаточно лишь определить, как будет выглядеть тот или иной элемент.

Таким образом, XML позволяет отделять данные от их представления и создавать в текстовом виде документы со структурой, указанной явным образом, а именно:

1. Явным образом выделять в XML-документе структуру, что в свою очередь сделало возможным дальнейшую программную обработку документа, например, при помощи технологии XSLT. При этом одной из главных особенностей является то, что данный документ по-прежнему остается понятным обычному человеку.

2. Отделять данные в XML-документе от того, каким образом они должны быть представлены визуально. Это в свою очередь дало широкие возможности для публикации данных на разных носителях, например, на бумаге или в сети интернет.

XML-документ очень похож на обычную HTML-страницу. Также, как и в HTML, инструкции, заключенные в угловые скобки называются тэгами и служат для разметки основного текста документа. В XML существуют открывающие, закрывающие и пустые тэги (в HTML понятие пустого тэга тоже существует, но специального его обозначения не требуется).

Тело документа XML состоит из элементов разметки (markup) и непосредственно содержимого документа - данных (content). XML - тэги предназначены для определения элементов документа, их атрибутов и других конструкций языка.

Правила создания XML- 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 (пробелы, переводы строк, табуляции не игнорируются, как в HTML)

Если XML-документ не нарушает приведенные правила, то он называется формально-правильным и все анализаторы, предназначенные для разбора XML- документов, смогут работать с ним корректно.

```
<?xml version="1.0" encoding="utf-8"?>
<pricelist>
<book id="1">
```

<title>Книга 1</title>
<author>Автор 1</author>
<price>Цена 1</price>
</book>
<book id="2">
<title>Книга 2</title>
<author>Автор 2</author>
<price>Цена 2</price>
</book>
<book id="3">
<title>Книга 3</title>
<author>Автор 3</author>
<price>Цена 3</price>
</book>
</pricelist>

Для того, чтобы обеспечить проверку корректности XML-документов, необходимо использовать анализаторы, производящие такую проверку и называемые верифицирующими. На сегодняшний день существует два способа контроля правильности XML-документа: DTD - определения (Document Type Definition) и схемы данных (Semantic Schema).

Конструкции языка

Содержимое XML- документа представляет собой набор элементов, секций CDATA, директив анализатора, комментариев, спецсимволов, текстовых данных.

Элементы данных

Элемент - это структурная единица XML- документа. Заключая слово rose в тэги (<flower>rose</flower>), мы определяем непустой элемент, 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>Minsk</city>
а эти - нет:
<rose>
<flower>
rose

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 со множеством возможных связей между элементами. Например, в следующем примере мы описываем месторасположение Минских высших учебных заведений (указываем, что Белорусский государственный университет расположен в городе Минске), используя для этого вложенность элементов XML:

<country id="Belarus">
<cities-list>
<city>
<title>Минск </title>
<universities-list>
<university id="1">
<title>Белорусская государственная академия связи </title>
<address URL=www.bsac.by />
</university>
<university id="2">
<title> Белорусский государственный университет </title>
<address URL=www.bsu.by />
</university>
</universities-list>
</city>
</cities-list>
</country>

Производя в последствии поиск в этом документе, программа клиента будет опираться на информацию, заложенную в его структуру - используя элементы документа. Т.е. 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 <river>Lena</river> мы можем догадаться, что речь в этой части документа идет о реке, а вот во фрагменте <name>Lena</name> - о имени.

В случае, если элемент не имеет содержимого, т.е. нет данных, которые он должен определять, он называется пустым. Примером пустых элементов в HTML могут служить такие тэги HTML, как
, <hr>. 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 />;).

Комментарии

Комментариями является любая область данных, заключенная между последовательностями символов <!-- и --> Комментарии пропускаются анализатором и поэтому при разборе структуры документа в качестве значащей информации не рассматриваются.

<!--- <test pattern="SECAM" /><test pattern="NTSC" /> -->

Атрибуты

Если при определении элементов необходимо задать какие-либо параметры, уточняющие его характеристики, то имеется возможность использовать атрибуты эдлемента. Атрибут - это пара "название" = "значение", которую надо задавать при определении элемента в начальном тэге. Пример:

<color RGB="true">#ff08ff</color>

<color RGB="false">white</color>

Примером использования атрибутов в HTML является описание элемента :

Black

Специальные символы

Для того, чтобы включить в документ символ, используемый для определения каких-либо конструкций языка (например, символ угловой скобки) и не вызвать при этом ошибок в процессе разбора такого документа, нужно использовать его специальный символьный либо числовой идентификатор. Например, < , > " или $(десятичная форма записи), &#x1a (шестнадцатеричная) и т.д.

Директивы анализатора

Инструкции, предназначенные для анализаторов языка, описываются в XML документе при помощи специальных тэгов - ; . Программа клиента использует эти инструкции для управления процессом разбора документа. Наиболее часто инструкции используются при определении типа документа или создании пространства имен.

CDATA

В XML документах фрагмент, помещённый внутрь CDATA, — это часть содержания элемента, которая помечена для парсера как содержащая только символьные данные, а не разметку. CDATA — это просто альтернативный синтаксис для отображения символьных данных, нет никакой смысловой разницы между символьными данными, которые объявлены как CDATA и символьными данными, которые объявлены в обычном синтаксисе и где «<» и «>» будут представлены как «<» и «>», соответственно.

Синтаксис и интерпретация

Раздел CDATA начинается со следующей последовательности символов:

<![CDATA[

и заканчивается с первым появлением последовательности:

]]>

Все символы, заключённые между этими двумя последовательностями, интерпретируются как символы, а не как разметка или ссылки на объект. Например, в этой строке: <sender>John Smith</sender> открывающий и закрывающий теги «sender» будут интерпретированы как разметка. Однако, если записать её вот так:

<![CDATA[<sender>John Smith</sender>]]>

то этот код будет интерпретирован так же, как если бы было записано:

<sender>John Smith</sender>

Таким образом, теги sender будут восприниматься так же, как «John Smith», то есть как текст.

Аналогично, если в содержимом элемента появляется цифровая последовательность ð, это будет интерпретировано как простой символ Юникода 00F0. Но если эта последовательность появится в разделе CDATA, она будет разделена на 6 символов: амперсанд, знак октоторпа, цифру 2, цифру 4, цифру 0 и точку с запятой.

Тема 2.4 CMS как способ создания web-страниц

CMS (Content Management System) - система управления содержимым (контентом) - компьютерная программа или информационная система, которая используется для организации и обеспечения процесса по совместному созданию, управлению и редактированию содержимого сайта.

Если смотреть с точки зрения обычного заказчика, то разработка сайта на основе какой-либо CMS должна приносить следующие преимущества:

– в работе используется наиболее эффективный инструмент для решения конкретной задачи (в зависимости от вида сайта и требований к его функционалу подбирают оптимальную CMS);

– использование CMS позволяет владельцу сайта самостоятельно создавать и удалять разделы сайта, редактировать различную информацию без привлечения стороннего специалиста - это одно из преимуществ над статическими сайтами;

– работа сайта постоянно тестируется множеством пользователем, а найденные ошибки и уязвимости достаточно оперативно устраняются, при этом сайт работает на самых передовых и проверенных технических решениях;

– временные затраты на разработку сайта существенно снижаются, так как разработчику не надо фиксировать своё внимание на чисто технических задачах: «как сделать ленту с

новостями» или «как научить CMS искать товары в каталоге», а можно сосредоточиться на информационной и визуальной составляющих будущего сайта;

До сих пор нет единой и чёткой классификации, принятой рынком, существующих CMS. Некоторые из наиболее используемых:

– Joomla - (самая противоречивая система) - яркая, современная, постоянно обновляющаяся, достаточно простая в разработке и использовании, предоставляется совершенно бесплатно;

– Drupal - (для любителей разработки сайтов с нуля) - это полноценная функциональная среда для разработки и создания всевозможных сайтов, которая предоставляется так же бесплатно и имеет гибкие возможности;

– 1С Битрикс - разработка от 1С, в которой можно найти различные редакции от простой, до портальной, при этом стоимость её значительно отличается от версии к версии;

– и многие-многие другие: WordPress, Koobi, MediaWiki, CM5, NetCat…

РАЗДЕЛ 3 ТЕХНОЛОГИИ WEB-ПРОГРАММИРОВАНИЯ

Тема 3.1 JavaScript

JavaScript – это полноценный язык программирования, который применяется к HTML-документу, и может обеспечить динамическую интерактивность на веб-сайтах.

JavaScript изначально создавался для того, чтобы сделать web-странички «живыми». Программы на этом языке называются *скриптами*. В браузере они подключаются напрямую к HTML и, как только загружается страничка – тут же выполняются.

Особенности:

– JavaScript дает HTML дизайнерам инструмент программирования - авторы HTML обычно не являются программистами, но JavaScript это язык сценариев с очень простым синтаксисом! Почти каждый может вставить небольшие "куски" кода в их HTML страницы;

– JavaScript реагирует на события - JavaScript может быть настроен на выполнение определенных действий, когда происходит нечто, например когда пользователь щелкает мышью на HTML элементе;

– JavaScript может читать и писать HTML элементы - JavaScript может читать и изменять содержимое HTML элемента;

– JavaScript может использоваться для проверки данных - JavaScript может использоваться для проверки данных, введенных в поля формы, прежде чем они будут переданы на сервер. Это предохраняет сервер от излишней обработки;

– JavaScript может использоваться для определения браузера посетителя - JavaScript может использоваться для определения браузера пользователя, и - в зависимости от того, какой браузер, - загружать другую страницу, спроектированную специально для этого браузера;

– JavaScript может использоваться для создание cookies - JavaScript может быть использован для сохранения и загрузки информации с пользовательского компьютера.

Комментарии

Комментарии добавляются для пояснения кода JavaScript, или чтобы сделать код более читабельным.

– однострочные комментарии начинаются с //;

– многострочные комментарии начинаются с /* и заканчиваются */.

Переменная может иметь короткое имя, например x, или более информативное имя, например carname (название автомобиля).

Правила для имен переменных JavaScript:

– имена переменных чувствительны к регистру (так как и JavaScript);

– имена переменных должны начинаться с буквы или символа подчеркивания.

Объявление переменных JavaScript происходит с помощью ключевого слова var: var name= «ТО»;

Арифметические операторы используются для выполнения арифметических операций между переменными и/или значениями. (таблица 1)

Таблица 1 – Арифметические операторы

+	Сложение	x=y+2	x=7	y=5
-	Вычитание	x=y-2	x=3	y=5
*	Умножение	x=y*2	x=10	y=5
/	Деление	x=y/2	x=2.5	y=5
%	Деление по модулю (остаток от деления)	x=y%2	x=1	y=5
++	Инкремент	x=++y	x=6	y=6
		x=y++	x=5	y=6
--	Декремент	x=--y	x=4	y=4
		x=y--	x=5	y=4

Операторы присваивания используются для присваивания значений переменным JavaScript.

Пусть x=10 и y=5, таблица 2 ниже объясняет операторы присваивания:

Таблица 2 – Операторы присваивания

Оператор	**Пример**	**Эквивалентная операция**	**Результат**
=	x=y		x=5
+=	x+=y	x=x+y	x=15
-=	x-=y	x=x-y	x=5
*=	x*=y	x=x*y	x=50
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

Оператор + также может использоваться, чтобы соединять строковые переменные или текстовые значения друг с другом. Если складывается число и строку, результатом будет строка.

```
txt1="ТО";
txt2="513";
txt3=txt1+txt2;
```

Операторы сравнения используются в логических предложениях для определения равенства или неравенства между переменными или значениями.

Пусть x=5, таблица 3 ниже объясняет операторы сравнения:

Таблица 3 – Операторы сравнения

Оператор	**Описание**	**Пример**
==	равно	x==8 это ложь x==5 это истина
===	точно равно (значение и тип совпадают)	x===5 это истина x==="5" это ложь
!=	не равно	x!=8 это истина
>	больше чем	x>8 это ложь
<	меньше чем	x<8 это истина
>=	больше или равно	x>=8 это ложь
<=	меньше или равно	x<=8 это истина

Логические операторы используются для определения логических отношений между переменными или значениями.

Пусть x=6 и y=3, таблица 4 ниже объясняет логические операторы:

Таблица 4 – Логические операторы

Оператор	**Описание**	**Пример**
&&	логическое И	(x < 10 && y > 1) это истина
||	логическое ИЛИ	(x==5 || y==5) это ложь
!	логическое НЕ	!(x==y) это истина

JavaScript также имеет условный оператор **?**, который присваивает значение переменной на основе некоторого условия.

имя_переменной=(условие)?значение1:значение2

В JavaScript используются следующие условные предложения:

– конструкция **if** - используется, чтобы выполнить некоторый код только в том случае, когда указанное условие истинно

```
if (условие)
{
код для выполнения, если условие истинно
}
```

– конструкция if...else - используется для выполнения одного фрагмента кода, если условие истинно, и другого, если условие ложно

```
if …
    else
                                                                    {
    код, который будет выполнен, если условие ложно
 }
```

– конструкция

– - используется для выбора одного из многих блоков кода для выполнения

```
    switch(n)
{
case 1:
        выполнить блок кода 1
                                                                    break;
case 2:
        выполнить блок кода 2
                                                                    break;
default:
    исполняемый код, если n не равно 1 или 2
}
```

JavaScript имеет три вида всплывающих окон: Сигнальное окно, окно Подтверждения, и окно Приглашения.

Сигнальное окно часто используется, когда необходимо гарантированно доставить информацию до пользователя. Когда сигнальное окно выскакивает, пользователь должен кликнуть "OK", чтобы продолжить.

alert("некоторый текст");

Окно подтверждения обычно используется, если необходимо, чтобы пользователь проверил или соглисился с чем-то. Когда окно подтверждения выскакивает, пользователь должен кликнуть либо "OK" либо "Отмена", чтобы продолжить. Если пользователь кликает на "OK", окно возвращает true (истина). Если пользователь щелкает на "Отмена", окно возвращает false (ложь).

confirm("некоторый текст");

Окно приглашения обычно используется, когда необходимо, чтобы пользователь ввел значение перед тем как войти на страницу. Когда окно приглашения выскакивает, пользователь должен нажать либо "OK" либо "Отмена" чтобы продолжить после ввода значения. Если пользователь нажимает "OK" окно возвращает введенное значение. Если же пользователь нажимает "Отмена" окно возвратит null (означает пустое значение, или отсутствие значения).

prompt("некоторый_текст","значение_по_умолчанию");

Чтобы предотвратить скрипт от выполнения браузером, когда страница загружается, необходимо поместить скрипт в функцию.

Функция содержит код, который будет выполнен по наступлении события или при вызове этой функции. Вызвать функцию можно из любого места страницы (или даже из других страниц, если функция находится во внешнем .js файле).

Функции могут быть определены как в секции <head>, так и в секции <body> документа. Однако, чтобы гарантировать, что функция будет прочитана/загружена браузером прежде, чем она вызывается, разместите функции в секцию <head>.

function имя_функции(var1,var2,...,varX)

{

некоторый код

}

Параметры var1, var2, и т.д. это переменные или значения, передаваемые функции. Скобки { и } определяют начало и конец функции.

Функция будет выполнена при возникновении события или при вызове функции.

Предложение return используется для указания значения, которое возвращается из функции. Таким образом, функции, которые возвращают значения, должны использовать предложение return.

Если объявить переменную, используя "var", внутри функции, переменная будет доступна только в пределах этой функции. При выходе из функции, переменная уничтожается. Эти переменные называются локальными переменными. Можно объявлять локальные переменные с одним и тем же именем в различных функциях, поскольку каждая из них распознается только в функции, где была объявлена.

Если объявить переменную вне функции, все функции на странице могут к ней обращаться. Время жизни этих переменных начинается, когда они объявляются, и заканчивается, когда страница закрывается.

Циклы выполняют блок кода указанное количество раз, или пока определенное условие является истинным.

В JavaScript существует два типа циклов:

– for - выполняет блок кода указанное количество раз

for (переменная=начальное_значение; переменная<=конечное_значение; шаг)
{
исполняемый код
}

– while - выполняет блок кода, пока указанное условие истинно

while (переменная<=конечное значение)
{
исполняемый код
}

Ключевое слово break прерывает цикл и переходит к выполнению кода, который следует после цикла (если такой код имеется).

Ключевое слово continue прерывает текущую итерацию цикла и переходит к следующей итерации.

Конструкция try ... catch позволяет проверять блок кода на наличие ошибок. Блок try содержит код, который будет исполняться и проверяться, а блок catch содержит код, который будет выполнен при возникновении ошибок.

```
try
{
    //Запустить некоторый код здесь
}
catch(err)
{
    //Обработка ошибок здесь
}
```

Можно переносить строку кода внутри текстовой строки с помощью обратного слэша. Пример ниже будет отображаться корректно:

```
document.write("Привет, \
ТО!");
```

Однако, вы не можете переносить строку кода так:

```
document.write \
("Привет, ТО");
```

Как правило, каждая команда пишется на отдельной строке – так код лучше читается:

```
alert('Привет');
alert('Мир');
```

Точку с запятой во многих случаях можно не ставить, если есть переход на новую строку. Так тоже будет работать:

```
alert('Привет')
```

```
alert('Мир')
```

В этом случае JavaScript интерпретирует переход на новую строчку как разделитель команд и автоматически вставляет «виртуальную» точку с запятой между ними.

Однако, важно то, что «во многих случаях» не означает «всегда»!

Например, запустите этот код:

```
alert(3 +
1
+ 2);
```

Выведет 6.

То есть, точка с запятой не ставится. Почему? Интуитивно понятно, что здесь дело в «незавершённом выражении», конца которого JavaScript ждёт с первой строки и поэтому не ставит точку с запятой. И здесь это, пожалуй, хорошо и приятно.

Но в некоторых важных ситуациях JavaScript «забывает» вставить точку с запятой там, где она нужна. Таких ситуаций не так много, но ошибки, которые при этом появляются, достаточно сложно обнаруживать и исправлять.

Такой код работает:

```
[1, 2].forEach(alert)
```

Он выводит по очереди 1, 2. Почему он работает – сейчас не важно, позже разберёмся. Важно, что вот такой код уже работать не будет:

```
alert("Сейчас будет ошибка")
[1, 2].forEach(alert)
```

Выведется только первый alert, а дальше – ошибка. Потому что перед квадратной скобкой JavaScript точку с запятой не ставит, а как раз здесь она нужна. Если её поставить, то всё будет в порядке.

Поэтому в JavaScript рекомендуется точки с запятой ставить. Сейчас это, фактически, стандарт, которому следуют все большие проекты.

Если JavaScript-кода много – его выносят в отдельный файл, который подключается в HTML:

```
<script src="…/script.js"></script>
```

Как правило, в HTML пишут только самые простые скрипты, а сложные выносят в отдельный файл. Браузер скачает его только первый раз и в дальнейшем, при правильной настройке сервера, будет брать из своего кеша.

Благодаря этому один и тот же большой скрипт, содержащий, к примеру, библиотеку функций, может использоваться на разных страницах без полной перезагрузки с сервера.

Если указан атрибут src, то содержимое тега игнорируется. В одном теге SCRIPT нельзя одновременно подключить внешний скрипт и указать код.

Вот так не сработает:

```
<script src="file.js">
  alert(1); // так как указан src, то внутренняя часть тега игнорируется
</script>
```

Нужно выбрать: либо SCRIPT идёт с src, либо содержит код. Тег выше следует разбить на два: один – с src, другой – с кодом, вот так:

```
<script src="file.js"></script>
<script>
  alert( 1 );
</script>
```

Браузер загружает и отображает HTML постепенно. Особенно это заметно при медленном интернет-соединении: браузер не ждёт, пока страница загрузится целиком, а показывает ту часть, которую успел загрузить. Если браузер видит тег <script>, то он по стандарту обязан сначала выполнить его, а потом показать оставшуюся часть страницы.

Такое поведение называют «синхронным». Как правило, оно вполне нормально, но есть важное следствие. Если скрипт – внешний, то пока браузер не выполнит его, он не покажет часть страницы под ним.

То есть, в таком документе, пока не загрузится и не выполнится big.js, содержимое <body> будет скрыто:

```
<html>
<head>
<script src="big.js"></script>
</head>
<body>
Этот текст не будет показан, пока браузер не выполнит big.js.
</body>
</html>
```

И здесь вопрос – действительно ли мы этого хотим? То есть, действительно ли оставшуюся часть страницы нельзя показывать до загрузки скрипта? Есть ситуации, когда мы не только НЕ хотим такой задержки, но она даже опасна.

Например, если мы подключаем внешний скрипт, который показывает рекламу или вставляет счётчик посещений, а затем идёт наша страница. Конечно, неправильно, что пока счётчик или реклама не подгрузятся – оставшаяся часть страницы не показывается. Счётчик посещений не должен никак задерживать отображение страницы сайта. Реклама тоже не должна тормозить сайт и нарушать его функциональность.

А что, если сервер, с которого загружается внешний скрипт, перегружен? Посетитель в этом случае может ждать очень долго!

Можно поставить все подобные скрипты в конец страницы – это уменьшит проблему, но не избавит от неё полностью, если скриптов несколько. Допустим, в конце страницы 3 скрипта, и первый из них тормозит – получается, другие два его будут ждать – тоже нехорошо.

Кроме того, браузер дойдёт до скриптов, расположенных в конце страницы, они начнут грузиться только тогда, когда вся страница загрузится. А это не всегда правильно. Например, счётчик посещений наиболее точно сработает, если загрузить его пораньше.

Поэтому «расположить скрипты внизу» – не лучший выход. Кардинально решить эту проблему помогут атрибуты async или defer:

Атрибут async. Поддерживается всеми браузерами, кроме IE9-. Скрипт выполняется полностью асинхронно. То есть, при обнаружении <script async src="..."> браузер не останавливает обработку страницы, а спокойно работает дальше. Когда скрипт будет загружен – он выполнится.

Атрибут defer. Поддерживается всеми браузерами, включая самые старые IE. Скрипт также выполняется асинхронно, не заставляет ждать страницу, но есть два отличия от async. Первое – браузер гарантирует, что относительный порядок скриптов с defer будет сохранён.

То есть, в таком коде (с async) первым сработает тот скрипт, который раньше загрузится:

<script src="1.js" async></script>

<script src="2.js" async></script>

А в таком коде (с defer) первым сработает всегда 1.js, а скрипт 2.js, даже если загрузился раньше, будет его ждать.

<script src="1.js" defer></script>

<script src="2.js" defer></script>

Поэтому атрибут defer используют в тех случаях, когда второй скрипт 2.js зависит от первого 1.js, к примеру – использует что-то, описанное первым скриптом.

Второе отличие – скрипт с defer сработает, когда весь HTML-документ будет обработан браузером. Например, если документ достаточно большой, то скрипт async.js выполнится, как только загрузится – возможно, до того, как весь документ готов. А defer.jsподождёт готовности всего документа.

Это бывает удобно, когда мы в скрипте хотим работать с документом, и должны быть уверены, что он полностью получен. При одновременном указании async и defer в современных браузерах будет использован только async, в IE9- – только defer (не понимает async).

В JavaScript существует несколько основных типов данных.

Число «number»

var n = 123;

n = 12.345;

Единый тип число используется как для целых, так и для дробных чисел.

Существуют специальные числовые значения Infinity (бесконечность) и NaN (ошибка вычислений).

Например, бесконечность Infinity получается при делении на ноль:

alert(1 / 0); // Infinity

Ошибка вычислений NaN будет результатом некорректной математической операции, например:

alert("нечисло" * 2); // NaN, ошибка

Эти значения формально принадлежат типу «число», хотя, конечно, числами в их обычном понимании не являются.

Строка “string”

В JavaScript одинарные и двойные кавычки равноправны. Можно использовать или те или другие.

var str = "Мама мыла раму";

str = 'Одинарные кавычки тоже подойдут';

Тип символ не существует, есть только строка.

В некоторых языках программирования есть специальный тип данных для одного символа. Например, в языке C это char. В JavaScript есть только тип «строка» string.

Логический «boolean»

У него всего два значения: true (истина) и false (ложь). Как правило, такой тип используется для хранения значения типа да/нет, например:

ar checked = true; // поле формы помечено галочкой

checked = false; // поле формы не содержит галочки

Значение null не относится ни к одному из типов выше, а образует свой отдельный тип, состоящий из единственного значения null:

var age = null;

В JavaScript null не является «ссылкой на несуществующий объект» или «нулевым указателем», как в некоторых других языках. Это просто специальное значение, которое имеет смысл «ничего» или «значение неизвестно».

В частности, код выше говорит о том, что возраст age неизвестен.

Значение undefined, как и null, образует свой собственный тип, состоящий из одного этого значения. Оно имеет смысл «значение не присвоено».

Если переменная объявлена, но в неё ничего не записано, то её значение как раз и есть undefined:

var x;

alert(x); // выведет "undefined"

В явном виде undefined обычно не присваивают, так как это противоречит его смыслу. Для записи в переменную «пустого» или «неизвестного» значения используется null.

Первые 5 типов называют «примитивными». Особняком стоит шестой тип: «объекты».Он используется для коллекций данных и для объявления более сложных сущностей.

Объявляются объекты при помощи фигурных скобок {...}, например:

var user = { name: "Вася" };

Оператор typeof возвращает тип аргумента. У него есть два синтаксиса: со скобками и без:

Синтаксис оператора: typeof x.

Синтаксис функции: typeof(x).

Работают они одинаково, но первый синтаксис короче.

Результатом typeof является строка, содержащая тип:

typeof undefined // "undefined"

typeof 0 // "number"

typeof true // "boolean"

typeof "foo" // "string"

typeof {} // "object"

Есть 5 «примитивных» типов: number, string, boolean, null, undefined и 6-й тип – объекты object.

Тема 3.2 PHP

PHP (англ. PHP: Hypertext Preprocessor) – скриптовый язык программирования общего назначения, интенсивно применяемый для разработки веб-приложений. В настоящее время поддерживается подавляющим большинством хостинг-провайдеров и является одним из лидеров среди языков программирования, применяющихся для создания динамических веб-сайтов.

Синтаксис переменной состоит из знака доллара - $ и "свободного" идентификатора которому присваивается какое-нибудь значение. Например:

```
<?php
$name = "Виктор";
?>
```

PHP является языком динамической типизации, тип данных переменной определяется на основе её значения. Ниже перечислены все типы, которые можно использовать в PHP:

- Boolean. Это логический тип, который содержит значение true или false.
- Integer. Содержит значения целого числа (Например: 4 или 10 или другое целое число).
- String. Содержит значение текста произвольной длины
- Float. Вещественное число (Например: 1.2, 3.14, 8.5498777).
- Object. Объект.
- Array. Массив.

– Resource. Ресурс (Например: файл).

– NULL. Значение NULL.

Для вывода данных на экран используется оператор echo.Все переменные в PHP начинаются со знака $. Имена переменных чувствительны к регистру. Тип данных определяется автоматически. Операторы аналогичны операторам языков C, JavaScript.

В PHP существует 3 типа комментариев.

Первый позволяет размещать комментарии в нескольких строках. Начинается такой тип комментариев с символов /* и заканчиваются */, например:

```
<?php
/* Тут может быть размещен любой текст,
даже в несколько строк */
?>
```

Следует иметь ввиду, что вложенные комментарии не допустимы. Такой код вызовет ошибку:

```
<?php
/* Тут может быть размещен любой текст,
/*даже в */
несколько строк */
?>
```

Следующие два типа являются однострочными. Такие комментарии начинаются с символов // или # и продолжаются до конца строки. Пример:

```
<?php
// Тут может быть размещен любой текст
# Только в одной строке !
echo "Привет Всем !";
?>
```

Функции в PHP. Формат описания функции пользователя:

```
function имя_функции(параметры){
тело функции
}
```

В языке PHP разработано множество библиотек встроенных функций. Например:

– phpinfo() – выводит на экран параметры конфигурации веб-сервера;

– count(array arr) вычисляет размер массива;

– sort(array arr) – сортирует массив по возрастанию;

– print_r(array arr) – выводит на экран массив;

– array_rand(array arr) – выполняет сортировку в случайном порядке элементов массива. Необязательный параметр num_req задает количество сортируемых элементов в массиве;

– in_array(mixed element, array arr) – возвращает true, если значение element найдено в массиве arr, и false иначе;

– strtoupper(string $str) – приводит str к верхнему регистру;

– strtolower(string $str) – приводит str к нижнему регистру;

– substr(string $str, int start, int length) – возвращает часть строки str, начиная с позиции start длиной length;

– string str_replace(string from, string to, string str) – заменяет в строке str все вхождения from на to;

– strlen(string $str) – возвращает длину строки;

– explode(string separator, string str) – делит строку str по разделителю separator. Результат возвращает в виде массива;

– ltrim(string $str) – удаляет начальные пробелы из строки;

– rtrim(string $str) – удаляет конечные пробелы из строки;

– trim(string $str) – удаляет начальные и конечные пробелы из строки;

– ord(string $str) – возвращает ASCII–код символа;

– time() – возвращает количество секунд, прошедших с 00:00:00 1.01.1970;

– mktime() – формирует временную метку;

– getdate() – возвращает ассоциативный массив с ключами: seconds, minutes, hours, mday, wday, mon, year,yday,weekday,month,0;

– date() – позволяет форматировать дату

Тема 3.3 Основы объектно-ориентированного программирования

PHP до недавнего времени обеспечивал лишь некоторую поддержку ООП. Однако, после выхода PHP5 поддержка ООП в PHP стала практически полной.

Стратегию ООП лучше всего описать как смещение приоритетов в процессе программирования от функциональности приложения к структурам данных. Это позволяет программисту моделировать в создаваемых приложениях реальные объекты и ситуации. Технология ООП обладает тремя главными преимуществами:

– она проста для понимания: ООП позволяет мыслить категориями повседневных объектов;

– повышено надежна и проста для сопровождения — правильное проектирование обеспечивает простоту расширения и модификации объектно-ориентированных программ. Модульная структура позволяет вносить независимые изменения в разные части программы, сводя к минимуму риск ошибок программирования;

– ускоряет цикл разработки – модульность и здесь играет важную роль, поскольку различные компоненты объектно-ориентированных программ можно легко использовать в

других программах, что уменьшает избыточность кода и снижает риск внесения ошибок при копировании.

Специфика ООП заметно повышает эффективность труда программистов и позволяет им создавать более мощные, масштабируемые и эффективные приложения.

Объектно-ориентированное программирование основано на: инкапсуляции, полиморфизме и наследовании.

Инкапсуляция - это механизм, объединяющий данные и обрабатывающий их код как единое целое.

Многие преимущества ООП обусловлены одним из его фундаментальных принципов — инкапсуляцией. Инкапсуляцией называется включение различных мелких элементов в более крупный объект, в результате чего программист работает непосредственно с этим объектом. Это приводит к упрощению программы, поскольку из нее исключаются второстепенные детали.

Инкапсуляцию можно сравнить с работой автомобиля с точки зрения типичного водителя. Многие водители не разбираются в подробностях внутреннего устройства машины, но при этом управляют ею именно так, как было задумано. Пусть они не знают, как устроен двигатель, тормоз или рулевое управление, — существует специальный интерфейс, который автоматизирует и упрощает эти сложные операции. Сказанное также относится к инкапсуляции и ООП — многие подробности "внутреннего устройства" скрываются от пользователя, что позволяет ему сосредоточиться на решении конкретных задач. В ООП эта возможность обеспечивается классами, объектами и различными средствами выражения иерархических связей между ними.

Полиморфизм позволяет использовать одни и те же имена для похожих, но технически разных задач. Главным в полиморфизме является то, что он позволяет манипулировать объектами путем создания стандартных интерфейсов для схожих действий. Полиморфизм значительно облегчает написание сложных программ.

Наследование позволяет одному объекту приобретать свойства другого объекта. При копировании создается точная копия объекта, а при наследовании точная копия дополняется уникальными свойствами, которые характерны только для производного объекта.

Классы и объекты в PHP

Класс - это базовое понятие в объектно-ориентированном программировании (ООП). Классы образуют синтаксическую базу ООП.

Их можно рассматривать как своего рода «контейнеры» для логически связанных данных и функций. Если сказать проще, то класс – это своеобразный тип данных.

Экземпляр класса – это объект. Объект – это совокупность данных(свойств) и функций(методов) для их обработки. Свойства и методы называются членами класса.

Если класс можно рассматривать как тип данных, то объект — как переменную (по аналогии). Скрипт может одновременно работать с несколькими объектами одного класса, как с несколькими переменными.

Внутри объекта данные и код (члены класса) могут быть либо открыты, либо нет. Открытые данные и члены класса являются доступными для других частей программы, которые не являются частью объекта. А вот закрытые данные и члены класса доступны только внутри этого объекта.

Описание классов в PHP начинаются служебным словом class:

```
class Имя_класса {
// описание членов класса - свойств и методов для их обработки
}
```

Для объявления объекта необходимо использовать оператор new:

```
Объект = new Имя_класса;
```

Данные описываются с помощью служебного слова var. Метод описывается так же, как и обыкновенная пользовательская функция. Методу также можно передавать параметры.

По общепринятым правилам имена классов ООП начинаются с прописной буквы, а все слова в именах методов, кроме первого, начинаются с прописных букв (первое слово начинается со строчной буквы).

Пример класса на PHP:

```
<?php
// Создаем новый класс Coor:
class Coor {
// данные (свойства):
var $name;
var $addr;
// методы:
 function Name() {
 echo "<h3>John</h3>";
 }
}
// Создаем объект класса Coor:
$object = new Coor;
?>
```

Для получения доступа к членам класса в PHP предназначен оператор ->. Пример:

```
<?php
// Создаем новый класс Coor:
class Coor {
```

```
// данные (свойства):
var $name;
// методы:
 function Getname() {
 echo "<h3>John</h3>";
 }
}
// Создаем объект класса Coor:
$object = new Coor;
// Получаем доступ к членам класса:
$object->name = "Alex";
echo $object->name;
// Выводит 'Alex'
// А теперь получим доступ к методу класса (фактически, к фунции внутри класса):
$object->Getname();
// Выводит 'John' заглавными буквами
?>
```

Чтобы получить доступ к членам класса внутри класса, необходимо использовать указатель $this, которы всегда относится к текущему объекту. Модифицированный метод Getname():

```
function Getname() {
echo $this->name;
}
```

Таким же образом, можно написать метод Setname():

```
function Setname($name) {
$this->name = $name;
}
```

Теперь для изменения имени можно использовать метод Setname():

```
$object->Setname("Peter");
$object->Getname();
```

А вот и полный листинг кода:

```
<?php
// Создаем новый класс Coor:
class Coor {
// данные (свойства):
var $name;
// методы:
```

```
 function Getname() {
 echo $this->name;
 }
 function Setname($name) {
 $this->name = $name;
 }
}
// Создаем объект класса Coor:
$object = new Coor;
// Теперь для изменения имени используем метод Setname():
$object->Setname("Nick");
// А для доступа, как и прежде, Getname():
$object->Getname();
// Сценарий выводит 'Nick'
?>
```

Указатель $this можно также использовать для доступа к методам, а не только для доступа к данным:

```
function Setname($name) {
$this->name = $name;
$this->Getname();
}
```

Конструкторы

Довольно часто при создании объекта требуется задать значения некоторых свойств. Конструктор представляет собой метод, который задает значения некоторых свойств (а также может вызывать другие методы). Конструкторы вызываются автоматически при создании новых объектов. Чтобы это стало возможным, имя метода-конструктора должно совпадать с именем класса, в котором он содержится. Пример конструктора:

```
<?
class Webpage {
var $bgcolor;
 function Webpage($color) {
 $this->bgcolor = $color;
 }
}
// Вызвать конструктор класса Webpage
```

```
$page = new Webpage("brown");
?>
```

Раньше создание объекта и инициализация свойств выполнялись раздельно. Конструкторы позволяют выполнить эти действия за один этап.

Деструкторы

В PHP отсутствует непосредственная поддержка деструкторов. Тем не менее, можно имитировать работу деструктора, вызывая функцию PHP unset(). Эта функция уничтожает содержимое переменной и возвращает занимаемые ею ресурсы системе. С объектами unset() работает так же, как и с переменными. После завершения работы с конкретным объектом вызывается функция, как показано ниже:

```
unset($Webpage);
```

Эта команда удаляет из памяти все содержимое $Webpage.

Необходимость в вызове деструкторов возникает лишь при работе с объектами, использующими большой объем ресурсов, поскольку все переменные и объекты автоматически уничтожаются по завершении сценария.

Инициализация объектов

Иногда возникает необходимость выполнить инициализацию объекта - присвоить его свойствам первоначальные значения. Предположим, имя класса Coor и он содержит два свойства: имя человека и город его проживания. Можно написать метод (функцию), который будет выполнять инициализацию объекта, например Init():

```
<?php
// Создаем новый класс Coor:
class Coor {
// данные (свойства):
var $name;
var $city;
// Инициализирующий метод:
 function Init($name) {
 $this->name = $name;
 $this->city = "London";
}
}
// Создаем объект класса Coor:
$object = new Coor;
// Для инициализации объекта сразу вызываем метод:
$object->Init();
?>
```

Главное не забыть вызвать функцию сразу после создания объекта. Для того, чтобы PHP знал, что определенный метод нужно вызывать автоматически при создании объекта, ему нужно дать имя такое же, как и у класса (Coor):

```
function Coor ($name)
$this->name = $name;
$this->city = "London";
}
```

Обращение к элементам классов осуществляется с помощью оператора :: "двойное двоеточие". Используя "двойное двоеточие", можно обращаться к методам классов. При обращении к методам классов, программист должен использовать имена этих классов.

```
<?php
class A {
    function example() {
        echo "Это первоначальная функция A::example().<br>";
    }
}
class B extends A {
    function example() {
        echo "Это переопределенная функция B::example().<br>";
        A::example();
    }
}
// Не нужно создавать объект класса A.
// Выводит следующее:
// Это первоначальная функция A::example().
A::example();
// Создаем объект класса B.
$b = new B;
// Выводит следующее:
//   Это переопределенная функция B::example().
//   Это первоначальная функция A::example().
$b->example();
?>
```

Тема 3.4 Web программирование на Java

Приложение Java, запускаемое и выполняемое в контейнере сервера приложений. Клиент общается с таким приложением посредством веб-браузера. Никаких дополнительных приложений на стороне клиента устанавливать не требуется. Сервлеты поддерживаются виртуальной машиной JVM, что предотвращает утечки памяти и обеспечивает функционирование garbage collection. Каждому клиенту сервлет выделяет независимый поток выполнения. Клиент посылает приложению HTTP-запрос, сервлет генерирует ответ и возвращает его клиенту в виде html-документа.

Сервлет:

– компонент приложений Java Enterprise Edition;
– загружается веб-сервером в контейнер;
– выполняется на стороне сервера;
– обрабатывает клиентские запросы;
– динамически генерирует ответы на запросы;
– находится в состоянии ожидания, если запросы отсутствуют;
– принимает запросы от других сервлетов (Servlet chaining);
– поддерживает соединения с ресурсами.

Наибольшее распространение получили сервлеты, обрабатывающие клиентские запросы по протоколу HTTP. Технология сервлетов является оболочкой протокола HTTP и поддерживает его как транспорт передачи данных от клиента серверу и обратно. Контейнер сервлетов поддерживает также протокол HTTPS (HTTP и SSL) для защищаемых запросов.

Сервлеты в промышленном программировании используются для:

– приема входящих данных от клиента;
– взаимодействия с бизнес-логикой системы;
– динамической генерации ответа клиенту.

Все сервлеты реализуют общий интерфейс Servlet из пакета javax.servlet. Для обработки HTTP-запросов в web можно воспользоваться в качестве базового класса абстрактным классом HttpServlet из пакета javax.servlet.http.

Жизненный цикл сервлета начинается с его инициализации и загрузки в память контейнером сервлетов при старте контейнера либо в ответ на первый клиентский запрос. Сервлет готов к обслуживанию любого числа запросов. Завершение существования происходит при выгрузке его из контейнера.

Первым вызывается метод init(). Он дает сервлету возможность инициализировать данные и подготовиться для обработки запросов. Чаще всего в этом методе размещается код, кэширующий данные фазы инициализации.

После этого сервлет можно считать запущенным, он находится в ожидании запросов от клиентов. Появившийся запрос обслуживается методом service(HttpServletRequest request, HttpServletResponse response) сервлета, вызываемый контейнером, а все параметры запроса упаковываются в экземпляр request интерфейса HttpServletRequest, передаваемый в сервлет. Еще одним параметром этого метода является экземпляр response интерфейса HttpServletResponse, в который загружается информация для передачи клиенту. Для каждого нового клиента при обращении к сервлету создается независимый поток, в котором производится вызов метода service(). Метод service() предназначен для одновременной обработки множества запросов.

При выгрузке приложения из контейнера, то есть по окончании жизненного цикла сервлета, вызывается метод destroy(), в теле которого следует помещать код освобождения занятых сервлетом ресурсов.

Практика включения HTML-кода в код сервлета не считается хорошей, так как эти действия «уводят» сервлет от его основной роли — контроллера приложения. Это приводит к росту объема кода сервлета, который на определенном этапе становится неконтролируемым и реализует вследствие этого антишаблон «Волшебный сервлет».

Технология Java Server Pages (JSP) обеспечивает разделение динамической и статической частей страницы, результатом чего является возможность изменения дизайна страницы, не затрагивая динамическое содержание. Это свойство используется при разработке и поддержке страниц, так как дизайнерам нет необходимости знать, как работать с динамическими данными.

JSP-код состоит из специальных тегов и выражений, которые указывают контейнеру соответствие между тегами и java-кодом для генерации сервлета или его части. Таким образом поддерживается документ, который одновременно содержит и статическую страницу, и теги Java, управляющие страницей. Статические части HTML-страниц посылаются в виде строк в метод write(). Динамические части включаются прямо в код сервлета. С момента формирования ответа сервера страница ведет себя как обычная HTML-страница с ассоциированным сервлетом.

Чтобы создать простейшую JSP, достаточно взять HTML-страницу и заменить расширение html на jsp. Только в этом случае для запуска страницы необходим специальный application server с контейнером сервлетов и особое размещение самой страницы.

Страницы JSP и сервлеты никогда не следует использовать в информационных системах друг без друга. Причиной являются принципиально различные роли, которые играют данные компоненты в приложении. Страница JSP ответственна за формирование пользовательского интерфейса и отображение информации, переданной с сервера. Сервлет выполняет роль контроллера запросов и ответов, то есть принимает запросы от всех связанных с ним JSP-страниц, вызывает соответствующую бизнес-логику для их (запросов) обработки и в зависимости от результата выполнения решает, какую JSP поставить этому результату в соответствие.

При необходимости расширения функциональности системы не следует создавать дополнительные сервлеты. Сервлет в приложении должен быть один.

Технология Java Server Pages (JSP) была разработана компанией Sun Microsystems (в настоящее время поглощена компанией Oracle), чтобы облегчить создание страниц с динамическим содержанием.

В то время как сервлеты наилучшим образом подходят для выполнения контролирующей функции приложения в виде обработки запросов и определения содержания и вида ответа, страницы JSP выполняют функцию отображения результатов работы приложения в виде текстовых документов типа HTML, XML, WML и некоторых других. JSP поддерживают как JavaScript, так и HTML-теги. JavaScript обычно используется, чтобы добавить функциональные возможности на уровне HTML-страницы.

Принято разделять динамическое и статическое содержимое JSP.

Динамические ресурсы. Результаты их деятельности изменяются во время выполнения приложения. Обычно представлены в виде выражений Expression Language, библиотек тегов и тегов разработчика.

Статические ресурсы. Не изменяются сами в процессе работы (HTML, JavaScript, изображения и т. д.).

Смысл разделения динамического и статического содержания в том, что статические ресурсы могут находиться под управлением HTTP-сервера в то время, как динамические нуждаются в движке (JSP Engine) и в большинстве случаев в доступе к уровню данных.

Рекомендуется разделить и разрабатывать параллельно две части приложения: часть, состоящяя только из динамических ресурсов, и часть, состоящяя только из статических ресурсов.

Некоторые преимущества использования JSP-технологии над другими методами создания динамического содержания страниц:

– разделение динамического и статического содержания. Возможность разделить логику приложения и дизайн веб-страницы снижает сложность разработки веб-приложений и упрощает их поддержку;

– независимость от платформы. Технологии Java не зависят от платформы, следовательно, JSP могут выполняться практически на любом веб-сервере. Разрабатывать JSP можно на любой платформе;

– многократное использование компонентов. Использование JavaBeans и Enterprise JavaBeans (EJB) позволяет мно гократно использовать компоненты, что ускоряет создание веб-приложений;

– теги. Спецификация JSP содержит библиотеку стандартных тегов JSTL, позволяет разработчику создавать собственные теги, кроме того, теги обеспечивают возможность использования JavaBean и обращение к классам бизнес-логики.

Содержимое Java Server Pages (теги HTML, теги JSP и скрипты) переводится в сервлет код-сервером. Этот процесс ответствен за трансляцию как динамических, так и статических элементов, объявленных внутри файла JSP. Об архитектуре сайтов, использующих JSP/Servlet-технологии, часто говорят, как о thin-client (использование ресурсов клиента незначительно), потому что большая часть логики выполняется на сервере.

В спецификации JSP 1.2 были объявлены только пять основных тегов:

– <%@ директива %> — используется для установки параметров серверной страницы JSP;

– <%! объявление %> — (нежелателен в современном программировании) содержит поля и методы, которые вызываются в expression-блоке и становятся полями и методами генерируемого сервлета. Объявление не должно производить запись в выходной поток out страницы, но может быть использовано в скриптлетах и выражениях;

– <% скриптлет %> — (нежелателен) вживление java-кода в JSP-страницу. Скриптлеты обычно используют маленькие блоки кода и выполняются во время обработки запроса клиента. Когда все скриптлеты собираются воедино в том порядке, в котором они записаны на странице, они должны представлять собой правильный код языка программирования. Контейнер помещает код Java в метод _jspService() на этапе трансляции;

– <%= вычисляемое выражение %> — (нежелателен) содержит операторы языка Java, которые вычисляются, после чего результат вычисления преобразуется в строку String и посылается в поток out;

– <%-- JSP-комментарий --%> — комментарий, который не отображается в исходных кодах JSP-страницы после этапа выполнения.

Процессы, выполняемые с файлом JSP при первом вызове:

– браузер делает запрос к странице JSP;

– JSP-engine анализирует содержание файла JSP и создает сервлет с кодом, основанным на исходном тексте файла JSP, при этом engine транслирует статическое содержимое в методы вывода и помещает его в метод _jspService(). Полученный сервлет будет ответственен за генерацию статических элементов, определенных во время разработки. Динамические элементы транслируются в java-код;

– код сервлета компилируется в файл *.class. и загружается в контейнер. В итоге сервлет на основе JSP установлен и готов к работе;

– выполняется метод init() сервлета;

– вызывается метод _jspService(), и сервлет логически исполняется, формируя экземпляр response;

– комбинация статического HTML и графики вместе с результатами исполнения динамических элементов, определенных в оригинале JSP, пересылаются браузеру через выходной поток объекта ответа HttpServletResponse.

Тема 3.5 Сессия и Cookie

При посещении клиентом веб-ресурса и выполнении вариантов запросов контекстная информация о клиенте не хранится. В протоколе HTTP нет возможностей для сохранения и изменения информации о предыдущих действиях клиента. При этом возникают проблемы в распределенных системах с различными уровнями доступа для разных пользователей. Действия, которые может делать администратор системы, не может выполнять гость. В данном случае необходима проверка прав пользователя при переходе с одной страницы на другую. В иных случаях необходима информация о предыдущих запросах клиента. Существует несколько способов хранения текущей информации о клиенте или о нескольких соединениях клиента с сервером.

Сессия есть сеанс между клиентом и сервером, устанавливаемый на определенное время, за которое клиент может отправить на сервер сколько угодно запросов. Сеанс устанавливается непосредственно между клиентом и веб-сервером в момент получения первого запроса к веб-приложению. Каждый клиент устанавливает с сервером свой собственный сеанс, который сохраняется до окончания работы с приложением.

Сеанс используется для обеспечения хранения данных при последовательном выполнении нескольких запросов различных веб-страниц или на обработку информации, введенной в пользовательскую форму в результате нескольких HTTP-соединений (например, клиент совершает несколько покупок в Интернет-магазине; студент отвечает на несколько тестов в системе дистанционного обучения). Как правило, при работе с сессией возникают следующие проблемы:

– поддержка распределенной сессии (синхронизация/репликация данных, уникальность идентификаторов и т. д.);

– обеспечение безопасности;

– проблема инвалидации сессии (expiration), предупреждение пользователя об уничтожении сессии и возможность ее продления (watchdog).

Чтобы открыть явный доступ к экземпляру сеанса/сессии пользователя веб-приложения, используются методы getSession(boolean create) или getSession() интерфейса HttpServletRequest. Экземпляр запроса возвращает ссылку на объект сессии. Метод не создает сессию, а только дает доступ с помощью запроса к экземпляру сессии HttpSession, соответствующему данному пользователю.

Если для метода getSession(boolean create) входной параметр равен true, то сервлет-контейнер проверяет наличие активного сеанса, установленного с данным клиентом. В случае успеха метод возвращает дескриптор этого сеанса. В противном случае метод устанавливает/создает новый сеанс:

```
HttpSession session = request.getSession(true);
```

После чего начинается сбор информации о клиенте.

Если метод getSession(boolean param) вызывать с параметром false, то ссылка на активный сеанс будет возвращена, если он существует, если же сеанс уничтожен или не был активирован, то метод возвратит null. Метод getSession() без параметров работает так же, как и getSession(true).

Сессия содержит информацию о дате и времени создания последнего обращения к сессии, которая может быть извлечена с помощью методов getCreationTime() и getLastAccessedTime().

Метод String getId() возвращает уникальный идентификатор, который получает каждый сеанс при создании. Метод isNew() возвращает false для уже существующего сеанса и true — для нового сеанса. Задать время (в секундах) бездействия сессии, по истечении которого экземпляр сессии будет уничтожен, можно методом setMaxInactiveInterval(long sec).

Чтобы сохранить значения переменной в текущем сеансе, используется метод setAttribute(String name, Object value) класса HttpSession, прочесть — getAttribute(String name), удалить — removeAttribute(String name). Список имен всех переменных, сохраненных в текущем сеансе, можно получить, используя метод Enumeration getAttributeNames(), работающий так же, как и соответствующий метод интерфейса HttpServletRequest.

Принудительно завершить сеанс можно методом invalidate(). В результате для сеанса будут уничтожены все связи с используемыми объектами, и данные, сохраненные в старом сеансе, будут потеряны для клиента и всех приложений.

Для хранения информации на компьютере клиента используются возможности класса javax.servlet.http.Cookie.

Cookie — это небольшие блоки текстовой информации, которые сервер посылает клиенту для сохранения в файлах cookies. Клиент может запретить браузеру прием файлов cookies. Браузер возвращает информацию обратно на сервер как часть заголовка HTTP, когда клиент повторно заходит на тот же веб-ресурс. Cookies могут быть ассоциированы не только с сервером, но и также с доменом; в этом случае браузер посылает их на все серверы указанного домена.

Файл cookie — это файл небольшого размера для хранения информации, который создается серверным приложением и размещается на компьютере пользователя. Браузеры накладывают ограничения на размер файла cookie и общее количество cookie, которые могут быть установлены на пользовательском компьютере приложениями одного веб-сервера.

Чтобы послать cookie клиенту, сервлет должен создать объект класса Cookie, указав конструктору имя и значение блока, и добавить их в объект-response. Конструктор использует имя блока в качестве первого параметра, а его значение — в качестве второго.

```
Cookie cookie = new Cookie("model", "Canon D7000");
response.addCookie(cookie);
```

Извлечь информацию cookie из запроса можно с помощью метода getCookies() объекта HttpServletRequest, который возвращает массив объектов, составляющих этот файл.

```
Cookie[ ] cookies = request.getCookies();
```

После этого для каждого объекта класса Cookie можно вызвать метод getValue(), который возвращает строку String с содержимым блока cookie. В данном случае этот метод вернет значение «Canon D7000».

Экземпляр Cookie имеет целый ряд параметров: путь, домен, номер версии, время жизни, комментарий. Одним из ключевых является срок жизни в секундах от момента первой отправки клиенту, определить который следует методом setMaxAge(long sec). Если параметр не указан, то cookie существует только до момента прерывания контакта клиента с приложением.

Тема 3.6 ASP.NET

ASP.NET – это веб-платформа, предоставляющая все необходимые службы для создания серверных веб-приложений корпоративного класса. ASP.NET создана на основе платформы .NET Framework, поэтому все функции .NET Framework доступны для приложений ASP.NET. Приложения могут быть написаны на любом языке, совместимом со средой CLR, включая Visual Basic и C#.

Для создания веб-приложений ASP.NET можно воспользоваться Visual Studio. Кроме того, имеется бесплатный самостоятельный продукт Visual Studio Express для Web, который включает в себя базовый набор функций веб-разработки, реализованных в Visual Studio.

Веб-страницы ASP.NET используются в качестве программируемого пользовательского интерфейса веб-приложения. Веб-страница ASP.NET предоставляет пользователю сведения в любом обозревателе или клиентском устройстве и реализует логику приложения с помощью серверного кода. Веб-страницы ASP.NET:

– основаны на технологии Microsoft ASP.NET, согласно которой код, выполняемый на сервере, динамически создает выходные данные веб-страниц для обозревателя или клиентского устройства.

– совместимы с любым веб-обозревателем или мобильным устройством. Веб-страница ASP.NET автоматически отображает HTML с такими возможностями, как стили, расположение и т. д. в соответствии с типом обозревателя. Кроме того, можно разработать веб-страницы ASP.NET для запуска на определенных обозревателях, например Microsoft Internet Explorer 6, и воспользоваться преимуществами функций, специфических для данного обозревателя.

– Совместимы с любым языком, который поддерживается средой CLR в .NET, включая Microsoft Visual Basic, Microsoft Visual C#, Microsoft J# и Microsoft JScript .NET.

– Построены на основе Microsoft .NET Framework. Это позволяет использовать все преимущества платформы, включая управляемую среду, строгую типизацию и наследование.

– Являются гибким инструментом, поскольку в них можно добавлять созданные пользователем или сторонним производителем элементы управления.

Компонент ASP.NET MVC (Model-View-Controller — модель-представление-контроллер) предлагает совершенно другой способ для построения веб-страниц по сравнению со стандартной моделью веб-форм.

Суть его состоит в разбиении приложения на три отдельных логических части. Модель включает весь бизнес-код приложения, например, логику доступа к данным и правила верификации. Представление создает для модели подходящее представление за счет ее визуализации в HTML-страницы. Контроллер координирует весь этот процесс за счет обработки операций взаимодействия с пользователем, обновления модели и передачи информации в представление.

В схеме MVC некоторые традиционные концепции ASP.NET, в том числе веб-формы, веб-элементы управления, состояние представления, обратные отправки и состояние сеанса, отходят на второй план.

Одним схема MVC кажется более чистой и больше подходящей для веб-приложений. Другие считают, что она заставляет прилагать дополнительные усилия и никакой очевидной выгоды при этом не приносит. Но если хоть какой-нибудь из перечисленных ниже моментов является важным, обязательно стоит рассмотреть вариант применения ASP.NET MVC:

– разработка через тестирование. Благодаря четкому разделению частей в приложении ASP.NET MVC, можно легко создать для него модульные тесты. В случае применения веб-форм автоматизированное тестирование является утомительным и зачастую невозможным;

– контроль над HTML-разметкой. В случае веб-форм программировать приходится с использованием развитого набора объектов, которые сами заботятся об управлении состоянием и генерацией HTML-разметки. В случае ASP.NET MVC содержимое вставляется больше похожим на привязку данных образом.

И хотя это означает, что проектирование страниц со сложным форматированием может потребовать больших усилий, это также означает возможность полностью контролировать каждую деталь в разметке. Это очень полезно при планировании написания какого-то клиентского сценария JavaScript либо использования сторонней библиотеки сценариев JavaScript, такой как jQuery.

– Контроль над URL-адресами. Хотя ASP.NET разработчикам продолжает предлагаться больший контроль над маршрутизацией URL-адресов, в ASP.NET MVC эта концепция является встроенной.

За сопоставление URL-адресов и логики приложения отвечают контроллеры, а это значит, что вместо /Products/List.aspx?category=Beverages можно использовать такие конфигурации URL, как /Products/List/Beverages. Такие понятные и удобные для чтения URL-адреса упрощают и делают более эффективной поисковую оптимизацию.

В Visual Studio поддерживаются **два пути создания веб-приложений** на базе ASP.NET, которые иногда приводят к некоторой путанице:

1. Разработка на основе проекта

При создании веб-проекта в Visual Studio генерируется файл проекта с расширением .csproj (при условии, что код пишется на языке C#), в котором фиксируется информация обо всех включаемых в состав проекта файлах и сохраняются кое-какие отладочные параметры. При запуске веб-проекта перед открытием веб-браузера Visual Studio сначала компилирует весь написанный код в одну сборку.

Разработка без использования проекта

2. Это альтернативный подход, который подразумевает создание просто веб-сайта безо всякого файла проекта.

При таком подходе Visual Studio предполагает, что каждый файл в каталоге веб-сайта (и всех его подкаталогах) является частью веб-приложения. В этом случае Visual Studio не требуется предварительно компилировать код. Вместо этого ASP.NET компилирует уже сам веб-сайт при первом запросе какой-нибудь входящей в его состав страницы.

В первой версии Visual Studio для .NET использовалась модель разработки на основе проекта. В Visual Studio 2005 эта модель была удалена и заменена разработкой без использования проекта. Однако это вызвало возмущение среди разработчиков.

Осознав, что для определенных сценариев все-таки больше подходила проектная разработка, Microsoft выпустила доступный для загрузки пакет, который возвращал в Visual Studio 2005 опцию проектной модели. В версии Visual Studio 2010 поддерживаются оба подхода.

РАЗДЕЛ 4 ЭКСПЛУАТАЦИЯ WEB-ПРОЕКТОВ

Тема 4.1 Продвижение и сопровождение web-проектов

Сопровождение сайта – это необходимые мероприятия по поддержанию Вашего сайта в работоспособном и актуальном состоянии в течение всего срока его службы днем и ночью. Веб-сайт является визитной карточкой любого предприятия в интернет-пространстве, поэтому наличие актуальной и свежей информации говорит о том, что предприятие работает и развивается. Включает в себя работу по комплексному техническому и информационному сопровождению сайта.

Перечень работ **по техническому сопровождению** сайта:

– сохранение бекапа сайта на внешний жесткий диск;
– контроль оплаты домена и хостинга;
– обновление движка (CMS) сайта;
– ежедневный контроль сайта на вирусы;
– круглосуточный контроль за доступностью сайта;
– наличие программиста в выходные, праздничные дни;
– исправление возникающих ошибок на сайте;
– взаимодействие с технической поддержкой.

Перечень работ **по информационному сопровождению** сайта:

– размещение на сайте предоставляемой от Заказчика информации;
– разработка новых компонентов для сайта (калькулятор, фильтр, сортировка и т.д.);
– добавление / удаление разделов на сайте;
– правка верстки страниц на сайте;
– доработка элементов дизайна сайта под знаменательные даты;
– создание баннеров для сайта;
– прочие работы по сайту.

Даже если на сайте есть интересный, уникальный, полезный пользователю контент, это еще не значит, что ресурс будет популярным. В самом деле, откуда потенциальным посетителям о нем узнать? Конкуренция почти во всех сферах невероятно высока, а значит, нужно помочь поисковым системам выделить конкретно ваш ресурс из сотен тысяч других. Для этого используется раскрутка сайта.

Когда-то для продвижения ресурса нужно было закупать ссылки и писать огромное количество seo-текстов, но сейчас этого недостаточно. В XXI веке услуга превратилась в целый комплекс глубоких аналитических процессов, предназначенных для решения ряда задач.

В самом широком смысле раскрутка сайта – это набор действий, целью которых является развитие и увеличение доли видимости ресурса в поисковых системах, привлечение максимального количества целевых посетителей, превращение их в покупателей товаров и услуг.

Способы раскрутки сайта, методы продвижения сайта в интернете

В интернете подробно описаны методы продвижения сайта – платные и бесплатные. На самом деле эта классификация не совсем правильна. В основе любого варианта лежит тяжелый человеческий труд – самый дорогостоящий компонент этого направления.

Можно ли повысить популярность ресурса бесплатно? До некоторой степени – да. Однако со временем задача будет требовать все больше времени и сил. Придется уделять ей практически каждый день! Большинству владельцев бизнеса это не слишком удобно, так что нужно будет обращаться к специалистам. Можно сделать это и на раннем этапе, чтобы сократить сроки достижения результата и повысить эффективность процесса.

В раскрутку сайта входит несколько важных мероприятий.

– **Аналитика**

Анализ поведения посетителей. Как они находят ваш сайт или приложение? Как с ним взаимодействуют? Сколько времени в среднем проводят на определенных страницах? Оптимальный способ получить такие сведения – использовать специальные сервисы, вроде Google Analytics. Продукт Google предоставляет наибольшее количество возможностей, но он довольно сложен в использовании. Большинство владельцев сайтов применяют лишь малую часть его возможностей. Чтобы добиться максимальной эффективности, лучше заказать услуги экспертов.

Назначение Google Analytics – сбор и анализ данных с разных устройств и цифровых средств. Система предоставляет отчеты в самом наглядном виде. Для начала продвижения сайта нужно установить на него код счетчика. Это несколько строк, которые следует скопировать прямо в HTML. После этого пользователь, посещающий ресурс, автоматически запускает аналитику и сайт запоминает все совершенные действия.

Как взаимодействовать с собранной информацией? Для этого у «Гугла» есть множество фильтров, которые следует настроить под нужды вашего бизнеса. Например, можно сделать так, чтобы система не учитывала посещения от ваших сотрудников, и т. д.

– **SEO (**Search Engine Optimization, поисковая оптимизация**)**

Обычно неспециалисты, которые говорят о продвижении сайта, подразумевают именно SEO. Эта услуга представляет собой поисковую оптимизацию, т. е. приведение ресурса в состояние, которое обеспечит его заметность для поисковых систем. Если делать все правильно, сайт постепенно начнет подниматься в результатах выдачи, а значит, вероятность его посещения будет расти. Почти 100% пользователей поисковиков заходят только по первым трем ссылкам. Затем показатель снижается. Лишь около 20%добираются до второй страницы.

Как добраться до верхних строк выдачи в условиях жесточайшей конкуренции? Нужно прибегнуть к SEO-раскрутке сайта. Она выполняется в несколько этапов:

Первый этап: работа внутри ресурса. Сюда входит исправление ошибок, добавление и переработка контента, создание перелинковки и многое другое. Специалисты называют эти процессы «внутренней оптимизацией». Следует учитывать, что у разных поисковиков разные требования к состоянию сайта. Рекомендуется обращать внимание на две самые популярные системы – Google и Яндекс.

Второй этап: внешнее продвижение сайта. Нужно работать на других ресурсах, каталогах статей, форумах и других площадках. Цель этого этапа – увеличение естественной ссылочной массы, которая будет повышать «авторитет» вашего ресурса и увеличивать его позиции в результатах выдачи.

Третий этап: удерживание занятых позиций. Конкуренты не дремлют, а тоже постоянно занимаются продвижением сайтов! Чтобы их мероприятия не «вытолкнули» ваш ресурс с заслуженного места, необходимо постоянно анализировать поисковую выдачу, сайты конкурентов, следить за новинками в области SEO, внедрять доработки на свой сайт…

Однако SEO – это еще не полный ответ на вопрос «что такое продвижение сайта». Для достижения успеха надо уделить внимание третьему моменту:

– **Юзабилити**

Цель коммерческого сайта – продать посетителю товары или услуги. Юзабилити, т. е. удобство пользования ресурсом – один из важнейших инструментов в достижении такой цели. По сути, клиентам безразлично, насколько изящные способы применялись во время разработки – их интересует исключительно возможность достичь своей цели, причем как можно быстрее и без усилий. Если интерфейс будет приятным, интересным и комфортным, посетитель не только купит у вас что-нибудь – он еще и непременно вернется. А может, даже порекомендует сайт друзьям.

Статистика гласит, что интернет-магазины теряют около 50% покупателей из-за неудачного юзабилити. Человек, который не смог быстро найти нужный товар, вряд ли будет тратить свое время – скорей всего, он уйдет к конкурентам, а сюда уже никогда не вернется. Таким образом, работа с юзабилити становится одним из важнейших инструментов в раскрутке сайта. Существует несколько требований, которым должен соответствовать ваш ресурс:

– понятная с первого взгляда структура;
– возможность быстро определить, где находится требуемая информация;
– соответствие текста форматам представления.

При работе с юзабилити нужно встать на место посетителя, представить себя «по ту сторону экрана».

Помните, что перечисленные выше направления дают эффективный результат только в комплексе. Будучи примененными по отдельности, они не обеспечат роста посещаемости и увеличения покупок.

Другие методы продвижения сайта

– **Реклама**

Нередко при раскрутке используется реклама:

– Контекстная. Текстово-графические материалы располагаются рядом с результатами поиска Google, Яндекса и аналогичных сервисов. Главная особенность контекстной рекламы – направленность на тех пользователей, что могут заинтересоваться вашей продукцией. Принцип действия прост: вы создаете и настраиваете объявления с определенными ключевыми словами. Пользователи, которые вводят эти слова в строке поиска, имеют шанс увидеть ваши объявления. Система работает по принципу аукциона – чем больше заплатить, тем выше вероятность демонстрации. Также к контекстной рекламе относятся ретаргетинг и ремаркетинг – показ объявлений на основании уже посещенных пользователем ресурсов.

– Медийная. Текстово-графические материалы размещаются на специальных рекламных площадках. По принципу действия этот метод продвижения сайта схож с рекламой в обычных СМИ, но у него есть преимущества – например, возможность использования анимированных изображений. Вдобавок, заинтересованный объявлением пользователь может сразу же перейти по ссылке и сделать покупку.

Существуют и иные методы рекламы сайтов. Например, можно вставлять ее в рассылку по подписке, или в клиенты программ, которые применяют пользователи.

– **E-mail-маркетинг**

Многие недооценивают этот способ раскрутки, и совершенно зря. Во-первых, электронная почта есть почти у каждого пользователя сети. Во-вторых, можно персонифицировать сообщения, подготовив текст и оформление под конкретного пользователя. В-третьих, многие получатели интересных писем распространяют их среди друзей и знакомых.

Продвижение сайта с помощью e-mail-рассылки может осуществляться следующими способами:

– Отправка сообщений подписчиками. Зарегистрировавшись на определенных ресурсах, пользователи могут дать согласие на получение рассылки по определенной тематике. Существуют открытые (которые могут получить все желающие) и закрытые (строго для определенных получателей) виды рассылки. Также они бывают бесплатными и платными. Инструмент эффективен за счет своей массовости. Письма работают на благо сайта и бренда в целом, напоминая пользователям о его существовании.

– Реклама в рассылках. Можно заказать размещение вашей рекламы в письмах других компаний. Обычно речь идет о баннерах с гиперссылками, которые располагаются в определенных частях тела письма. Для достижения эффективности нужно аккуратно выбирать

тематику рассылки. Например, реклама курсов маляров-штукатуров вряд ли принесет результат в письмах для поклонников дорогих автомобилей.

– Индивидуальная рассылка. Сложный, но эффективный инструмент, который требует серьезных трат времени и сил. Каждое сообщение направляется конкретному пользователю (или небольшой группе пользователей). Предварительно надо собрать адреса и проанализировать предпочтения получателя. Главным достоинством является то, что письмо получает заинтересованный пользователь, так что вероятность покупки весьма высока.

Следует упомянуть и о т. н. «спаме». Спам – это несанкционированная массовая рассылка. Помните – ни в коем случае нельзя отправлять большое количество рекламных сообщений пользователям, которые не давали на это согласия! В большинстве стран такие действия преследуются по закону, но самое главное – эффективность спам-писем минимальна. Они приносят скорее вред, чем пользу, поскольку плохо влияют на авторитет компании и снижают лояльность клиентов.

– **SMM**

Аббревиатура SMM означает Social Media Marketing, то есть маркетинг в социальных сетях. Это эффективный инструмент привлечения внимания к ресурсу, однако для успешной реализации нужно знать множество нюансов. Кроме соцсетей, можно использовать его принципы в сообществах, дневниках, блогах, форумах и аналогичных площадках.

Аудитория социальных сетей огромна – по сути, она уже превосходит аудиторию крупнейших телеканалов. Главные особенности пользователей таких ресурсов – внимательность и активность. Работа специалиста по SMM имеет много общего с PR. Работать нужно непосредственно в сообществах с целевой аудиторией, используя прямые и скрытые меры взаимодействия.

Чего можно добиться, применяя SMM?

– увеличения лояльности аудитории;
– повышения посещаемости сайта;
– продвижения бренда;
– PR.

Эксперты считают этот инструмент одним из самых перспективных. Огромное количество компаний (от малого до крупного бизнеса) уже завели паблики «ВКонтакте» и на Facebook. Эти сообщества используются для налаживания контакта с потребителем, продвижения товаров и услуг.

Обычно используются следующие инструменты:

– Ведение блога – публикация записей, оформление материалов. Можно создавать свой контент или использовать уже имеющийся.

– Общение в тематических сообществах, взаимодействие с пользователями, ответы на комментарии.

– «Агенты влияния» – скрытый маркетинг.

– Тематические обсуждения.

– Прямая реклама. Можно заказать упоминание вашего сообщества у другого сообщества, или у популярного блогера.

– Вирусный маркетинг.

– Создание положительного информационного фона.

– Оптимизация вашего сайта под социальные сети.

SMM не дает мгновенного результата. Инструмент нацелен на долгосрочную перспективу. Его главным достоинством является невысокая стоимость при возможности достичь отличного результата.

Итак, что такое раскрутка сайта? Это множество методов и инструментов, с помощью которых удается привлечь пользователей на ресурс. Помните, что результата можно добиться только с помощью комплекса мер.