

Содержание

Введение.....	2
1 Обзор стандарта HTML 5.....	2
1.1 Языки гипертекстовой разметки.....	2
1.2 Элементы оформления текста.....	6
1.3 Таблицы.....	12
1.4 Графика.....	13
1.5 Формы и элементы управления.....	16
2 Каскадные таблицы стилей CSS.....	22
2.1 Синтаксис CSS правил.....	22
2.2 Стилизация текста.....	31
2.3 Табличная и блочная верстка.....	35
2.4 Видимость и эффекты.....	42
3 Событийное программирование JavaScript.....	47
3.1 Синтаксис языка JavaScript.....	47
3.2 Объектная модель браузера.....	75
3.3 Доступ к элементам страницы.....	77
3.4 Обработка форм.....	85
3.5 Асинхронная передача данных.....	88
3.6 Форматы обмена данными.....	91
3.7 Библиотека JQuery.....	95
4 Сервер приложений.....	103
4.1 Установка сервера приложений.....	103
4.2 Настройка сервера приложений.....	108
4.3 Администрирование сервера приложений.....	110
5 Серверное программирование на PHP.....	110
5.1 Элементы языка PHP.....	110
5.2 Обработка входных параметров.....	110
5.3 Объектно-ориентированное программирование.....	110
6 Разработка веб-приложений.....	110
6.1 Паттерн проектирования MVC.....	110
6.2 Маршрутизация URL.....	111
6.3 Отладка и профилирование.....	111
6.4 Механизмы аутентификации и авторизации.....	111
6.5 Безопасность веб-приложений.....	111

Введение

1 Обзор стандарта HTML 5

1.1 Языки гипертекстовой разметки

SGML (*Standard Generalized Markup Language* — стандартный обобщённый язык разметки) — метаязык, на котором можно определять язык разметки для документов.

Изначально SGML был разработан для совместного использования машинно-читаемых документов в больших правительственных и аэрокосмических проектах. Он широко использовался в печатной и издательской сфере, но его сложность затруднила его широкое распространение для повседневного использования.

SGML предоставляет множество вариантов синтаксической разметки для использования различными приложениями.

SGML стандартизован ISO: «ISO 8879:1986 Information processing—Text and office systems—Standard Generalized Markup Language (SGML)».

От SGML произошли языки разметки HTML и XML. HTML — это приложение SGML, а XML — это подмножество SGML, разработанное для упрощения процесса машинного разбора документа.

HTML (*HyperText Markup Language* — язык гипертекстовой разметки) — стандартный язык разметки документов в сети Интернет. Большинство веб-страниц содержат описание разметки на языке HTML. Язык HTML интерпретируется браузерами, а полученный в результате интерпретации форматированный текст отображается на экране монитора компьютера или мобильного устройства.

Язык HTML соответствует международному стандарту ISO 8879.

В сети Интернет HTML-страницы, как правило, передаются браузерам от веб-серверов по протоколу HTTP в виде простого текста или по протоколу HTTPS с использованием шифрования.

Язык HTML был разработан британским учёным Тимом Бернерсом-Ли приблизительно в 1986—1991 годах в ЦЕРН в Женеве в Швейцарии. HTML представляет собой небольшой набор структурных и семантических элементов — тегов. С помощью HTML можно легко создать относительно простой оформленный документ. Помимо упрощения структуры документа, в HTML внесена поддержка гипертекста. Позже в него были добавлены возможности мультимедиа.

Изначально язык HTML был задуман и создан как средство структурирования и форматирования документов без их привязки к средствам отображения. С

течением времени это свойство HTML было ориентировано на потребности в мультимедийном и графическом оформлении. Для одинакового отображения HTML страниц в различных браузерах потребовалось стандартизировать используемые теги и атрибуты и утвердить их в качестве стандарта веб-консорциумом (W3C).

Официальной спецификации HTML 1.0 не существует. До 1995 года существовало множество неофициальных стандартов HTML. Чтобы стандартная версия отличалась от них, ей сразу присвоили второй номер.

Стандарт HTML 3 был предложен W3C в марте 1995 года. Он обеспечивал возможности создание таблиц, «обтекание» изображений текстом и отображение сложных математических формул, поддержка графических изображений в GIF формате.

Несколько лет в Интернете продолжалась так называемая «война браузеров», связанная с различной реализацией HTML в браузерах *Microsoft Internet Explorer* и *Netscape Navigator*, поэтому следующей версией стандарта HTML стала 3.2, в которой были опущены многие нововведения версии 3.0, но добавлены нестандартные элементы, поддерживаемые браузерами *Netscape Navigator* и *Mosaic*.

В версии HTML 4.0, опубликованной в 1999 г. произошло обновление стандарта. Многие элементы были отмечены как устаревшие и nereкомендованные, например тег *font*, используемый для изменения свойств шрифта, был помечен как устаревший (вместо него рекомендовано использовать таблицы стилей CSS). С введением этой спецификации, а также повсеместного распространения Интернет на пользовательском сегменте рынка, разработчики браузеров стали активнее поддерживать рекомендации W3C.

В настоящее время W3C принята версия HTML 5. Стандарт был официально рекомендован к применению W3C в 2014 году, но еще с 2013 года всеми браузерами оперативно осуществлялась поддержка, а разработчиками — использование рабочего стандарта HTML 5. Целью разработки HTML 5 является улучшение поддержки мультимедиа-технологий, сохранение обратной совместимости, удобочитаемости кода для человека и простоты анализа HTML страниц для парсеров.

В стандарте HTML 5 вводится понятие семантической верстки, удобной для более эффективной последующей стилизации страницы средствами CSS, а также для индексации страниц поисковыми системами. В дополнение к этому HTML 5 устанавливает API, который может быть использован совместно с языком JavaScript, убраны устаревшие теги, добавлен элемент холст с методами рисования 2D, контроль над проигрыванием медиафайлов, управление хранилищем веб-данных, реализация технологии *drag-and-drop* применительно к элементам страницы DOM, регистрация обработчиков MIME типов, возможность применения микроразметки и др.

С принятием стандарта HTML 5 наконец появилась возможность полного

разделения содержания, представления и поведения веб-страниц. Эта возможность получила реализацию в новой концепции разработки Web 2.0.

XHTML

В 1998 году веб-консорциум начал работу над новым языком разметки, основанным на HTML 4, но соответствующим синтаксису XML. Впоследствии новый язык получил название XHTML. Первая версия XHTML 1.0 одобрена в качестве рекомендации W3C в 2000 г. В дальнейшем вся работа над стандартом XHTML была приостановлена. Последняя используемая спецификация XHTML 2.0 была перенесена разработчиками в стандарт HTML 5.

Согласно синтаксису XHTML:

- все элементы должны быть закрыты. Теги, которые не имеют закрывающего тега (например, `
`), должны завершаться символом / (слеш): `
`;
- недопускаются атрибуты без содержания. Логические атрибуты всегда должны иметь значения: `<td nowrap="nowrap">`;
- имена тегов и атрибутов должны быть записаны только строчными буквами;
- XHTML гораздо строже относится к ошибкам в коде, например, символы `<` и `&` везде, включая URL адреса, должны замещаться специальными сущностями `<` и `&`; соответственно. По рекомендации W3C браузеры, встретив ошибку в XHTML, должны сообщить о ней и не обрабатывать документ. В случае с HTML браузеры пытаются обработать страницу;
- кодировкой по умолчанию является UTF-8.

XHTML с успехом используется в информационных системах, имеющих XML совместимые шаблонизаторы (например XSLT процессоры) для вывода генерируемого содержимого.

XML (*eXtensible Markup Language*) - расширяемый язык разметки, является подмножеством SGML, рекомендован W3C. Спецификация XML описывает XML-документы и частично описывает поведение XML-процессоров - программ, читающих XML-документы и обеспечивающих доступ к их содержимому. XML разрабатывался как язык с простым формальным синтаксисом, удобный для создания и обработки документов программами и одновременно удобный для чтения и создания документов человеком, с подчёркиванием нацеленности на использование в Интернете.

Язык XML называется расширяемым, поскольку он не фиксирует разметку, используемую в документах: разработчик волен создать разметку в соответствии с потребностями конкретной области, будучи ограниченным лишь синтаксическими правилами языка. Сочетание простого формального синтаксиса, удобства, расширяемости, использования кодировки UTF-8 для представления содержания документов привело к широкому использованию как XML, так и множества производных специализированных языков на базе XML

в самых разнообразных программных средствах.

Пример корректного XML документа:

```
<?xml version="1.0" encoding="utf-8"?>
<book>
  <title>Введение в Perl</title>
  <authors>Рэндал Шварц, Том Крустманен</authors>
  <publisher>BNV Group</publisher>
  <year>1999</year>
  <content>Содержание</content>
</book>
```

В настоящее время XML широко используется для межсетевого обмена данными в веб-сервисах, в протоколе SOAP, в качестве содержания документов открытого формата ODF, для хранения структурированных данных в файловых хранилищах, в системных конфигурационных файлах и др.

Синтаксис HTML

Документ на языке HTML представляет собой набор элементов, заключенных в угловые скобки — тегов. Теги могут быть одинарными, не имеющими содержимого и парными (контейнерными), имеющими содержимое. Перед именем завершающего контейнерного тега ставится символ / (слеш).

Пример одиночного тега: `
`

Пример контейнерного тега: `<p>Загадки разума</p>`

Контейнерные теги могут включать в себя другие вложенные теги:

```
<p><strong>Что в имени тебе моем?</strong></p>
```

Каждый тег может содержать атрибуты и через символ = (равенство) их значения, заключенные в кавычки:

```
<p class="pod">Александр Великий</p>
```

Атрибуты являются свойствами, предоставляющими дополнительную информацию о теге.

Имена тегов и их атрибутов специфицированы в стандарте HTML и соответствуют общим правилам идентификаторов: имена содержат латинские буквы, цифры, символы тире и подчеркивания; каждое имя начинается с латинской буквы. Прописные и заглавные буквы в именах не различаются.

Если тег имеет несколько атрибутов, то они отделяются друг от друга пробелами: `<meta http-equiv="content-type" content="text/html">`

Если атрибут не имеет содержания, то символ = (равенство) и пустые кавычки опускаются: `<td nowrap>Пупрова победа</td>`

Универсальные атрибуты: *id* — уникальный идентификатор элемента на веб-странице; *class* — имя класса стилевой таблицы; *src* — источник встраиваемого содержимого; *href* — ссылка на связанный ресурс.

Запрещается использовать угловые скобки < и >, а также кавычки вне HTML разметки. При необходимости использования данных символов в тексте они должны замещаться на специальные сущности < > и " соответственно. Список наиболее часто употребляемых сущностей:

Символ	Сущность	Символ	Сущность	Символ	Сущность
<	<	«	«	®	®
>	>	»	»	©	©
"	"	–	–	пробел	
'	&apost;	—	—	код UTF-8	Ӓ

Для указания комментариев в тексте HTML документа используется метатег `<!-- текст комментариев -->`. Внутри комментариев разрешается добавлять другие теги. Содержимое комментариев браузером не отображается.

Структура HTML документа

Типовая структура HTML документа:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Заголовок страницы</title>
  </head>
  <body>
    ... содержание страницы ...
  </body>
</html>
```

Согласно спецификации HTML 5 первым тегом документа является метатег `<!DOCTYPE html>`, после которого размещается сам HTML документ. Данный метатег представляет собой команду для корректного отображения документа в браузере.

Тег `<html>` является главным корневым тегом веб-страницы, он содержит два контейнерных тега `<head>` и `<body>`, задающих заголовок страницы и ее содержание. Заголовок `<head>` содержит служебную часть, которая содержит различные теги, определяющие тип и формат содержимого для браузера (метатеги), а также ссылки на различное подгружаемое содержимое: каскадные стилевые таблицы CSS, скрипты со сценарием поведения Javascript и др. В данном примере указано минимальное содержимое веб-страницы в HTML 5: тег `<meta>` с помощью атрибута `charset` задает кодировку документа; тег `<title>` задает заголовок окна. Тело документа `<body>` представляет собой непосредственно выводимое содержимое на экран браузера.

1.2 Элементы оформления текста

Семантическая верстка страницы

Стандартом HTML 5 предусмотрены семантические теги для верстки содержимого страницы. Структура сайта может состоять из нескольких отдельных частей, которые могут быть представлены соответствующими контейнерными тегами: шапка сайта `<header>`, главное верхнее меню `<nav>`, боковая панель `<aside>`, содержимое `<article>` и подвал `<footer>`. Содержимое, в свою очередь, может состоять из одной или нескольких секций `<section>`, содержащих заголовки, абзацы, списки и другие структурные HTML элементы. Элементы `<section>` могут быть вложенными.

Также в HTML 5 может использоваться семантический тег `<main>`, задающий основное содержимое страницы. На странице может быть только один тег `<main>`, при этом он не должен располагаться внутри элементов `<article>`, `<aside>`, `<footer>`, `<header>` или `<nav>`.

Основное отличие семантической верстки от физической состоит в том, что теги семантической верстки никаким образом не влияют на изначальное представление HTML страницы в браузерах — если разметка не стилизована средствами CSS, то она не отображается. Впоследствии для каждого такого элемента можно определить свой собственный стиль отображения, не прибегая к описанию именных классов и уникальных стилей в каскадных таблицах CSS.

Таким образом, использование семантической верстки HTML 5 позволяет значительно упростить представление отдельных элементов страницы и улучшить читабельность HTML кода.

Заголовки

HTML предлагает шесть заголовков разного уровня, которые показывают относительную важность секции, расположенной после заголовка. Так, элемент `<h1>` представляет собой наиболее важный заголовок первого уровня, а `<h6>` служит для обозначения наименее важного заголовка шестого уровня.

Пример заголовка:

`<h1>Заголовок первого уровня</h1>`

В браузере заголовки отображаются шрифтом жирного начертания от `<h1>` к `<h6>` в сторону уменьшения размера. Заголовок `<h4>` отображается размером текущего текста.

Элементы `<h1>`,...,`<h6>` относятся к контейнерным элементам, они всегда начинаются с новой строки, как и элементы, следующие за ними. До и после заголовка добавляется абзацный интервал.

Абзацы

Основной структурной единицей текста является абзац, выделяемый в контейнерный тег `<p>`. До и после абзаца добавляется абзацный интервал.

По умолчанию, содержимое абзаца отображается в браузере выровненным по левому краю экрана (родительского контейнерного тега). Тип выравнивания

можно установить с помощью атрибута *align*, принимающим одно из четырех значений:

left — выравнивание
содержимого абзаца слева;
right — выравнивание
содержимого абзаца справа;
center — выравнивание
содержимого абзаца по центру;
justify — выравнивание
содержимого абзаца по ширине.



Атрибут *align* является устаревшим.

Вместо использования атрибута *align* рекомендуется устанавливать отдельный стиль для тега `<p>` в каскадной таблице CSS.

Блочные элементы

Блочные элементы характеризуются тем, что они занимают всю доступную ширину, высота элемента определяется его содержимым, и каждый элемент начинается с новой строки.

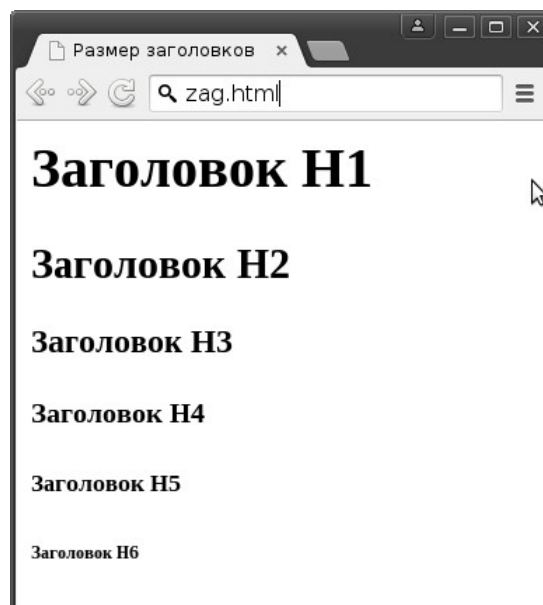
Для оформления текста в HTML имеется несколько блочных элементов. Все эти элементы представляют собой контейнеры:

`<div>` - представляет собой универсальный контейнер, служащий для блочной верстки. В отличие от тега `<p>` при отображении элемента браузером отсутствуют абзацные интервалы. Стилизация определяется средствами CSS.

`<blockquote>` - предназначен для выделения длинных цитат внутри документа. Текст, обозначенный этим тегом, традиционно отображается как выровненный по ширине абзац с отступами слева и справа (около 40 пикселей), с абзацными интервалами снизу и сверху.

`<address>` - предназначен для оформления контактных данных. Текст отличается от оформления абзаца `<p>` курсивным начертанием.

`<pre>` - задает блок исходного отформатированного текста. Такой текст отображается обычно моноширинным шрифтом, со всеми пробелами между словами. Обычно при отображении HTML страницы в браузере любое количество пробелов идущих подряд, а также символов перевода строки, отображаются как один пробел. Тег `<pre>` позволяет обойти эту особенность и отображать исходное форматирование



текста.

`<hr>` - задает декоративную горизонтальную разделительную черту. Данный элемент не имеет завершающего тега и является одиночным.

Строчные элементы

Строчные элементы занимают ширину и высоту, определяемую содержимым и могут размещаться внутри абзацев и других блочных элементов. Как правило, строчные элементы служат для выделения и оформления текста. Большинство строчных элементов представляют собой контейнеры:

`` - представляет собой универсальный контейнер строчного содержимого. Стилизация определяется средствами CSS.

`` - выделение текста (также ``). Содержимое тега при просмотре в браузере выделяется полужирным шрифтом.

`` - акцентирование текста (также `<i>`). Содержимое тега при просмотре в браузере выделяется курсивом.

`<q>` - содержимое элемента в браузере отображается в кавычках.

`<cite>` - оформление цитаты. Содержимое тега при просмотре в браузере выделяется курсивом.

`
` - разрыв строки. Данный элемент не имеет завершающего тега и является одиночным. Отображение последующего за ним текста в браузере переносится на следующую строку.

Пример: `<p>Первая строка.
Вторая строка</p>`

Списки

Стандарт HTML позволяет представлять два вида списков: нумерованных и маркированных.

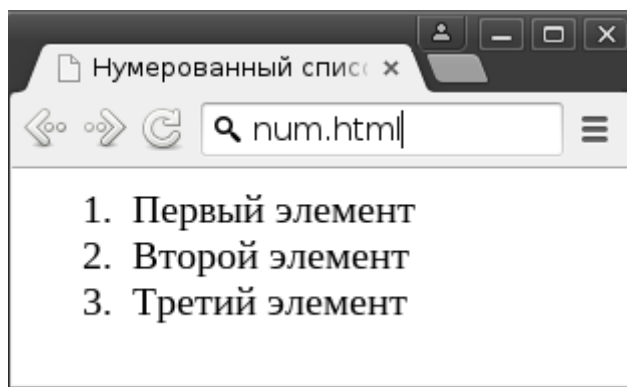
Нумерованный список задается контейнерным тегом ``. Списки состоят из произвольного количества вложенных элементов ``, также представляющих собой контейнеры. При отображении в браузере каждый элемент списка будет иметь абзацный отступ и нумерацию. Необязательный атрибут *start* задает начальное число, с которого начинается нумерация (по умолчанию с 1). Для установки нумерации отдельного элемента списка `` может использоваться атрибут *value*.

Пример нумерованного списка и его отображение в браузере:

```

<ol>
  <li>Первый элемент</li>
  <li>Второй элемент</li>
  <li>Третий элемент</li>
</ol>

```



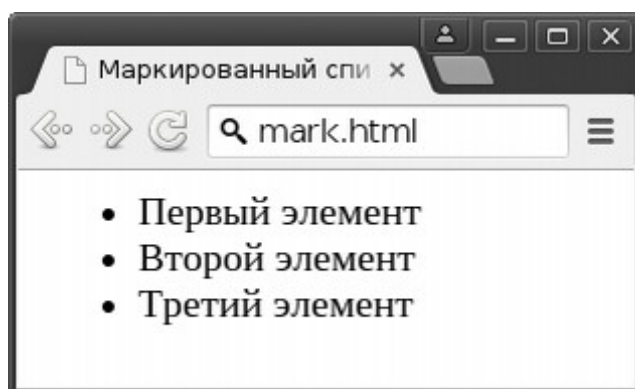
Маркированный список задается контейнерным тегом ``, содержащим элементы списка ``. При отображении в браузере каждый элемент списка будет иметь абзацный отступ и предваряться маркером.

Пример маркированного списка и его отображение в браузере:

```

<ul>
  <li>Первый элемент</li>
  <li>Второй элемент</li>
  <li>Третий элемент</li>
</ul>

```



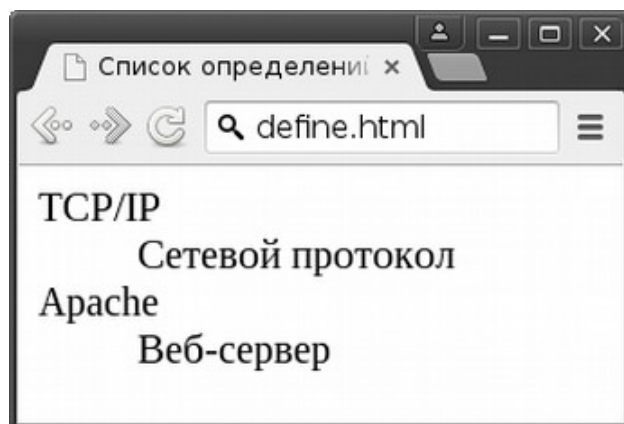
Список определений задается контейнерным тегом `<dl>`. Элементами списка определений являются пары тегов `<dt>`, определяющих термины, и тегов `<dd>`, определяющих разъяснения этих терминов. При отображении в браузере термины выравниваются по левому краю, а разъяснения выводятся с новой строки с небольшим отступом.

Пример списка определений и его отображение в браузере:

```

<dl>
  <dt>TCP/IP</dt>
  <dd>Сетевой протокол</dd>
  <dt>Apache</dt>
  <dd>Веб-сервер</dd>
</dl>

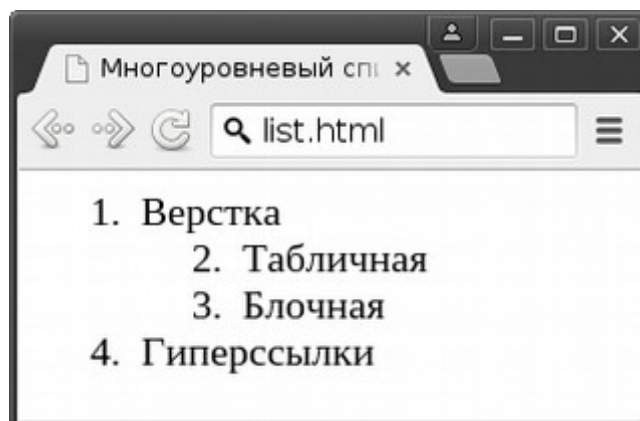
```



В HTML разрешается использовать вложенные списки. Для этого все теги внутреннего списка вкладываются в элемент `` внешнего списка. При отображении вложенных списков в браузере выводится только текущий уровень нумерации элементов. Каждый уровень списка отображается со своим отступом.

Пример вложенного списка и его отображение в браузере:

```
<ol>
  <li>Верстка
    <ol start="2">
      <li>Табличная</li>
      <li>Блочная</li>
    </ol>
  </li>
  <li value="4">Гиперссылки</li>
</ol>
```



Гиперссылки

Гиперссылки в HTML представляются строчным контейнерным тегом `<a>`. По умолчанию, гиперссылки отображаются в браузере подчеркнутым текстом синего цвета. Посещенные гиперссылки отображаются сиреневым цветом. При наведении указателя на гиперссылку он принимает вид указательного пальца. При активации гиперссылки происходит переход по заданному URL адресу и открытие в окне браузера документа, находящихся по этому адресу.

Атрибуты: *href* — URL адрес документа, на который следует перейти;

target — цель гиперссылки, указывающая имя окна или фрейма, куда будет загружаться документ. Для загрузки в новое окно браузера указывается значение *target="_blank"*;

download — вывод диалогового окна с предложением загрузки или сохранения документа (работает в Chrome, Firefox, Opera, Microsoft Edge).

В качестве гиперссылок могут использоваться как абсолютные, так и относительные URL адреса. При использовании абсолютной адресации в URL рекомендуется указывать протокол. Основные типы URL, используемые при отображении HTML страниц в браузерах:

http://htmlbook.ru/test.html - протокол передачи гипертекста (*HyperText Transfer Protocol*);

https://htmlbook.ru/test.html - защищенный протокол передачи гипертекста с использованием шифрования (*HyperText Transfer Protocol Security*);

ftp://htmlbook.ru/file.zip - файловый протокол передачи данных (*File Transfer Protocol*);

file:///d:/example/test.zip — адрес файла на локальном компьютере;

mailto:ivanov@gmail.com — адрес электронной почты;

skype:ivanov?call — вызов абонента по адресу Skype.

Относительные адреса позиционируются от папки размещения текущей страницы на веб-сервере:

```
<a href="example/test.html">Относительная ссылка</a>
```

`Абсолютная ссылка`

Переход по гиперссылке будет зависеть от типа загружаемого документа. Например архивные файлы с расширением zip или rar будут сохраняться на локальный диск:

`Тестовый архив`

1.3 Таблицы

Таблица в HTML представляет собой отдельный блочный элемент `<table>`. В составе таблицы можно выделить отдельные структурные элементы: `<caption>` - заголовок; `<thead>` - начальный колонтитул; `<tbody>` - содержание; `<tfoot>` - конечный колонтитул. Все эти элементы являются опциональными и, за исключением тега `<caption>`, могут включать одну или несколько строк таблицы `<tr>`. Если в таблице содержание и колонтитулы не указаны, то строки `<tr>` могут включаться непосредственно в содержание тега `<table>`.

Каждая строка таблицы `<tr>` может включать одну или несколько ячеек. Ячейки могут быть представлены двумя тегами: `<td>` - ячейка; `<th>` - заголовок столбца. По умолчанию текст заголовка столбца отображается в браузере полужирным шрифтом, выровненным по центру; текст простой ячейки — выровненный по левому краю; текст заголовка — выровненным по центру. Размер таблицы по умолчанию определяется ее содержимым, границы таблицы не отображаются, по вертикали содержимое всех ячеек выравнивается по центру.

Пример таблицы, ее структура и отображение в браузере:

```
<table>
  <caption>Успеваемость</caption>
  <thead>
    <tr>
      <th>Предмет</th>
      <th>Оценка</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Математика</td>
      <td>4 (хорошо)</td>
    </tr>
    <tr>
      <td>Программирование</td>
      <td>5 (отлично)</td>
    </tr>
  </tbody>
</table>
```



```

<tr>
  <td>Средний балл</td>
  <td>4,5</td>
</tr>
</tfoot>
</table>

```

Предмет	Оценка
Математика	4 (хорошо)
Программирование	5 (отлично)
Средний балл	4,5

Атрибут *border* тега таблицы *<table>* задает ширину внешней границы таблицы в пикселах (по умолчанию, 0) и позволяет сделать ее отображаемой. Данный атрибут является устаревшим, рекомендуется использовать каскадную таблицу стилей CSS.

Атрибут *colspan* тегов *<td>* и *<th>* задает объединение соседних ячеек по горизонтали (число колонок). Атрибут *rowspan* — задает объединение соседних ячеек по вертикали (число строк).

Пример таблицы с объединенными ячейками и ее отображение в браузере:

```

<table border="1">
  <caption>Успеваемость</caption>
  <tr>
    <td rowspan="2">Иванов</td>
    <td colspan="2">Оценки</td>
  </tr>
  <tr>
    <td>4</td>
    <td>5</td>
  </tr>
</table>

```

Иванов	Оценки	
	4	5

Таблицы могут иметь произвольную вложенность друг в друга — на этом основан так называемый табличный дизайн страницы. Размеры таблиц, а также отдельных строк и столбцов, стиль отображения границ, выравнивание содержимого ячеек и другие свойства задаются с помощью стилизации CSS.

1.4 Графика

Форматы графических изображений

В HTML страницах могут быть внедрены графические файлы трех основных форматов:

Формат GIF (*Graphics Interchange Format*) — разработан в 1987 году, используется для хранения элементов оформления HTML страниц. Изображения GIF поддерживают кодирование только 256 цветов, что в большинстве случаев является недостаточным для полноцветного

представления. В настоящий момент считается устаревшим. Достоинства формата GIF: использование алгоритма сжатия LZW без потерь качества; малый размер файлов и высокая скорость их загрузки; поддержка прозрачности; возможность анимирования изображений.

Формат PNG (*Portable Network Graphics*) — разработан в 1996 году на замену устаревшего формата GIF, используется для хранения полноцветных изображений, содержащих текст, чертежи и схемы (тонкие линии). Достоинства формата PNG: использование алгоритма сжатия *Deflate* без потерь качества; поддержка прозрачности; использование полноцветной палитры *TrueColor* (16М цветов, 24 бит/пиксел). Анимирование изображений невозможно.

Формат JPEG (*Joint Photographic Expert Group*) — разработан в 1993 году для хранения полноцветных изображений в фотографии. Достоинства формата JPEG: использование высокоэффективного алгоритма сжатия с усреднением, позволяющего сохранять без видимых потерь качества полноцветные фотографические изображения в относительно малых по размеру файлах. Прозрачность и анимирование изображений не поддерживается. Формат плохо подходит для хранения изображений, содержащих текст, схемы и тонкие линии.

Формат SVG (*Scalable Vector Graphics*) - разработан в 2001 году для представления векторной 2D графики в Интернете. Данные об объектах представляются в формате XML, версия SVG 1.1 поддерживается большинством современных браузеров. Отрисовка графических объектов производится средствами браузера на основе описанных в файле команд. Имеется поддержка анимации, векторных примитивов, градиентной заливки, стилизация средствами CSS, объектами SVG можно управлять с помощью Javascript. Достоинством является относительно малый размер файлов и наличие открытого интерфейса для взаимодействия объектов с пользователями.

Вставка изображения на страницу

Для вставки изображения на страницу используется одиночный тег ``. Обязательный атрибут `src` указывает URL изображения. Изображение будет вставлено на место размещения тега в тексте HTML страницы. Тег `` является встроенным строчным элементом.

Некоторые используемые атрибуты:

alt — обозначение альтернативного текста, выводимого на месте изображения в браузере в случае отключения загрузки изображений. Рекомендуемый атрибут.

title — текст подсказки, всплывающей при наведении указателя на изображение. Пример:

```

```

В данном примере изображение содержится в файле *flowers.jpg* в каталоге *images* на веб-сервере. Изображение загружается и выводится на экран браузера при отображении страницы.

Изображение также может быть содержимым гиперссылки. В этом случае

активация перехода происходит при щелчке мыши на изображении:

```
<a href="description.html"></a>
```

Для определения холста изображения используется контейнерный тег семантической разметки `<figure>`, в который могут включаться теги изображения ``, подписи `<figcaption>` и другие теги. Тег `<figcaption>` содержит текст подписи и также является контейнерным. Пример:

```
<figure>
  
  <figcaption>Цветы</figcaption>
</figure>
```

Дополнительное оформление фона и стиля страницы с помощью изображений, а также выравнивание, обтекание текстом, изменение размера изображений рекомендуется осуществлять с помощью таблиц CSS.

Векторная графика

Для вставки векторной графики в формате SVG на HTML страницу используется контейнерный тег `<svg>`, содержащий команды отрисовки объектов. На экране браузера тег SVG отображается в виде блочного объекта - холста. Ширина и высота холста может быть задана дополнительными атрибутами *width* и *height*.

Для задания ширины и цвета линий при отрисовке векторных объектов в командах используется атрибуты *stroke-width* — ширина линии в пикселах и *stroke* — цвет линии (в формате браузера), по умолчанию, черный. Атрибут *fill* задает цвет заполненной области (в формате браузера) или значение *none*, если заполнение не предусмотрено.

Список некоторых SVG команд и их атрибутов:

`<line>` - отображение прямой линии. Основные атрибуты:

x1 и *y1* - координаты начальной точки;

x2 и *y2* - координаты конечной точки;

`<polyline>` - отображение ломаной линии. Основные атрибуты:

points — пары координат опорных точек, разделенных запятыми и пробелами. Например: *points="0,50 50,0 100,50"*.

`<rect>` - отображение прямоугольника. Основные атрибуты:

x, *y* - координаты левого верхнего угла, *width* — ширина; *height* — высота.

`<circle>` - отображение окружности. Основные атрибуты:

cx и *cy* — координаты центра; *r* — радиус.

`<ellipse>` - отображение эллипса. Основные атрибуты:

cx и *cy* — координаты центра; *rx* — радиус по горизонтали; *ry* — радиус по вертикали.

Пример линии, красного треугольника и синей окружности:

```
<svg width="120" height="150">  
  <line x1="0" y1="30" x2="100" y2="30" stroke-width="1" stroke="#000"/>  
  <polyline points="0,50 50,0 100,50" fill="rgb(120,0,0)"/>  
  <circle r="50" cx="50" cy="100" fill="blue"/>  
</svg>
```

К объектам SVG также может быть применена трансформация. Для этого используются атрибуты *transform-origin* и *transform* вида: *transform-origin="x y" transform="функция()*. Вид и параметры функции аналогичны параметрам CSS трансформации, указанным в разделе 2.

1.5 Формы и элементы управления

Для взаимодействия клиента с веб-сервером в HTML используются клиентские формы. Форма, имеющая поля ввода и другие элементы управления, заполняется клиентом с помощью браузера, проверяется обработчиком Javascript на стороне клиента и затем отправляется на веб-сервер, где управление передается отдельному серверному приложению, написанному на одном из серверных скриптовых языков, например, PHP. Взаимодействие клиента с сервером осуществляется с помощью сетевого протокола HTTP (*HyperText Transfer Protocol*).

Для создания формы используется контейнерный тег *<form>*. Атрибут *action* указывает URL адрес серверного приложения. Если данные формы обрабатываются и отправляются на сервер сценарием Javascript, то в качестве значения атрибута *action* можно указать пустой URL адрес: *action="#"*.

Стандартные атрибуты формы:

name — имя формы. Используется для идентификации формы в Javascript.

method — метод HTTP отправки формы на сервер. Атрибут может принимать значения *GET* или *POST*. Если атрибут не задан, используется метод *GET*. В случае использования метода *GET* данные формы отправляются на сервер в строке с URL адресом серверного приложения после знака ? (вопрос) в виде пар: *имя=значение* разделенных символом & (амперсанд) и передаются управляющей программе. Имена определяются атрибутами *name* элементов управления, а значения вводятся пользователем или устанавливаются по умолчанию с помощью атрибутов *value*. Национальный алфавит и специальные символы кодируются в формате %HH, представляющие собой шестнадцатиричный код UTF-8. Пример:

http://htmlbook.ru/handler.php?nick=%C2%E0%ED%FF&page=5

В случае использования метода *POST* данные формы также передаются управляющей программе, но не в строке URL, а в содержании HTTP запроса:

http://htmlbook.ru/handler.php

nick=%C2%E0%ED%FF&page=5

enctype — способ кодирования данных формы. Атрибут может принимать значения: *application/x-www-form-urlencoded* для передачи данных, закодированных в UTF-8; *multipart/form-data* — для передачи некодированных бинарных данных (файлов); *text/plain* — для передачи некодированных текстовых данных. Если атрибут не задан, используется первый способ кодирования *application/x-www-form-urlencoded*.

Метки

Меткой является вспомогательный контейнерный тег `<label>`. Метки служат для связи дополнительной текстовой информации с элементами формы. Существует два способа связывания элемента формы и метки:

- использование атрибута *id* элемента формы и указания его имени в качестве атрибута *for* элемента `<label>`.
- элемент формы помещается внутрь контейнера `<label>`.

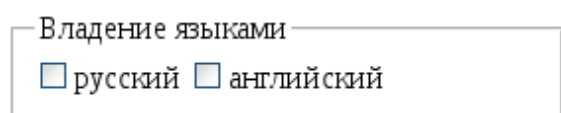
Элементы управления

К элементам управления формы относятся поля ввода, скрытые поля, поля для отправки файлов, области редактирования, выпадающие списки, переключатели, флажки и кнопки. Каждый элемент управления имеет свой вид отображения в браузере.

Поля ввода, скрытые поля, поля для отправки файлов, переключатели, флажки и кнопки создаются с помощью одиночного тега `<input>`. Тип элемента управления определяется значением атрибута *type*.

Для группировки элементов формы используется контейнерный тег `<fieldset>`. Такая группировка облегчает работу с формами, содержащими большое число данных. Элемент `<fieldset>` отображается браузером в виде рамки. Рамка может включать заголовочную надпись, определяемую содержимым вложенного контейнерного тега `<legend>`. Пример:

```
<fieldset>
  <legend>Владение языками</legend>
  <input type="checkbox">русский
  <input type="checkbox">английский
</fieldset>
```



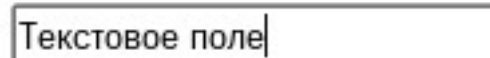
Владение языками

☐ русский ☐ английский

Текстовое поле ввода

Предназначение: ввод символов с клавиатуры. Пример:

```
<input name="family" type="text">
```



Текстовое поле

Некоторые используемые атрибуты:

value — значение поля по умолчанию.

size — ширина поля в моноширных символах.

maxlength — максимальное количество вводимых символов.

autofocus — устанавливает автофокус (атрибут без содержания).

form — имя формы. Используется для привязки отдельного элемента к заданной форме.

disabled — блокирует доступ и изменение элемента (атрибут без содержания).

readonly — блокирует изменение элемента (атрибут без содержания).

placeholder — задает текст подсказки, выводимой в поле.

required — проверяет заполненность вводимого поля (атрибут без содержания). В случае, если поле незаполнено, в браузер выводится предупреждающее сообщение.

pattern — устанавливает регулярное выражение в соответствии с ECMA-262 (*ECMAScript 2015 Language Specification*) для фильтрации вводимого текста. В браузере вводится только текст, разрешенный шаблоном. Примеры регулярных выражений: `\d` — одна цифра; `\w` — одна латинская буква; `\s` — один пробел; `[A-Я]` — один символ из диапазона; `\D` — не цифра; `\S` — не пробел; `^` - начало строки; `$` - конец строки; `.` - один любой символ; `.*` - любое количество любых символов; `^\S+` - первое слово; `\S+$` - последнее слово; `\d{3}` — три цифры; `\d{3,6}` — от трех до шести цифр; `\.` - десятичная точка; `[^0]` — не ноль; `(one|two)` — фраза one или two и т.д.

Поле ввода пароля

Предназначение: маскировка вводимых символов браузером. Пример:

`<input name="passw" type="password">`



Обратите внимание, что несмотря на маскировку символов браузером, при отправке данных на сервер значения полей ввода паролей передаются по протоколу HTTP в незашифрованном виде. Для передачи конфиденциальной информации используйте шифрование трафика SSL (*Secure Socket Layer*) и протокол HTTPS.

Виды и значения остальных атрибутов совпадают с текстовым полем ввода.

Скрытое поле ввода

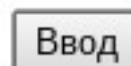
Предназначение: передача не отображаемых браузером данных на сервер. Может включать атрибут *value*, содержащий предустановленное значение. Пример:

`<input name="userid" type="hidden" value="135">`

Текстовая кнопка

Предназначение: связывает событие *onclick* элемента с обработчиком Javascript. Используется для предварительной обработки данных формы перед отправкой их на сервер. Пример:

`<input name="btn" type="button" value="Ввод">`



Некоторые используемые атрибуты:

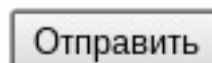
value – текст кнопки, отображаемый браузером;

onclick – функция обработчика события Javascript.

Кнопка отправки данных

Предназначение: отправка данных формы на сервер. Пример:

`<input name="btn" type="submit">`



Атрибуты те же, что и у обычной кнопки.

Кнопка сброса

Предназначение: возвращение данных формы в первоначальное состояние. Пример:

`<input name="btn" type="reset">`



Атрибуты те же, что и у обычной кнопки.

Флажок выбора опций

Предназначение: установка и снятие флажка опции. Используется при выборе нескольких элементов из предложенных. Если при отправке формы флажок не установлен, данные элемента не передаются. Пример использования:

`<label>Выберите теплые цвета:`

`<input name="opt1" type="checkbox">красный`

`<input name="opt2" type="checkbox">желтый`

`</label>`

Выберите теплые цвета:

☒ красный ☒ желтый

Некоторые используемые атрибуты:

value – значение элемента. Если этот атрибут отсутствует, то при выборе опции элемент принимает значение *оп*.

checked – делает флажок установленным по умолчанию.

disabled — блокирует доступ к элементу (атрибут без содержания). Независимо от состояния элемента данные не передаются.

required — проверяет наличие установленного флажка (атрибут без содержания).

Переключатель

Предназначение: выбор одного из нескольких предложенных элементов из группы. Все элементы переключателя должны иметь одинаковое имя *name*. Если при отправке формы переключатель не установлен, его данные не передаются. Пример использования:

`<label>Выберите верный ответ:`

`<input name="opt" type="radio" value="1">да`
`<input name="opt" type="radio" value="0">нет`
`</label>`

Выберите верный ответ:

☒ да ☐ нет

Атрибуты те же, что и у флажка выбора опций. Атрибут *checked* может быть только у одного элемента из группы.

В случае, если форма содержит несколько независимых переключателей, элементы в них группируются по одноименному атрибуту *name*.

Поле для отправки файлов

Предназначение: прикрепление содержания файла к отправляемой форме. Форма должна отправляться на сервер с помощью метода *POST*, способ кодирования данных *enctype* должен быть установлен как *multipart/form-data*. При нажатии на кнопку "Обзор" в браузере открывается диалог выбора файла. После выбора файла его имя появляется в поле ввода, а содержимое прикрепляется к форме и подготавливается для отправки на сервер. Пример использования:

`<input name="attach" type="file">`

Выберите файл payment_1...4669.pdf

Дополнительные атрибуты:

accept – устанавливает фильтр на MIME-типы прикрепленных файлов. При нескольких значениях MIME-типы перечисляются через запятую. Возможно обобщение типов, например, *image/** - для всех графических файлов.

multiply – позволяет указывать одновременно несколько файлов в поле (атрибут без содержания).

maxlength, *size* и *value* аналогичные текстовому полю ввода.

Поле электронной почты

Предназначение: указание адреса электронной почты. Адрес проверяется на корректность ввода. Пример: `<input name="email" type="email">`

Поле ввода чисел

Предназначение: ввод целых чисел. Содержит кнопки-стрелки вверх-вниз, изменяющие введенное значение на один шаг. Пример:

`<input name="num" type="number">`

Дополнительные атрибуты: *value* – значение по умолчанию; *min* – минимальное значение; *max* – максимальное значение; *step* – шаг приращения (по умолчанию, 1).

Поле выбора чисел из диапазона

Предназначение: выбор целых чисел из диапазона. Отображает ползунок, изменяющий значение на один шаг. Пример:

`<input name="num" type="range">`

Дополнительные атрибуты: *value* – значение по умолчанию; *min* – минимальное значение (по умолчанию, 0); *max* – максимальное значение (по умолчанию, 100); *step* – шаг приращения (по умолчанию, 1).

Область редактирования

Предназначение: ввод многострочного текстового содержимого, предоставление возможности онлайн редактирования. Область образуется с помощью контейнерного тега `<textarea>`. Пример использования:

`<textarea name="content">`

Область редактирования

`</textarea>`



Дополнительные атрибуты:

cols — ширина текстовой области в символах моноширного текста.

rows — высота текстовой области в строках, отображаемых без прокрутки содержимого.

wrap – режим переноса на следующую строку. Может принимать одно из трех значений: *physical*, *virtual* и *off*.

disabled, *readonly*, *placeholder* и *required* — аналогичные текстовому полю ввода.

Выпадающий список

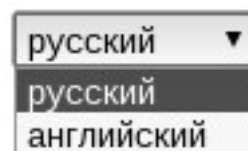
Предназначение: выбор одного или нескольких элементов из заданного списка. Список задается контейнерным тегом `<select>`, включающим контейнерные элементы `<option>` с текстовым содержимым. При выборе элемента из списка, его значение *value* устанавливается для контейнера `<select>` и передается на сервер. Пример использования:

`<select name="lang">`

`<option value="RU">русский</option>`

`<option value="EN">английский</option>`

`</select>`



Дополнительные атрибуты тега `<select>`:

multiple – возможность множественного выбора элементов списка (атрибут без содержания). Множественный выбор элементов осуществляется с участием нажатой клавиши `<Ctrl>`.

size – определяет высоту списка в элементах (по умолчанию, 1).

disabled — блокирует выбор элементов (атрибут без содержания).

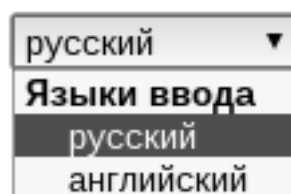
required — проверяет выбран ли какой-либо элемент из списка (атрибут без содержания). В случае, если элемент не выбран, в браузер выводится предупреждающее сообщение.

value – значение элемента списка. Атрибут относится к тегу `<option>`. Если атрибут не установлен, то значение выбранного элемента определяется его содержимым.

selected – выделение элемента списка по умолчанию (атрибут без содержания). Атрибут относится к одному или нескольким (в случае множественного выбора) тегам `<option>`.

Элементы списка могут быть сгруппированы в иерархию с помощью специального контейнерного тега `<optgroup>`. Атрибут *label* используется для обозначения заголовка группы списка. Заголовок группы отображается выделенным текстом в браузере и является невыбираемым элементом. Пример использования:

```
<select name="lang">
  <optgroup label="Языки ввода">
    <option value="RU">русский</option>
    <option value="EN">английский</option>
  </optgroup>
</select>
```



Комбинированный список

Предназначение: выбор при наборе в текстовом поле. Список задается контейнерным тегом `<datalist>`, включающим контейнерные элементы `<option>` с текстовым содержимым. С помощью атрибута *id* тега `<datalist>` список связывается с текстовым полем `<input>`, имеющим соответствующий атрибут *list*. Список становится доступным при получении полем фокуса или при наборе текста. Пример:

```
<input name="lang" list="lang">
<datalist id="lang">
  <option value="RU">русский</option>
  <option value="EN">английский</option>
</datalist>
```



2 Каскадные таблицы стилей CSS

2.1 Синтаксис CSS правил

Каскадной таблицей стилей CSS (Cascading Style Sheets) называется формальный язык описания внешнего вида документа, написанного с использованием языка разметки (HTML). CSS представляет собой набор правил - параметров форматирования, которые применяются к элементам документа, чтобы изменить их внешний вид. Термин «каскадные таблицы стилей» впервые был предложен Хокон Виум Ли в 1994 году. В декабре 1996 года консорциумом W3C была предложена первая редакция CSS, она включала управление шрифтами и цветом, задание межсимвольных и междустрочных интервалов,

выравнивание и отступы для абзацев, таблиц и других элементов. Поддержка CSS впервые была реализована в браузерах Internet Explorer 4 и Netscape Navigator 4, что позволило значительно упростить верстку HTML документов. В версии CSS 2, принятой в мае 1998 года были добавлены блочная верстка, типы носителей, селекторы, указатели и генерируемое содержимое. В рассматриваемой в настоящее время спецификации CSS 3 добавлено управление отображением векторной графики SVG, трансформацией и анимацией без использования языка Javascript.

Стилизация страницы обычно применяется после разработки структуры и наполнения ее содержанием средствами HTML. Параметры форматирования объединяются в правила вида:

```
селектор1, селектор2, ... {  
    свойство1: значение1;  
    свойство2: значение2; ...  
}
```

Селекторы определяют теги, к которым применяется данное правило. Каждый селектор может применяться к одному или нескольким тегам веб-страницы. CSS правило может определяться в трех различных местах (в порядке уменьшения значимости):

- Локальным. Стиль определяется в атрибуте *style* тега. Переопределяет внешние и страничные стили. Правило распространяется только на текущий тег. Например:

```
<div style="text-align: center">Текст по центру</div>
```

- Страничным. Стиль определяется внутри контейнерного тега вида: `<style type="text/css">`, размещаемого в заголовке HTML документа. Переопределяет внешние стили. Правило распространяется на весь HTML документ. Например:

```
<style type="text/css">  
    div { text-align: center; }  
</style>
```

Дополнительный атрибут *media* служит для определения класса устройств, к которым применимы данные стили, например: *screen* – экран компьютера (по умолчанию), *print* – принтер, *handheld* – смартфон.

- Внешним. Стиль определяется во внешнем CSS файле. Внешний файл подключается в заголовке страницы с помощью тега вида:

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

где *styles.css* – имя внешнего CSS файла.

Определение стилей

При загрузке веб-страницы сначала загружаются внешние CSS файлы и к тегам веб-страницы применяются все определенные в них стили, затем загружаются

страничные стили, наконец, последними применяются локальные стили. При этом более специфичные правила переопределяют менее специфичные. Таким образом, все правила, описанные в каскадных таблицах стилей применяются последовательно ко всем встречаемым тегам, дополняя и переопределяя их свойства. Входящие в правила *селекторы* могут состоять из имен тегов, классов, идентификаторов, псевдоклассов и псевдоэлементов.

Имена тегов применяют, когда необходимо определить стиль для отдельных тегов веб-страницы. Для этого используется следующий синтаксис правил:

тег { свойство: значение; ... }. Пример описания CSS стиля:

div { text-align: center; } / Для всех тегов div */*

При этом структура веб-страницы остается неизменной, а все содержание тегов *div* будет выровнено по центру:

<div>Текст по центру</div>

Группирование селекторов. Когда на веб-странице одновременно используется множество селекторов, возможно появление повторяющихся стилевых правил. Чтобы не повторять правила для каждого селектора, их можно сгруппировать для удобства представления и сокращения кода. Для группировки селекторов их наименования перечисляются через запятую:

селектор1, селектор2, ... { свойство: значение; ... }. Примеры стилей:

p, div { text-align: center; } / Для всех тегов p и div */*

h1, h2, h3 { font-family: Arial, Helvetica, sans-serif; } / Для заголовков */*

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 { свойство: значение; ... }. Пример стиля:

p.cite { color: red; } / Абзацы с классом cite */*

Для применения классов к некоторым абзацам веб-страницы для соответствующих тегов *<p>* необходимо указать атрибут *class*:

<p class="cite">Красный цвет текста</p>

Можно использовать классы и без указания тега:

.имя_класса { свойство: значение; ... }. Пример стиля:

.cite { margin-left: 20px; } / Для всех тегов */*

При такой записи класс можно применять к любому тегу:

<div class="cite">Отступ слева 20px</div>

Классы удобно использовать, когда нужно применить стиль к разным элементам веб-страницы: ячейкам таблицы, ссылкам, абзацам и др., например:

div { background-color: #fff; } / Цвет фона белый*/*

div.odd { background-color: #ccc; } / Цвет фона серый*/*


```
<div>белый цвет фона</div>
<div class="odd">серый цвет фона</div>
```

К любому элементу можно добавить несколько классов, описываемых отдельными правилами. Такие классы называются *мультиклассами*. В этом случае классы перечисляются в атрибуте class через пробел:

```
<div class="cite odd">Отступ слева и серый цвет фона</div>
```

Для выделения таких тегов можно описать отдельное правило, указав селекторы классов без пробелов:

```
.cite.odd { margin-left: 20px; background-color: #ccc; }
```

Идентификаторы (ID селекторы) определяют уникальные имена элементов, которые используются для изменения их стилей и обращений к ним посредством Javascript. Синтаксис применения идентификатора следующий.

#имя_идентификатора { свойство: значение; ... }. Пример стиля:

```
#help { width: 300px; background-color: gray; }
```

Для применения стиля идентификатора к элементу веб-страницы в соответствующем теге необходимо указать атрибут *id*:

```
<div id="help">Как поймать льва в пустыне?</div>
```

Так же как и при использовании классов, идентификаторы можно применять к конкретному тегу:

тег#имя_идентификатора { свойство: значение; ... }. Пример стиля:

```
div#help { width: 300px; background-color: gray; }
```

Контекстные селекторы используются для выделения тегов из сложной структуры веб-страницы и применения к ним стилизации. Простой контекстный селектор состоит из нескольких селекторов, разделенных пробелом, что соответствует вложенности тегов на веб-странице:

селектор1 селектор2 { свойство: значение; ... }. Пример стиля:

```
p div { background-color: #ccc; }
```

В этом случае стиль будет применяться ко всем тегам *<div>*, размещенным внутри тегов *<p>*. Уровень вложенности тегов при этом значения не имеет.

Соседние селекторы применяются к непосредственно следующим друг за другом тегам на одном уровне иерархии веб-страницы. Для управления стилями соседних элементов используется символ + (плюс) между селекторами:

селектор1 + селектор2 { свойство: значение; ... }

Соседние селекторы удобно использовать для тех тегов, к которым автоматически добавляются отступы, чтобы самостоятельно регулировать их величину. Например, расстояние между подряд идущими тегами *<h1>* и *<h2>* можно регулировать с помощью соседних селекторов:

```
h1 + h2 { margin-top: -10px; } /* Смещение заголовка 2 вверх */
```

Родственные селекторы применяются к любым, подряд идущим слева направо тегам, размещаемые на одном уровне иерархии (имеющие общего родителя). Для управления стилями родственных элементов используется символ ~ (тильда) между селекторами:

селектор1 ~ селектор2 { свойство: значение; ... }

h1 ~ h2 { margin-top: -10px; } / Сдвиг всех заголовков 2 вверх */*

Дочерние селекторы применяются к непосредственно вложенным друг в друга тегам. Для управления стилями дочерних элементов используется символ > (больше) между селекторами:

селектор1 > селектор2 { свойство: значение; ... }

ul > li { list-style: none; } / Убрать маркеры списка */*

Селекторы атрибутов используются для выделения тегов, имеющих специфичные атрибуты:

[атрибут] { свойство: значение; ... } или

селектор[атрибут] { свойство: значение; ... }. Пример стиля:

q[title] { color: red; }

В результате к тегам <q> имеющим атрибут *title* применяется красный цвет.

Для выделения любых тегов, имеющих специфичные атрибуты имя тега можно не указывать:

[title] { color: red; } / Для любого тега с атрибутом title */*

Для выделения тегов, имеющих определенные значения атрибутов используется следующее правило:

селектор[атрибут="значение"] { свойство: значение; ... }, например:

a[target="_blank"] { color: red; } / Для ссылки в новом окне */*

Также для других видов содержания атрибутов:

селектор[атрибут^="значение"] — атрибут начинается со значения;

селектор[атрибут\$="значение"] — атрибут заканчивается значением;

селектор[атрибут="значение"]* — атрибут включает символы значения;

селектор[атрибут~="значение"] — атрибут включает слово значение;

Все перечисленные методы можно комбинировать между собой, определяя стиль для элементов, которые содержат два и более атрибута. В подобных случаях квадратные скобки идут подряд, например:

[атрибут="значение"][атрибут="значение"] { свойство: значение; ... }

Универсальный селектор соответствует любому элементу веб-страницы. Для обозначения универсального селектора применяется символ * (звёздочка):

** { свойство: значение; ... }*. Пример стиля:

** { margin: 0; padding: 0; } /* Для всех элементов */*

Псевдоклассы и псевдоэлементы

Псевдоклассы определяют динамическое состояние элементов, которое изменяется с помощью действий пользователя, а также положение в дереве HTML документа. Псевдоклассы чаще всего применяются к кнопкам, гиперссылкам и другим активным элементам. Общий синтаксис:

селектор:псевдокласс { свойство: значение; ... }. Пример стиля:

a:active { color: #f00; } / Цвет активной ссылки */*

При использовании псевдоклассов браузер не перегружает текущий документ, это применяется для получения различных динамических эффектов на странице.

Допускается применять псевдоклассы к именам идентификаторов или классов, а также к контекстным селекторам. Если псевдокласс указывается без селектора впереди (:hover), то он будет применяться ко всем элементам документа. Примеры стилей CSS:

*a.menu:hover { color: green; }
.menu a:hover { background-color: #fc0; }
:hover { color: yellow; }*

Условно все псевдоклассы делятся на три группы:

- определяющие состояние элементов;
- имеющие отношение к дереву элементов;
- указывающие язык текста.

Псевдоклассы, определяющие состояние элементов. К этой группе относятся псевдоклассы, которые распознают текущее состояние элемента и применяют стиль только для этого состояния.

:active - происходит при активации пользователем элемента. Например, ссылка становится активной, если навести на неё курсор и щёлкнуть мышкой. Несмотря на то, что активным может стать практически любой элемент веб-страницы, псевдокласс *:active* используется преимущественно для ссылок.

:link - применяется к непосещенным ссылкам, т. е. таким ссылкам, на которые пользователь ещё не нажимал. Браузер некоторое время сохраняет историю посещений, поэтому ссылка может быть помечена как посещенная хотя бы потому, что по ней был зафиксирован переход ранее.

Запись *a {...}* и *a:link {...}* по своему результату равноценна, поскольку в браузере даёт один и тот же эффект, поэтому псевдокласс *:link* можно не указывать. Исключением являются якоря, на них действие *:link* не распространяется.

:focus - применяется к элементу при получении им фокуса. Например, для текстового поля формы получение фокуса означает, что курсор установлен в поле, и с помощью клавиатуры можно вводить в него текст.

:hover — активизируется при наведении на элемент курсора мыши, но без нажатия на кнопку, например:

```
tr:hover { background-color: #fc0; } /* Изменение цвета фона строки */
```

:visited - применяется к посещённым ссылкам. Обычно такая ссылка меняет свой цвет по умолчанию на фиолетовый, но с помощью стилей цвет и другие параметры можно задать самостоятельно.

Селекторы могут содержать более одного псевдокласса, которые перечисляются подряд через двоеточие, но только в том случае, если их действия не противоречат друг другу:

a:visited:hover

Псевдоклассы, имеющие отношение к дереву документа. К этой группе относятся псевдоклассы, которые определяют положение элемента в дереве документа и применяют к нему стиль в зависимости от его статуса.

:first-child - применяется к первому элементу селектора, который расположен в дереве элементов документа, например:

```
b:first-child { color: red; } /* Термин выделен красным цветом */
```

<p>Дихотомический метод заключается в разбиении выделенной области решений пополам</p>

:last-child - применяется к последнему элементу селектора, расположенного в дереве элементов документа.

:only-child - применяется к первому элементу селектора, расположенного в дереве элементов документа, только если он единственный.

:nth-child(n) - применяется к *n*-му порядковому элементу селектора, расположенного в дереве элементов документа. Параметр *n* также может принимать значения *even* (все четные элементы), *odd* (все нечетные элементы) и *an+b*, где *a* и *b* - произвольные целые числа. Псевдокласс часто используется при оформлении таблиц. Пример - задание цвета фона всех четных строк:

```
tr:nth-child(even) { background: #f0f0f0; }
```

:not(селектор) задаёт правила стилей для элементов, которые не содержат указанный селектор. Пример — задание цвета фона для всех абзацев, кроме именованного с атрибутом *id="main"*:

```
p:not(#main) { background: yellow; }
```

Псевдоклассы, задающие язык текста. С помощью псевдоклассов можно изменять стиль оформления иностранных текстов, а также некоторые настройки, характерные для разных языков, например, вид кавычек в цитатах.

:lang - определяет язык, который используется в документе или его фрагменте. На веб-странице язык устанавливается через атрибут *lang*, он обычно добавляется к тегу *<html>*:

```
Элемент:lang(язык) { свойство: значение; ... }
```

В качестве языка могут выступать следующие значения: *ru* — русский; *en* — английский; *de* — немецкий ; *fr* — французский; *it* — итальянский. Пример:

```
q:lang(ru) {  
  quotes: "\00AB" "\00BB"; /* Кавычки для русского языка */  
}
```

Псевдоэлементы позволяют задать стиль элементов, не определенных в дереве элементов документа, а также генерировать содержимое, которого нет в исходном коде текста. Синтаксис использования псевдоэлементов:

Селектор::Псевдоэлемент { свойство: значение; ... }

Каждый псевдоэлемент может применяться только к одному селектору.

::after - применяется для вставки назначенного контента после содержимого элемента. Этот псевдоэлемент работает совместно со стилевым свойством *content*, которое определяет содержимое для вставки.

```
p.new::after {  
  content: " - Новьё!"; /* Добавляет текст после абзаца */  
}
```

::before - аналогичен псевдоэлементу **::after**, но вставляет контент до содержимого элемента.

```
li::before {  
  content: "\20aa "; /* Добавляет перед элементом символ юникода*/  
}
```

::first-letter - определяет стиль первого символа в тексте элемента, к которому добавляется. Это позволяет создавать в тексте буквицу.

::first-line - определяет стиль первой строки блочного текста. Длина этой строки зависит от многих факторов, таких как используемый шрифт, размер окна браузера, ширина блока, языка и т.д.

Наследование правил и специфичность

Наследованием называется перенос правил форматирования для элементов, находящихся внутри других. Такие элементы являются дочерними, и они наследуют некоторые стилевые свойства своих родителей, внутри которых располагаются. Например, если в стилях для селектора *table* задать цвет текста, то он автоматически устанавливается для содержимого всех ячеек таблицы:

```
table { color: red; }
```

Необходимо отметить, что наследуются не все свойства элементов. Перечень наследуемых свойств элементов приведен в справочнике [1].

Приоритеты наследования (в сторону увеличения):

1. Стиль браузера.
2. Стиль автора.
3. Стиль пользователя.

4. Стиль автора с добавлением *!important*.
5. Стиль пользователя с добавлением *!important*.

Если к одному элементу одновременно применяются противоречивые стилевые правила, то более высокий приоритет имеет правило, у которого значение специфичности селектора больше.

Для определения специфичности используется следующее правило: за каждый идентификатор начисляется 100 баллов, за каждый класс и псевдокласс начисляется 10 баллов, за каждый селектор тега и псевдоэлемент начисляется 1 балл. Складывая указанные значения, получаем значение специфичности для каждого селектора.

Встроенный стиль, добавляемый к тегу через атрибут *style*, имеет специфичность 1000, поэтому всегда перекрывает связанные и глобальные стили. Однако добавление к стилю свойства *!important* перекрывает в том числе и встроенные стили. Пример:

```
#menu ul li { color: #000; } /* специфичность 102 */
.two { color: red; } /* специфичность 10 */

<div id="menu">
  <ul>
    <li>Первый элемент</li>
    <li class="two">Второй элемент</li>
  </ul>
</div>
```

В данном примере правило *.two* имеет меньшую специфичность и перекрывается первым правилом, следовательно, оба элемента списка будут отображены черным цветом. Для выделения второго элемента можно изменить первое правило, убрав идентификатор и тем самым понизив специфичность:

```
ul li { color: #000; } /* специфичность 2 */
```

Другим способом выделения второго элемента является добавление идентификатора ко второму правилу и повышение его специфичности:

```
#menu .two { color: red; } /* специфичность 110 */
```

Так как стилей может быть много, они могут размещаться в различных местах документа и даже в разных файлах, сопоставить правила бывает непросто. Альтернативным решением является использование свойства *!important*, что сразу приводит к желаемому результату.

```
.two { color: red !important; } /* Добавляем свойство !important */
```

@-правила используются для подключения дополнительных свойств.

@font-face { свойства } — определение шрифта с заданными свойствами и его загрузка. Имя файла шрифта задается с помощью параметра *src*. Например:

```
@font-face {
  font-family: Pompadur; /* Имя шрифта */
```

```
src: url(font/pompadur.ttf); /* Путь к файлу со шрифтом */
}
```

@media носитель { } — определение стиля для заданного типа устройств:

```
@media screen { /* Стиль для отображения в браузере */
  body {
    font-family: Arial, Verdana, sans-serif; /* Рубленый шрифт */
    font-size: 12pt; /* Размер шрифта */
  }
}
```

@page :поле { margin: размер } — определение размеров полей для печати страниц, заданных типом носителя *print*. Параметр *:поле* может иметь одно из трех значений - *:left* — для левых страниц, *:right* — для правых страниц или *:first* — для первой страницы, например:

```
@page :first {
  margin: 1cm; /* Отступы для первой страницы */
}
```

@keyframes или **@-webkit-frames** (для браузера Google Chrome) — определяют ключевые кадры анимации. Общий формат:

```
@keyframes <переменная> { from { стиль } to { стиль } } или
@keyframes <переменная> { nn% { стиль } nn% { стиль } }, где
```

параметр *<переменная>* связывает ключевой кадр с атрибутом *animation*;

параметр *from* задает начальный стиль анимированного элемента (0%), а параметр *to* — конечный стиль анимированного элемента (100%). Начальный и конечный стили могут также устанавливаться в виде процентов. Пример:

```
@keyframes fadeIn {
  0% { opacity: 0; } /* уровень прозрачности 0 */
  100% { opacity: 1; } /* уровень прозрачности 1 */
}
```

2.2 Стилизация текста

Параметры шрифта

Для стилизации текста могут использоваться как универсальные, так и специальные свойства, задающие характеристики шрифта и текста. По умолчанию все стили шрифта принимают значение *inherit* (наследуемое). Стиль шрифта может быть задан универсальным составным свойством *font*. Упрощенный вид свойства:

```
font: [<наклон>] [<строч>] [<толщина>] [<размер> [/h_стр]] <имя>
```

Обязательно указывается имя шрифта. например:

```
p { font: bold italic 14pt sans-serif; }
```

Для абзаца значение *bold* устанавливает жирное начертание, *italic* — курсив, размер шрифта *14pt* и семейство шрифтов *sans-serif*. Каждое из этих свойств может быть задано отдельной строкой, например:

```
p { font-style: italic;
    font-weight: bold;
    font-size: 12pt;
    font-family: sans-serif; }
```

Имена шрифтов *<имя>* устанавливаются атрибутом *font-family* в виде списка, разделенного запятыми. Браузер выбирает первый шрифт и если он отсутствует на компьютере, выбирается следующий из списка. В случае указания семейства шрифтов, выбирается один шрифт из семейства: *serif* (с засечками), *sans-serif* (без засечек), *cursive* (рукописные), *fantasy* (декоративные) и *monospace* (моноширные). Если имя шрифта содержит пробелы, то оно заключается в кавычки. Пример:

```
h1 { font-family: "Times New Roman", serif }
```

Параметр *<наклон>* устанавливается атрибутом *font-style*. Возможные значения: *normal* (по умолчанию), *italic* (курсив) и *oblique* (небольшой наклон).

Параметр *<строч>* устанавливается атрибутом *font-variant*. Возможные значения: *normal* (по умолчанию), *small-caps* - малые строчные буквы.

Толщина шрифта *<толщина>* устанавливается атрибутом *font-weight*. Возможные значения выражаются числами от 100 до 900 по возрастанию степени выделения, а также константами *normal* (400, по умолчанию) или *bold* (700). Также можно указать выделение с уменьшением *lighter* или увеличением *bolder* относительно толщины текущего шрифта.

Размер шрифта *<размер>* устанавливается атрибутом *font-size*. Размеры шрифтов, а также других элементов CSS, можно задать в абсолютных и относительных единицах: *px* — пикселах, *pt* — типографских пунктах, *cm* — сантиметрах, *mm* — миллиметрах, *em* — размера буквы *m* текущего шрифта, % - в процентах от размера родительского элемента. Всего определено 7 стандартных градаций размеров шрифта, определяемых константами: *xx-small*, *x-small*, *small*, *medium*, *large*, *x-large* и *xx-large*. Также можно указать размер с уменьшением *smaller* или увеличением *larger* относительно текущего шрифта.

Высота строки текста *<h_cmp>* устанавливается с помощью атрибута *line-height*. При этом можно задать как абсолютное, так относительное значение. Пример двойной высоты строки:

```
p { line-height: 2 }
```

Атрибут стиля *color* задает цвет текста. Цвет в CSS может задаваться в виде имени, полных *#НННННН* или сокращенных *#ННН* шестнадцатиричных чисел, а также в виде функций *rgb(r, g, b)* или *rgba(r, g, b, a)*. Параметры *r*, *g*, *b* задают уровни красного, зеленого и синего цветов (от 0 до 255), а также уровня

прозрачности *α* (от 0 до 1), например:

```
h1 { color: #f00 } /* Красный цвет. То же: red, #ff0000, rgb(255,0,0) */
```

Уровень прозрачности также может быть задан отдельно свойством *opacity*:

```
div { opacity: 0.5 }
```

Атрибут *text-decoration* задает дополнительное оформление шрифта: *underline* (подчеркивание), *line-through* (зачеркивание) или *none* — отмена:

```
a:link { text-decoration: none }
```

Атрибут *text-transform* изменяет регистр символов текста: *uppercase* — верхний регистр, *lowercase* — нижний регистр, *capitalize* — первая заглавная буква, *none* — отмена стиля.

Для указания дополнительного расстояния между символами текста используется атрибут *letter-spacing*. Для отображения сжатого текста межсимвольный интервал можно указать отрицательным:

```
.cond { letter-spacing: -2px }
```

Для указания дополнительного расстояния между отдельными словами текста используется атрибут *word-spacing*:

```
.wide { word-spacing: 3px }
```

Для управления разрывом строк используется атрибут *white-space*, задающий правила отображения и переноса пробелов. Возможные значения: *normal* — лишние пробелы игнорируются, строки переносятся автоматически; *pre* — сохранение форматирования текста; *nowrap* — строки не переносятся; *pre-wrap* — сохранение форматирования, длинные строки переносятся; *pre-line* — сохранение только переводов строк, длинные строки переносятся.

Значение *break-word* атрибута *word-wrap* позволяет переносить длинные слова в предложениях.

Параметры абзацев

Для блочных элементов можно задать параметры горизонтального и вертикального выравнивания. Для горизонтального выравнивания текста используется атрибут *text-align*. Возможные значения: *left* — по левому краю (по умолчанию), *right* — по правому краю, *center* — по центру, *justify* — по ширине. Кроме этого, можно задать отступ для красной строки абзаца с помощью атрибута *text-indent*, например:

```
p { text-indent: 1cm }
```

Для вертикального выравнивания элементов используется атрибут *vertical-align*. Он определяет выравнивание текущего фрагмента текста относительно родительского элемента. Возможные значения: *baseline* — по базовой линии текста (по умолчанию); *top* — по верхней границе; *middle* — по центру; *bottom* — по нижней границе; *text-top* и *text-bottom* — по верхней или нижней границе текста; *sub* и *super* — по базовой линии нижнего или верхнего

индекса. Кроме текстовых фрагментов вертикальное выравнивание может применяться к вложенным в контейнеры *inline* элементам: изображениям, содержимому таблиц и т.д.

Для оформления тени у текста можно использовать атрибут *text-shadow*:

text-shadow: <цвет> <смещ_х> <смещ_у> [<размытие>]

Горизонтальное и вертикальное смещение может быть задано отрицательным числом, например:

h1 { text-shadow: gray 2px -2px 1px }

Параметры фона

Фон можно указывать для фрагмента текста, блочного элемента, таблицы, строки или ячейки таблицы или для всей страницы. Для задания фона элемента обычно указывается цвет или фоновое изображение.

Атрибут *background-color* устанавливает цвет фона. Значение атрибута *transparent* устанавливает прозрачный фон. По умолчанию цвет страницы задается браузером. Пример задания желтых букв на синем фоне:

body { color: yellow; background-color: blue }

Атрибут *background-image* устанавливает фоновое изображение. Возможные значения атрибута: *none* (по умолчанию) или *url(URL_адрес_изображения)*. Форматы GIF и PNG обеспечивают поддержку прозрачности. Пример:

*body { background-color: yellow;
background-image: url("images/fon.png") }*

В данном случае через прозрачный фон изображения будет просвечивать желтый цвет, установленный атрибутом *background-color*.

В случае, если изображение меньше, чем элемент страницы, в браузере оно будет повторяться по вертикали и по горизонтали начиная с левого верхнего угла элемента. Параметры повторения изображения задаются с помощью атрибута *background-repeat*. Возможные значения: *repeat* (с повторением), *no-repeat* (без повторения), *repeat-x* (повторение по горизонтали), *repeat-y* (повторение по вертикали).

Для указания начальной позиции изображения в элементе используется атрибут *background-position*:

background-position: <смещ_х> [<смещ_у>]

Значение *смещ_х* указывает смещение изображения относительно элемента по горизонтали в указанных единицах измерения. Также может принимать значения: *left* (слева), *center* (по центру) или *right* (справа). Значение *смещ_у* указывает смещение изображения относительно элемента по вертикали в указанных единицах измерения. Также может принимать значения: *top* (сверху), *center* (по центру) или *bottom* (снизу). В случае указания только горизонтального смещения, по вертикали изображение выравнивается по

центру. Пример:

```
p { background-position: 1cm top }
```

В данном примере фон абзаца фиксируется на 1 см от левого края и выравнивается по верхней границе элемента.

Для управления прокруткой фонового изображения используется атрибут *background-attachment*. Возможные значения: *scroll* (по умолчанию) — браузер прокручивает фоновое изображение вместе с содержимым страницы, *fixed* — фон фиксируется и не прокручивается.

Параметры списков

Для задания вида маркеров или нумерации пунктов списка используется атрибут *list-style-type*. Возможные значения для маркированных списков: *disk* — маркер в виде символа круга (по умолчанию), *circle* — маркер в виде символа окружности, *square* — маркер в виде квадрата. Возможные значения для нумерованных списков: *decimal* — десятичная нумерация (по умолчанию), *decimal-leading-zero* — десятичная с начальным нулем, *lower-alpha* и *lower-latin* — строчные латинские буквы, *upper-alpha* и *upper-latin* — заглавные латинские буквы, *lower-roman* и *upper-roman* — строчные и заглавные римские цифры. Для отмены маркировки и нумерации списка используется значение *none*.

Для задания графического изображения вместо маркера списка используется атрибут *list-style-image* со значением *url(URL_адрес_изображения)*. Пример:

```
ul { list-style-image: url("images/ball.png") }
```

2.3 Табличная и блочная верстка

Текстовая и фреймовая верстка

На заре Интернет, до 1995 года, использовалась только *текстовая* верстка. HTML страницы представляли собой набор текстовых абзацев с редкими вкраплениями изображений и таблиц. Навигация осуществлялась с помощью гиперссылок, размещаемых в начале и в конце каждой страницы. Такие страницы достаточно быстро загружались в браузере, но при размещении большого объема информации быстро стали неудобными. Элементы навигации потребовалось размещать таким образом, чтобы они всегда оставались доступными, а обновлялось бы только содержимое.

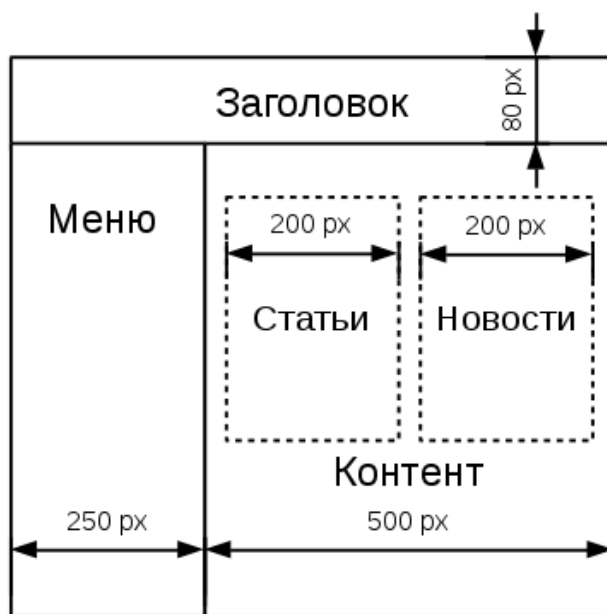
В 1995 году в браузере Netscape Navigator появилась поддержка *фреймов*. При использовании фреймовой верстки экран браузера делился на две или три части и в каждый из них загружался отдельный HTML документ, содержащий например, логотип сайта, боковое меню или содержимое — контент. Достоинство использования фреймов — увеличение скорости загрузки страниц, так как в части фреймов содержимое не обновлялось. В качестве недостатка можно отметить громоздкость и негибкость конструкции, невозможность изменения заданной разбивки страницы.

Табличная верстка

С ростом скорости Интернет и появлением средств стилизации необходимость во фреймовой верстке отпала и в начале XXI века его полностью заменила *табличная* верстка. Пример табличного дизайна — оформление сайта в виде газетной полосы. Текст разбивается на две или три колонки и стилизуется. Для позиционирования ячеек таблиц используется фиксированная или плавающая верстка. При этом активно используются вложенные таблицы с невидимыми границами.

Фиксированная табличная верстка обеспечивается с помощью фиксации размера таблиц и ячеек. Пример фиксированной табличной верстки:

```
<table>
  <tr colspan="2">
    <td height="80">Заголовок</td>
  </tr>
  <tr>
    <td width="250">Меню</td>
    <td width="500">
      <table border="0">
        <tr>
          <td width="200">Статьи</td>
          <td width="200">Новости</td>
        </tr>
      </table>
      Контент
    </td>
  </tr>
</table>
```



Достоинства фиксированной верстки:

- сайт одинаково выглядит во всех браузерах;
- можно позиционировать содержимое как угодно, даже разрезать изображения и загружать их в разные ячейки таблицы.

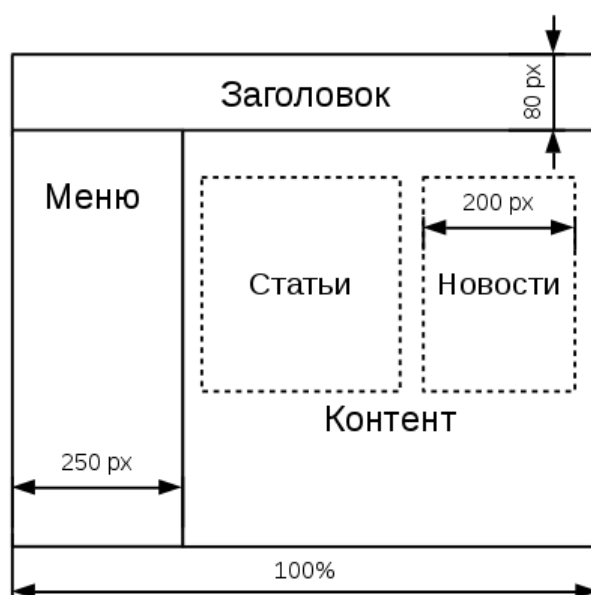
Недостаток фиксированной верстки — при использовании на экранах с большим разрешением остается много свободного места, а на маленьких экранах, наоборот, может появиться прокрутка. Альтернативой фиксированной верстки, лишенной этого недостатка, явилась плавающая верстка.

Плавающая табличная верстка обеспечивается с помощью фиксации размера таблиц в процентах и лишь некоторых ячеек в пикселах, например, высоты логотипа и ширины меню. При этом размеры оставшихся ячеек автоматически раздвигаются до размера родительского элемента, который определяется размером окна браузера. Плавающая табличная верстка многие годы являлась стандартом сайтостроения, пока не была вытеснена более современной блочной версткой. Пример плавающей табличной верстки:

```

<table width="100%">
  <tr colspan="2">
    <td height="80">Заголовок</td>
  </tr>
  <tr>
    <td width="250">Меню</td>
    <td>
      <table border="0" width="100%">
        <tr>
          <td>Статьи</td>
          <td width="200">Новости</td>
        </tr>
      </table>
      Контент</td>
    </tr>
  </table>

```



Недостатки табличного дизайна:

- большой объем и время загрузки страниц, содержащих большое количество однотипных табличных тегов;
- сложный и избыточный табличный код плохо поддается редактированию и индексации поисковыми роботами.

Блочная верстка

С развитием стилизации и стандарта CSS появилась возможность использовать блочную верстку практически для всех элементов страниц. Это существенно упростило структуру сайтов и позволило отделить содержание страниц от их представления. Основным структурным элементом блочной верстки являются контейнерные элементы div. С помощью стилей обеспечивается выравнивание и позиционирование контейнеров относительно друг друга. Вышеприведенный пример, оформленный с помощью блочной верстки:

```

header { height: 80px; } /* высота 80 px */
.menu { width: 250px; float: left; } /* ширина 250 px, обтекание слева */
.content { padding-left: 250px; } /* отступ слева 250 px */
.news { width: 200px; float: right; } /* ширина 200 px, обтекание справа */
.articles { padding-right: 200px; } /* отступ справа 200 px */

```

```

<body>
  <header>Заголовок</header>
  <div class="menu">Меню</div>
  <div class="content">
    <div class="news">Новости</div>
    <div class="articles">Статьи</div>
  </div>
</body>

```

Параметры размеров

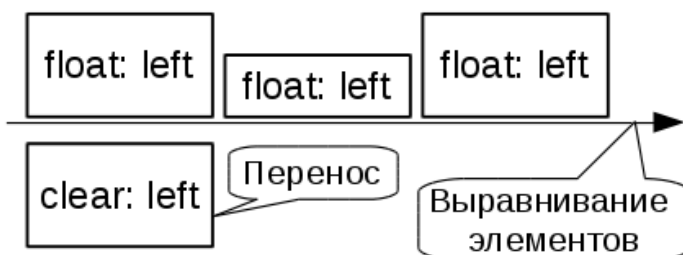
Для задания размеров блочных элементов используются атрибуты *width* и *height*, задающих соответственно ширину и высоту элемента. По умолчанию ширина и высота элемента определяются его содержимым (значение *auto*). Можно задать относительное значение размера в процентах от родительского элемента. Если размер родительского элемента не задан, то он вычисляется от размера окна браузера. Например:

```
.header { height: 10% }
```

При задании относительных размеров элементов может возникнуть необходимость указать для них минимальные или максимальные возможные размеры. Для указания минимальной ширины и высоты используются атрибуты *min-width* и *min-height*. Для указания максимальной ширины и высоты используются атрибуты *max-width* и *max-height*. Значения данных атрибутов также могут быть как абсолютными, так и относительными.

Параметры размещения

Местоположение блочных элементов относительно друг друга определяется атрибутом *float*. Он задает способ выравнивания текущего блока относительно родительского элемента. Возможные значения: *none* — блочные элементы следуют друг за другом сверху вниз, без обтекания (по умолчанию), *left* — блок выравнивается по левому краю родительского элемента, остальное содержимое обтекает его справа, *right* — блок выравнивается по правому краю родительского элемента, содержимое обтекает его слева. Данный атрибут применяется к нескольким элементам для организации плавающей верстки, когда элементы выстраиваются по горизонтали.



Если необходимо явно указать, что текущий элемент будет располагаться ниже других, имеющих установленный атрибут *float*, к такому элементу применяется атрибут *clear*. Возможные значения: *none* — выравнивание не меняется (по умолчанию); *left* — текущий элемент располагается ниже всех элементов, имеющих значение *float: left*; *right* — текущий элемент располагается ниже всех элементов, имеющих значение *float: right*; *both* — текущий элемент располагается ниже всех элементов, имеющих установленное значение *left* или *right* атрибута *float*. Пример стиля для тега `<footer>`:

```
footer { clear: both }
```

Параметры переполнения

В случае, когда содержимое не помещается в установленный размер элемента, возникает переполнение. Поведение отображения содержимого при переполнении определяется атрибутом *overflow*. Возможные значения: *visible* — автоматически увеличится высота элемента (по умолчанию), *hidden* — не

помещающееся содержимое элемента будет обрезано. Высота элемента останется прежней, *scroll* — элемент будет иметь полосы прокрутки для отображения непомещающегося содержимого, *auto* — если возникнет переполнение содержимого, то появятся полосы прокрутки.

Необходимо отметить, что атрибут *overflow* устанавливается, когда высота элемента *height* указана в абсолютных единицах. При использовании относительных размеров высоты переполнение не работает.

Атрибуты *overflow-x* и *overflow-y* задают поведение отображения при переполнении содержимого элемента по горизонтали или вертикали, соответственно. Значения этих атрибутов аналогичны *overflow*.

Параметры отступов

В стандарте CSS используются два вида отступов:

- внутренний отступ, между содержимым и внешней границей элемента;
- внешний отступ, между элементом и соседними блочными элементами.

Величина внутренних отступов определяется свойством *padding*:

padding: <отступ1> [<отступ2>] [<отступ3>] [<отступ4>]

Если задан один параметр <отступ1>, то он применяется ко всем сторонам элемента; если два параметра, то первый указывает верх-низ, второй — правую-левую стороны; если три — первый указывает верх, второй — правую-левую стороны, третий - низ; если четыре параметра — первый указывает верх, второй — правую сторону, третий — низ, четвертый — левую сторону. Пример:

.indented { padding: 0cm 2cm 2cm 2cm }

Атрибуты внутренних отступов могут быть указаны отдельно: *padding-left*, *padding-top*, *padding-right*, *padding-bottom*.

Пример:

.indented { padding-left: 2cm }

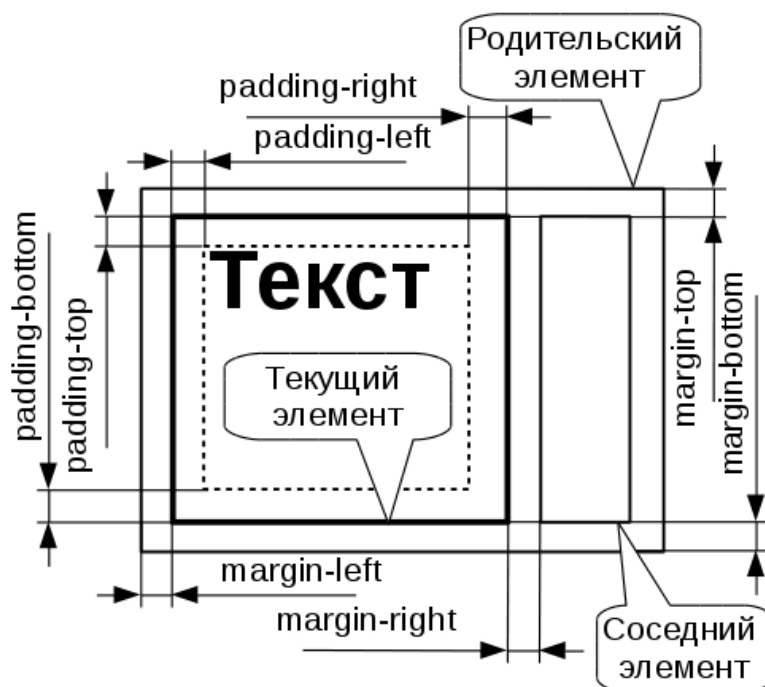
Величина внешних отступов определяется свойством *margin*:

margin:
[<отступ2>] [<отступ3>]
[<отступ4>]

Параметры внешних отступов атрибута *margin* аналогичны параметрам внутренних отступов *padding*. Пример задания отступов сверху и снизу:

h1 { margin: 5mm 0mm }

Атрибуты внешних отступов могут быть указаны отдельно:



margin-left, margin-top, margin-right, margin-bottom. Например:

```
h1 { margin-top: 5mm }
```

Необходимо отметить, что для таблицы внешние отступы ячеек задаются отдельным атрибутом *border-spacing*, вида:

```
border-spacing: <отступ1> [<отступ2>]
```

Если задан один параметр *<отступ1>*, то он применяется ко всем сторонам ячейки таблицы; если два параметра, то первый указывает отступ слева-справа, второй — верх-низ.

Параметры рамок

Для задания рамки вокруг блочного элемента используется атрибут *border*:

border: <толщина> <стиль> <цвет>, например:

```
td { border: 1px solid black }
```

Толщина рамки <толщина> задается атрибутом *border-width*, имеющим вид:

```
border-width: <толщина1> [<толщина2>] [<толщина3>] [<толщина4>]
```

Параметры толщины атрибута *border-width* применяются к границам рамки в последовательности, аналогичной параметрам отступов. Значения толщины могут задаваться одной из констант: *thin* — тонкая, *medium* — средняя, *thick* — толстая или одним из размерных значений CSS. Пример задания тонкой границы рамки сверху и снизу:

```
h1 { border-width: thin 0 }
```

Атрибуты толщины границы также могут быть указаны отдельно: *border-left-width* — для левой границы, *border-top-width* — для верхней границы, *border-right-width* — для правой границы, *border-bottom-width* — для нижней границы. Например, для нижней границы рамки:

```
h1 { border-bottom-width: 2px } /* толщина 2px */
```

Стиль рамки <стиль> задается атрибутом *border-style*, имеющим вид:

```
border-style: <стиль1> [<стиль2>] [<стиль3>] [<стиль4>]
```

Параметры стиля атрибута *border-style* применяются к границам рамки в последовательности, аналогичной параметрам отступов. Значения стиля могут задаваться одной из констант: *none* или *hidden* — рамка отсутствует (по умолчанию), *solid* — сплошная линия, *dotted* — пунктирная линия, *dashed* — штриховая линия, *double* — двойная линия, *groove* и *ridge* — вдавленная или выпуклая 3D линия, *inset* и *outset* — вдавленная или выпуклая 3D плоскость. Пример задания выпуклой 3D границы рамки вокруг всего элемента:

```
h1 { border-style: ridge }
```

Атрибуты стиля границы также могут быть указаны отдельно: *border-left-style* — для левой границы, *border-top-style* — для верхней границы, *border-right-style* — для правой границы, *border-bottom-style* — для нижней границы.

Например, для нижней границы рамки:

```
h1 { border-bottom-style: solid } /* сплошная линия */
```

Цвет рамки *<цвет>* задается атрибутом *border-color*, имеющим вид:

```
border-color: <цвет1> [<цвет2>] [<цвет3>] [<цвет4>]
```

Параметры цвета атрибута *border-color* применяются к границам рамки в последовательности, аналогичной параметрам отступов. Пример:

```
h1 { border-color: lightgreen } /* светло-зеленый цвет */
```

Атрибуты цвета границы также могут быть указаны отдельно: *border-left-color* — для левой границы, *border-top-color* — для верхней границы, *border-right-color* — для правой границы, *border-bottom-color* — для нижней границы. Например, для нижней границы рамки:

```
h1 { border-bottom-color: #0F0 } /* светло-зеленый цвет */
```

Радиус закругления углов рамки указывается с помощью атрибута *border-radius*, имеющим вид:

```
border-radius: <r1> [<r2>] [<r3>] [<r4>]
```

Параметры радиусов атрибута *border-radius* применяются к границам рамки в последовательности, аналогичной параметрам отступов, начиная с верхнего левого угла. Каждый параметр может быть как числом, указывающим радиус круга, так и числовой дробью, указывающей размеры полуосей эллипса. Размеры могут быть заданы как в размерных единицах CSS, так и в процентах от ширины блока. Пример:

```
h1 { border-radius: 2px/3px } /* радиус углов рамки 2 на 3 пиксела */
```

Параметры выделения

Для задания выделения блочного элемента используется атрибут *outline*. Выделение представляет собой границу, окружающую данный элемент. В отличие от рамки, выделение не увеличивает размер элемента. Параметры выделения такие же, как и параметры рамок.

outline: <толщина> <стиль> <цвет>, например:

```
p { outline: thin dotted black }
```

Для задания параметров выделения также можно использовать отдельные свойства: *outline-width*, *outline-color*, *outline-style*. Параметры свойств выделения аналогичны параметрам свойств рамок.

Параметры таблиц

Для выравнивания содержимого ячеек таблиц по-горизонтали и по-вертикали используются ранее рассмотренные атрибуты *text-align* и *vertical-align*. В отличие от блоков, при установке атрибута *vertical-align* для ячеек таблицы обычно используются три типа выравнивания содержимого: *top* — по верхней границе, *middle* — по центру и *bottom* — по нижней границе.

Для установки внутренних отступов для содержимого ячеек используется атрибут *padding* или его производные: *padding-left*, *padding-top*, *padding-right* и *padding-bottom*.

Для задания внешних отступов между границами ячеек используется атрибут *border-spacing*. Атрибут *margin* задает внешние отступы для таблицы в целом. Аналогично, атрибут *align* задает выравнивание таблицы в целом, как блочного элемента, относительно родительского элемента (или страницы). Например, выравнивание таблицы по центру страницы задается стилем:

```
table { align: center }
```

Для задания границ ячеек используются атрибут *border* или его производные. Необходимо отметить, что атрибут *border* может применяться отдельно как к таблице в целом, так и для отдельных ячеек. В этом случае, каждая ячейка заключается в отдельную рамку.

Для объединения рамок ячеек в таблице используется специальный атрибут *border-collapse*. Допустимые значения: *separate* — отдельная рамка (по умолчанию), *collapse* — объединение рамок ячеек в одну сетку.

Для задания размера ячеек используются стандартные атрибуты ширины и высоты: *width* и *height*. Эти же атрибуты применимы и к таблице в целом.

По умолчанию размеры ячеек и таблицы в целом определяются их содержимым. Для изменения этого поведения применительно к таблице используется атрибут *table-layout*. Возможные значения: *auto* — размер таблицы и ячеек определяются содержимым (по умолчанию), *fixed* — размеры таблицы и ячеек будут установлены фиксированными.

Если при задании фиксированных размеров элемента, его содержимое не будет уместаться, то возникнет переполнение. Поведение отображения содержимого при переполнении ячеек таблицы задается при помощи вышеуказанных атрибутов *overflow*, *overflow-x* и *overflow-y*.

Атрибут *caption-side* устанавливает местоположение заголовка таблицы относительно его содержания. Возможные значения: *top* — заголовок сверху (по умолчанию), *bottom* — заголовок снизу. Атрибут применяется к тегу `<table>`.

Атрибут *empty-cells* указывает поведение при отображении пустых ячеек таблицы. Возможные значения: *show* — пустые ячейки выводятся на экран вместе с их оформлением, *hide* — пустые ячейки не будут выводиться на экран. Атрибут может применяться также к таблице целиком.

2.4 Видимость и эффекты

Параметры курсора

Форма и вид курсора, отображаемого при наведении на определенный элемент страницы, устанавливается с помощью атрибута *cursor*. Наиболее часто используемые значения: *auto* — устанавливаемое автоматически браузером (по

умолчанию), *default* — указатель стрелка, *none* — курсор не отображается, *help* — стрелка с вопросительным знаком, *pointer* — указующий перст (используется при наведении на гиперссылку), *progress* — стрелка с песочными часами (используется при ожидании), *wait* — песочные часы (используется при ожидании), *text* — текстовый курсор (используется при вводе текста).

Параметры и вид отображения

Вид отображения элемента устанавливаются с помощью атрибута *display*. Это многоцелевое свойство, которое определяет, как элемент должен быть показан в документе. Оно устанавливает, будет ли элемент блочным, строчным, строчно-блочным и т.д. Возможные значения: *inline* — строчный элемент (по умолчанию для строчных элементов); *block* — блочный элемент (по умолчанию для блочных элементов); *none* — не отображается, место под него не резервируется; *inline-block* — строчно-блочный элемент, комбинирует внутреннее поведение блочного и внешнее поведение строчного элемента; *list-item* — элемент списка; *table* — таблица; *inline-table* — таблица с внешним поведением строчного элемента; *table-caption* — заголовок таблицы; *table-column* — столбец таблицы; *table-row* — строка таблицы; *table-cell* — ячейка таблицы; *table-header-group* — секция заголовка таблицы; *table-row-group* — секция содержания таблицы; *table-footer-group* — секция подвала таблицы; *table-column-group* — группа столбцов таблицы. Наиболее часто используется свойство *none*, позволяющее делать элемент невидимым:

```
.hidden { display: none }
```

Значения атрибута *display*, начинающиеся с *table-* позволяют использовать теги `<div>` в качестве элементов таблицы.

Параметры отображения элемента также могут устанавливаться с помощью атрибута *visibility*. Применяется ко всем элементам страницы. Возможные значения: *visible* — видимый (по умолчанию), *hidden* — скрытый, *collapse* — скрывает заданные строки и столбцы таблицы.

Позиционирование

Позиция блочного элемента на странице обеспечивается его поведением, заданным с помощью атрибута *position*. Возможные значения: *static* — блок непозиционируемый (по умолчанию); *absolute* — абсолютно позиционируемый, место на странице не выделяется. Если родительский элемент позиционирован, то текущий элемент позиционируется от него, иначе — от края экрана; *fixed* — подобный абсолютному позиционированию, но не меняет позицию элемента при прокрутке содержимого родительского элемента; *relative* — относительно позиционируемый, положение элемента устанавливается относительно его исходного места, место под элемент выделяется.

Для установки позиции элемента используются атрибуты *top*, *left*, *right* и *bottom*, устанавливающие расстояние от верхней, левой, правой или нижней

границы текущего элемента до соответствующей границы родительского элемента или экрана. Значение *auto* вышеуказанных атрибутов не изменяет положение текущего элемента. Пример — левый верхний угол элемента с `id="search"` на экране имеет координаты (200, 100):

```
#search { position: absolute; left: 200px; top: 100px }
```

Перекрытие и область видимости

По умолчанию позиционированный элемент, определенный позже, перекрывает элементы, определенные раньше него. Это поведение можно изменить с помощью атрибута *z-index*. Значение атрибута устанавливается целым числом, определяющим порядок перекрытия. Элементы с большим значением *z-index* перекрывают элементы с меньшим значением. Значение *auto* определяет порядок перекрытия по умолчанию. Пример:

```
#search { z-index: 2 }
```

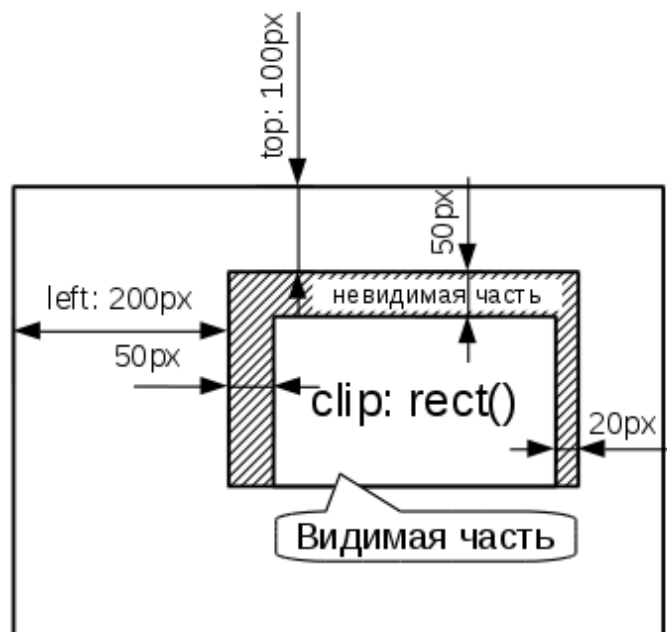
Область видимости позиционированного элемента определяется атрибутом *clip*, задающим прямоугольник маски и имеющим вид:

```
clip: rect(<верх>, <право>, <низ>, <лево>)
```

Параметры *<верх>*, *<право>*, *<низ>* и *<лево>* определяют расстояние от верхней, правой, нижней и левой границы текущего элемента до области маски.

Прямоугольник маски задает видимую область элемента. Значение *auto* атрибута *clip* убирает маску и делает видимым весь элемент. Пример:

```
#search { position: absolute;
  left: 200px; top: 100px
  clip: rect(50px, 20px, 0px, 50px)
}
```



Градиенты

Функция *linear-gradient()* добавляет линейный градиент к фону элемента. Она выступает значением свойства *background-image* или *background*. Вид:

```
background-image: linear-gradient(<направл>, <цветн>, <цветк>);
```

Параметры *<цветн>* и *<цветк>* задают начальный и конечный цвет градиента с необязательной позицией цвета относительно оси градиента в процентах.

Параметр *<направл>* задает направление градиента. Направление может задаваться с помощью угла наклона *deg*, отсчитываемого от вертикали по часовой стрелке, например *45deg* или с помощью позиции, определяемой константой *to* и комбинациями *left*, *right*, *top* и *bottom*, например: *to top* (или

0deg) — снизу вверх; *to left* (или *270deg*) — справа налево; *to top right* (или *45deg*) — от левого нижнего угла к правому верхнему. Пример:

background-image: linear-gradient(to top, #fefcea, #f1da36);

Трансформации

Трансформации позволяют выполнять над элементом различные геометрические преобразования: масштабирование, переносы, повороты и др. Для этого используется атрибут *transform* (для браузера Google Chrome и Android используется атрибут *-webkit-transform*, для браузера Internet Explorer – атрибут *-ms-transform*). Вид атрибута *transform*:

transform: <функция>

Параметр *<функция>* определяет функцию трансформации. Для отмены трансформации используется значение *none*. Виды функций трансформации:

rotate(<угол>) — поворот элемента на заданный угол по часовой стрелке, относительно точки привязки, заданной дополнительным атрибутом *transform-origin*. Обычно значения атрибута определяются координатами, задаваемыми в виде двух чисел или констант: *left*, *center*, *right* – по горизонтали и *top*, *center*, *bottom* – по вертикали. Например:

```
.box { transform-origin: left, top; /* левый верхний угол элемента */  
      transform: rotate(45deg); } /* поворот на 45 градусов */
```

scale(sx [,sy]) — масштабирование элемента по горизонтали и вертикали. Значения *sx* и *sy* задаются в относительных единицах — текущий размер элемента принимается за 1. Функции *scaleX(sx)* и *scaleY(sy)* задают отдельное масштабирование элемента по горизонтали или вертикали.

skewX(<угол>) и *skewY(<угол>)* — наклон элемента на заданный угол

translate(dx [,dy]) — перенос элемента по горизонтали и вертикали на заданное значение *dx* и *dy*. Функции *translateX(dx)* и *translateY(dy)* задают отдельный перенос элемента по горизонтали или вертикали. Значения *dx* и *dy* могут быть отрицательными:

```
.box { transform: translate(-2px) } /* смещение влево на 2px */
```

Анимация по кадрам

К элементам страниц можно применить вид анимации по ключевым кадрам. Для этого используется атрибут *animation* (для браузера Google Chrome используется атрибут *-webkit-animation*), который работает в паре с правилом *@keyframes* (для браузера Google Chrome используется правило *@-webkit-keyframes*). Общий вид атрибута *animation*:

animation: <переменная> [<время>] [<функция>] [<задержка>] [<итер>] [<направление>] [<режим>] [<состояние>]

Кроме параметра *<переменная>*, каждый из этих параметров может быть в любой комбинации установлен с помощью отдельных атрибутов: *animation-*

duration, *animation-timing-function*, *animation-delay*, *animation-iteration-count*, *animation-direction*, *animation-fill-mode* и *animation-play-state*.

Параметр <переменная> связывает анимацию с правилом @keyframes:

```
.fadeIn { animation: fadeIn 3s; }
```

```
@keyframes fadeIn { from { opacity: 0; } to { opacity: 1; } }
```

Таким образом элемент со стилем *.fadeIn* будет изменять свою прозрачность в течение 3 секунд.

Параметр <время> задается атрибутом *animation-duration* и определяет время действия цикла анимации в секундах *S* или миллисекундах *ms*, например *animation-duration: 3s* — три секунды.

Параметр <функция> задается атрибутом *animation-timing-function* и определяет временную функцию анимирования. Возможные значения: *ease* — ускорение в середине (по умолчанию); *ease-in* — постепенное ускорение; *ease-out* — постепенное замедление; *ease-in-out* — медленное начало и конец; *linear* — линейная скорость; *step-start* и *step-end* — начальное или конечное состояние без анимации; *steps(n, start* или *end)* — ступенчатая функция с заданным числом шагов; *cubic-bezier(a,b,c,d)* — кривая Безье.

Параметр <задержка> задается атрибутом *animation-delay* и определяет время ожидания перед выполнением анимации (по умолчанию 0). Время задается в секундах *S* или миллисекундах *ms*.

Параметр <умер> задается атрибутом *animation-iteration-count* и определяет число итераций (1 по умолчанию). Бесконечное число итераций задается константой *infinite*.

Параметр <направление> задается атрибутом *animation-direction* и определяет направление движения анимации. Возможные значения: *normal* — с начала до конца и возврат в исходное состояние (по умолчанию); *alternate* — с начала до конца и плавный возврат в исходное состояние; *reverse* — с конца до начала; *alternate-reverse* - с конца до начала и плавный возврат в исходное состояние.

Параметр <режим> задается атрибутом *animation-fill-mode* и определяет состояние элемента после окончания анимации. Возможные значения: *none* — стили не применяются, по умолчанию элемент будет находиться в исходном состоянии; *forwards* — к элементу применяется стиль последнего ключевого кадра; *backwards* — к элементу применяется стиль первого ключевого кадра.

Параметр <состояние> задается атрибутом *animation-play-state* и определяет состояние анимации. Возможные значения: *running* — проигрывать анимацию (по умолчанию); *paused* — приостановить анимацию.

Анимировать можно различные свойства стилей: прозрачность, размер, цвет, положение элементов. Это придает элементам страницы динамичность и современный вид.

Анимация переходов

При наведении курсора мыши можно использовать анимацию переходов. Обычно при использовании псевдокласса *:hover* изменение происходит мгновенно, атрибут *transition* позволяет задать продолжительность и способ перехода (для Android используется атрибут *-webkit-transition*). Общий вид:

transition: <свойство> [<время>] [<функция>] [<задержка>]

Кроме того, каждый из параметров может быть в любой комбинации установлен с помощью отдельных атрибутов: *transition-property*, *transition-duration*, *transition-timing-function*, *transition-delay*.

Параметр *<свойство>* задается атрибутом *transition-property* и определяет свойство элемента, подлежащего анимации, например: *top*. Значение *none* не задает ни одно свойство, значение *all* относится ко всем свойствам. В случае анимации нескольких свойств, они перечисляются через запятую.

Параметр *<время>* задается атрибутом *transition-duration* и определяет время действия перехода, аналогично атрибуту *animation-duration*.

Параметр *<функция>* задается атрибутом *transition-timing-function* и определяет временную функцию перехода, аналогично атрибуту *animation-timing-function*.

Параметр *<задержка>* задается атрибутом *transition-delay* и определяет время ожидания перед выполнением перехода, аналогично атрибуту *animation-delay*. Пример анимации перехода для стиля меню:

```
.menu {  
    position: absolute;  
    top: -100px;  
    transition: top 2s;  
}  
.menu:hover { top: 0; }
```

3 Событийное программирование JavaScript

3.1 Синтаксис языка JavaScript

Язык Javascript является объектно-ориентированным интерпретатором, встроенным в браузер, и служит для обработки программных сценариев страницы. Сценарии (или скрипты) обеспечивают динамическую обработку событий и взаимодействие пользователя с интерфейсом страницы. Спецификация языка описывается стандартом ECMAScript и постоянно развивается консорциумом W3C.

В отличие от языка Java, язык Javascript не компилирует скрипты, а выполняет их непосредственно, что накладывает ряд особенностей, например, возможно использование переменных в скриптах без их предварительного объявления и

т.д. Также для обеспечения безопасности клиентских приложений, язык Javascript не может совершать прямые файловые операции ввода-вывода, не имеет прямого доступа к функциям операционной системы. Возможность работы с файлами у скриптов ограничена пределом "песочницы" браузера (отведенного каталога). С помощью технологии Javascript можно выполнять запросы к серверу (в том числе, без перезагрузки страницы, используя технологию AJAX), иметь программный доступ ко всем элементам страницы, включая CSS стили (с помощью объектной модели документа - DOM), управлять поведением браузера (с помощью объектной модели браузера - BOM) и т. д.

На стороне клиента сценарии Javascript обеспечивают эффективное взаимодействие пользователя с интерфейсом страницы и страницы — с сервером. Программы на Javascript могут выполнять первичную фильтрацию передаваемых на сервер, принимать от сервера данные в форматах XML и JSON, осуществлять предобработку форм, генерацию элементов пользовательского интерфейса и другие операции.

В основе синтаксиса языка Javascript лежит интерпретируемый код, заключенный в функции — обработчики. Код скриптов обычно загружается вместе с загрузкой страницы и размещается в специальном контейнерном теге:

```
<script>
  alert("Hello, world!");
</script>
```

Контейнер `<script>` чаще всего размещается в заголовке страницы `<head>` и сразу же интерпретируется. Иногда скрипты размещают в других местах страницы, например, в подвале, чтобы их интерпретация и выполнение не задерживало загрузку страницы. В настоящее время для асинхронной загрузки скриптов используется атрибут без содержания *async*.

Часто скрипты размещают в отдельных текстовых файлах с расширением *.js* которые подключают к контейнеру `<script>` с помощью атрибута *src*:

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

В данном примере скрипт содержится в файле *form.js*, размещенным в локальном каталоге *js* относительно текущего корня веб-сервера. В качестве значения также может быть указан абсолютный URL адрес скрипта:

```
<script async src="http://www.google-analytics.com/analytics.js"></script>
```

Синтаксис сценариев на языке Javascript включает набор команд, разделенных символом `;` (точка с запятой):

```
alert('Hello'); alert('world!!!'); // два текстовых сообщения
```

Переменные

Для объявления переменных в Javascript используется ключевое слово *var*:

```
var <идентификатор> [=<значение>];
```

Идентификатор представляет собой имя переменной, состоящее из латинских

букв, цифр, символов: \$ (денежный знак) и _ (подчеркивание), начинающееся не с цифры. В качестве идентификаторов нельзя использовать зарезервированные команды языка. При объявлении нескольких переменных, идентификаторы разделяются запятыми. Пример:

```
var author = 'Ivan', age = 25, message_12 = 'Hello!';
```

При описании переменных регистр букв имеет значение. Традиционно в Javascript заглавными буквами объявляются константы:

```
var GREEN = '#00FF00';
```

Если при инициализации переменной ключевое слово *var* не указано, то такая переменная является *глобальной*.

В языке Javascript определено шесть типов данных:

1) *Число* (number) представляет собой целое или действительное число в обычной или экспоненциальной форме записи. Кроме того, существует два специальных значения — *NaN* (не число) и *Infinity* (бесконечность). На представление числа в Javascript отводится 32 бита (4 байта). Примеры:

23, 3.14, -8, 6e-3, 0.006, NaN.

Кроме десятичных чисел, в программах на Javascript можно использовать шестнадцатичные, восьмеричные и двоичные числа. Для этого их предваряют префиксами: *0x* (шестнадцатичное), *0* (восьмеричное), *0b* (двоичное). Примеры чисел: *0xffff*, *0777*, *0b1111*.

2) *Строка* (string) представляет собой любую последовательность символов, заключенных в кавычки или апострофы. Символьного типа данных в языке нет. Строка может содержать специальные символы типа "\n" (перевод строки) или быть пустой "". В Javascript используется внутреннее представление строк в формате Unicode. Примеры строк: *"программа"*, *'Hello'*, *'A'*.

3) *Булевый тип* (boolean) представляет логический тип данных. Переменные этого типа принимают одно из двух значений: *true* (истина) или *false* (ложь).

4) *Пустой тип* (null) – отдельный тип данных, означает: значение неизвестно. Пример: *age = null*;

5) *Неопределенный тип* (undefined) – отдельный тип данных, означает: значение не присвоено. Пример: *var age*;

6) *Объект* (object) — представляет собой объект Javascript. Объект объявляется с помощью фигурных скобок и может состоять из полей и методов. Пример:

```
var user = { name: "Василий" };
```

Комментарии

Комментарии в языке Javascript указываются в стиле C++.

Однострочный комментарий указывается с помощью символов // (двойной слеш) и действует до конца строки. Пример:

var age; // undefined – значение не присвоено

Многострочный комментарий указывается внутри последовательности символов */** и **/* и действует на все многострочное содержимое:

/ ...комментарий... */*

Операторы

Язык Javascript имеет C-подобные операторы и команды.

Оператор присвоения служит для присвоения значений переменным:

c = a + b; // присвоение суммы a и b

Возможно последовательное применение нескольких операторов присвоения. Оператор присвоения правоассоциативен (выполняется в последовательности справа налево). Пример:

c = d = x; // сначала d = x, потом c = d

*Арифметические операторы: + (сложение), - (вычитание), * (умножение), / (деление), % (остаток от деления)* подобные используемым в других языках. Все арифметические операторы являются бинарными и левоассоциативными. Оператор *+* (сложение) в зависимости от типов аргументов может использоваться как для сложения чисел, так и для объединения (конкатенации) строк.

Операторы сравнения: < (меньше), > (больше), == (равно), != (не равно), <= (меньше или равно), >= (больше или равно), === (строго равно), !== (строго не равно). Последние два оператора отличаются от обычного равенства и неравенства тем, что при их выполнении над аргументами не производится преобразование типов. Пример:

c = (2 == '2'); // результат c = true

c = (2 === '2'); // результат c = false

Операторы сравнения имеют логический тип возвращаемого значения и часто используются в условных конструкциях языка.

Логические операторы: && (логическое И, конъюнкция), || (логическое ИЛИ, дизъюнкция), ! (логическое НЕ, отрицание). Логические операторы применяются к логическим выражениям и имеют логический тип возвращаемого значения: *true* или *false*. Пример:

*c = !(2*2 == 4); // результат c = false*

c = (5 < 2) || (3 > 2); // результат c = true

Унарные операторы представляют собой операторы с одним аргументом: *+* (унарный плюс), *-* (унарный минус), *++* (инкремент), *--* (декремент), *!* (логическое отрицание), *~* (битовое НЕ).

Операторы инкремент (увеличение на 1) и декремент (уменьшение на 1) могут быть *инфиксные* (ставятся перед аргументом) или *постфиксные* (ставятся после аргумента). Это влияет на порядок вычислений. Остальные унарные операторы

являются инфиксными. Пример:

```
c = 2; a=c++; // a = 2, c = 3
```

```
c = 2; a=++c; // a = 3, c = 3
```

Побитовые операторы: & (побитовое И), | (побитовое ИЛИ), ^ (побитовое исключающее ИЛИ), ~ (побитовое НЕ), >> (сдвиг вправо), << (сдвиг влево), >>> (сдвиг вправо с заполнением 0).

Побитовые операторы, в отличие от логических, применяются к операндам побитно. Они имеют числовой тип возвращаемого значения. Операторы сдвига являются аналогами умножения (деления) на число 2 в степени равной количеству сдвигаемых бит. Оператор сдвига вправо с заполнением 0, в отличие от обычного сдвига при выполнении добавляет нули слева. Пример:

```
c = 5 & 6; // c = 4 (0b101 & 0b110 = 0b100)
```

```
c = -9 >> 2; // c = -3 (0xffffffd)
```

```
c = -9 >>> 2; // c = 1073741821 (0x3fffffd)
```

Побитовые операции выполняются после операций сравнения.

Операторы с присвоением: += (сложение), -= (вычитание), *= (умножение), /= (деление) и т.д. являются сокращенной формой записи двух операторов: присвоения и действия. Пример:

```
c += 2; // то же, что c = c + 2
```

Приведение простых типов

В языке Javascript может использоваться явное и неявное преобразование простых типов. Преобразования типов используются для строк, чисел и логических переменных.

Функция `typeof(x)` позволяет определить тип переменной. Результатом ее выполнения является строка, содержащая тип. Пример:

```
typeof(3); // "number"
```

В случае сложения строки с числом, если строка может быть преобразована в число, выполняется преобразование и сложение ее как числа, в противном случае, выполняется операция объединения (конкатенация). Пример:

```
c = 3 + '2'; // результат c = 5
```

```
c = 'help' + 4; // результат c = 'help4'
```

Для явного приведения к строке также может использоваться конструктор `String(x)`, например: `String(null);` // строка "null".

Оператор + (унарный плюс) используется для приведения аргумента к числу:

```
c = +'3' + 2; // результат c = 5
```

Для преобразования строк в числа можно использовать специальные функции мягкого преобразования. Основная их особенность заключается в том, что строка в этом случае преобразуется посимвольно в число до тех пор, пока это возможно. Таким образом достигается положительный результат.

Функция `parseInt(<строка>[,<основание>])` преобразует строку к целому числу по заданному основанию (по умолчанию, к десятичному). Например:

```
c = parseInt("3px"); // результат c = 3
d = parseInt("ff",16); // результат d = 255
```

Функция `parseFloat(<строка>)` преобразует строку к действительному числу. Например:

```
c = parseFloat("3.14"); // результат c = 3.14
```

Для явного приведения к числу также можно использовать конструктор `Number(x)`, например: `Number("3");` // число 3.

Для того, чтобы узнать, является ли значение данного выражения числом, используется функция `isNaN(x)`, возвращающая логическое значение:

```
isNaN("25 руб."); // true (не число).
```

При выполнении приведения к числу значения `null` и `false` преобразуются в число 0, значение `true` — в 1, значение `undefined` — в `NaN`.

Оператор `!!` (двойное отрицание) используется для приведения аргумента к логическому значению, например: `alert(!!"0");` // true

При использовании логического типа строка `""`, а также числа 0, `NaN`, `undefined` и `null` интерпретируются как `false`, остальные — как `true` (в том числе и строка `"0"`).

Для явного приведения к логическому типу также можно использовать конструктор `Boolean(x)`, например: `Boolean("1");` // true.

Приоритет операций

В Javascript существует 19 приоритетов операций.

Некоторые из них (в порядке убывания приоритета):

- 1) группировка операторов, заключенных в скобки `()`;
- 2) операторы доступа к элементам: `.` (точка), `[]`;
- 3) вызов функции `()`, команда `new`;
- 4) унарные операторы: `+`, `-`, `++`, `--`, логическое и побитовое отрицание: `!`, `~`;
- 5) умножение, деление и остаток: `*`, `/` и `%`;
- 6) сложение и вычитание: `+` и `-`;
- 7) побитовые сдвиги: `>>`, `<<`, `>>>`;
- 8) операторы сравнения: `<`, `>`, `<=`, `>=`, `==`, `!=` и т.д.;
- 9) побитовое И `&`;
- 10) побитовое ИЛИ `|`;
- 11) логическое И `&&`;
- 12) логическое ИЛИ `||`;
- 13) присвоение: `=`, `+=`, `-=` и т.д.

Сначала выполняются операции с высшим приоритетом. Пример:

```
c = 2 + 4*6 / 3*2 | 1 << 2; // 2 + (24/3*2) | 4 = (2 + 16) | 4 = 22
```

Ввод — вывод значений

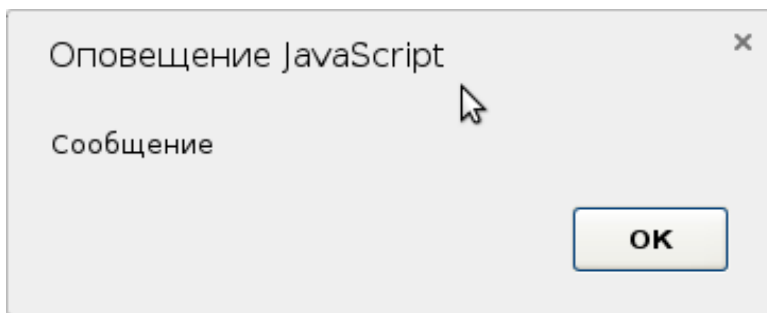
Пользователь может взаимодействовать с Javascript программой с помощью модальных окон. Модальные окна прерывают работу программы до тех пор, пока пользователь не введет данные или не нажмет соответствующую кнопку. В Javascript используется три вида функций для вывода модальных окон:

1) Функция *alert* выводит информационное сообщение. Формат:

```
alert(<сообщение>);
```

Пример:

```
alert("Сообщение"); // вывод информационного сообщения
```



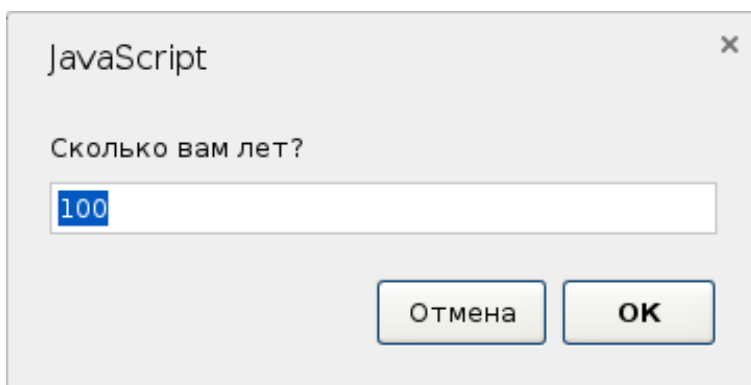
Текстовое окно используется для оповещения пользователя и для отладки. При нажатии кнопки *OK* или клавиши *<Esc>* окно исчезает.

2) Функция *prompt* выводит окно с сообщением и полем ввода. Формат:

```
<переменная> = prompt(<сообщение>, <значение_по_умолч>);
```

Пример:

```
var year = prompt("Сколько вам лет?", 100);
```



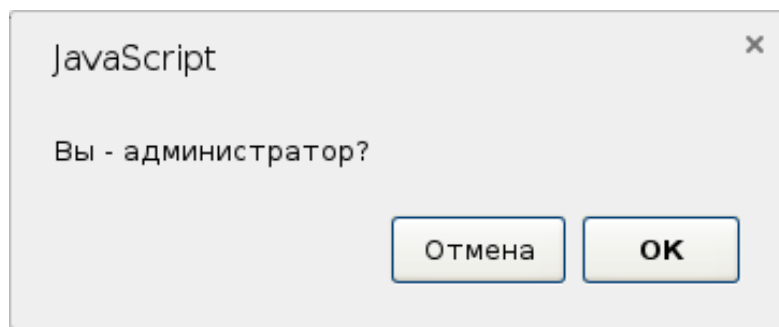
Введенное в поле ввода значение присваивается указанной переменной. Вторым параметром задается значение поля по умолчанию. Если он не используется, рекомендуется задать пустую строку *""*. После нажатия клавиши *OK* функция возвращает введенное текстовое значение. При нажатии кнопки *<Отмена>* или клавиши *<Esc>* возвращается текстовое значение *"null"*.

3) Функция *confirm* выводит запрос подтверждения с двумя кнопками. Формат:

```
<переменная> = confirm(<запрос>);
```

Пример:

```
var admin = confirm("Вы - администратор?");
```



При нажатии на кнопку *ОК* функция возвращает значение *true*, в ином случае — *false*.

Управляющие конструкции

Язык Javascript имеет С-подобные управляющие конструкции.

Простой условный оператор (если-то) имеет вид:

```
if (<выражение>) <действие>;
```

Если выражение в круглых скобках истинно, то выполняется действие. Пример:

```
if (a < 0) alert("Число a - отрицательное!");
```

При необходимости выполнения нескольких действий по условию, используются операторные скобки `{ }`. После операторных скобок точка с запятой не ставится. Пример:

```
if (a < 0) { a = -a; alert("Модуль числа " + a); }
```

Другой тип условного оператора (если-то-иначе) имеет вид:

```
if (<выражение>) <действие1> else <действие2>;
```

Если выражение истинно, то выполняется *действие1*, иначе — выполняется *действие2*. Пример:

```
if (x < 0) alert('x - отрицательное'); else alert('x - положительное');
```

Обратите внимание, что перед оператором *else* должна стоять точка с запятой. При необходимости в операторе можно использовать операторные скобки, а также использовать вложенные и составные конструкции:

```
if (d > 0) {  
    x1 = (-b+Math.sqrt(d))/(2*a);  
    x2 = (-b-Math.sqrt(d))/(2*a);  
    alert("корни x1="+x1+" x2="+x2);  
} else if (d = 0) {  
    x1 = -b/(2*a);  
    alert("корень x1="+x1);  
} else alert("корней нет");
```

Функция извлечения квадратного корня относится к глобальному объекту *Math*.

Оператор *присвоения с условием* имеет вид:

<переменная> = (<условие>) ? <выражение1>: <выражение 2>

f = (x > 0)? x: -x; // функция модуля f=|x|

Этот оператор является сокращенным вариантом конструкции:

if (x>0) f = x; else f = -x;

Оператор выбора применяется при проверке нескольких условий:

```
switch (<переменная>) {  
    case <значение1>: <действие1>;  
    ...  
    case <значениеN>: <действиеN>;  
    [default: <оператор по умолчанию>;]  
}
```

Если переменная принимает одно из заданных значений, то выполняется соответствующее действие. Пример:

```
switch (a) {  
    case 0: alert('ложь'); break;  
    case 1: alert('истина'); break;  
    default: alert('значение не установлено');  
}
```

Команда *break* прекращает вычисления после вывода сообщения и производит выход из оператора *switch*. Если она не указана, то происходит переход к следующей строке оператора *switch*. Условие *default* срабатывает при выборе любого другого значения переменной *a*, кроме 0 и 1.

Оператор параметрического цикла имеет вид:

for ([<начало>]; [<условие>]; [<шаг>]) <тело цикла>;

Параметр *<начало>* задает начальное условие, параметр *<условие>* — условие выполнения цикла, параметр *<шаг>* — изменение в каждом цикле. Пример:

for (i = 1, s = 1; i <= 5; i++) s=i; // значение 5! = 1*2*3*4*5 = 120*

В данном примере перед началом цикла выполняется два действия: *i=1* и *s=1*. Цикл выполняется за 5 итераций, в каждой из которых переменная *i* увеличивается на 1. Тело цикла может состоять из нескольких операций:

```
s = a = 0;  
dx = 0.01;  
for (i = 0; i < 100; i++) { // интеграл x²dx от 0 до 1  
    x = a + i * dx;  
    s += x*x*dx;  
} // значение s = 0.32835
```

Оператор цикла с предусловием имеет вид:

while (<условие>) <тело цикла>;

Тело цикла выполняется, если только выполняется заданное условие:

```
s = x = 0;
while (x <= 100) { // сумма ряда 1/x, где x изменяется от 1 до 100
    x++;
    s += 1 / x;
} // значение s = 5.1873...
```

Оператор цикла с постусловием имеет вид:

```
do <тело цикла> while (<условие>);
```

Тело цикла продолжает выполняться, если выполняется заданное условие:

```
s = x = 0;
do { // сумма ряда 1/x от 1 до 1/100
    x++;
    s += 1 / x;
} while (x <= 100); // значение s = 5.1873...
```

Цикл с постусловием отличается от цикла с предусловием тем, что выполняется хотя бы один раз.

Оператор прерывания цикла *break* позволяет дострочно выйти из цикла:

```
var sum = 0;
while (true) { // бесконечный цикл
    var value = +prompt("Введите число", "");
    if (!value) break; // выход, если число не введено
    sum += value;
} // sum - сумма введенных чисел
```

Оператор продолжения цикла *continue* позволяет продолжить цикл сначала:

```
for (i = 0; i < 10; i++) {
    if (i % 2 == 0) continue;
    alert(i); // вывод нечетных значений
}
```

В случае если нужно прервать/продолжить выполнение сразу нескольких циклов, вместе с командами *break/continue* используются метки, представляющие собой идентификатор с символом : (точка с запятой). Метка ставится перед началом прерываемого/продолжаемого цикла:

```
c = 0;
label: for (i = 0; i < 3; i++) {
    for (j = 0; j < 3; j++) {
        c += i + j;
        if (c > 10) break label;
    }
} // значение c = 0+1+2+1+2+3+2 = 11
```

Массивы

Простой массив в языке Javascript — это переменная, содержащая однотипные элементы. Для создания массива и обращения к элементам используются квадратные скобки [] и индекс (порядковый номер) элемента. Нумерация элементов массива начинается с 0. Пример:

```
var empty = [ ]; // пустой массив
var odd = [0, 2, 4, 6, 8]; // массив из 5 чисел
var list = ["Иван", "Андрей", "Наталья"]; // массив из 3 слов
var c = odd[2]; // значение c = 4;
alert(list[1]); // вывод "Андрей"
```

Поскольку массив является встроенным объектом типа *Array*, для объявления можно использовать его конструктор:

```
var a = new Array(2, 3, 4); // массив из 3 чисел
```

Многомерный массив объявляется с помощью вложенных квадратных скобок []. Элементы такого массива адресуются двумя (или более) подряд идущими индексами. Пример:

```
var a = [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ]; // матрица 3x3
alert(a[1][1]); // вывод элемента 5
```

К элементам массива также можно обращаться с помощью именованных ключей. Такой массив называется *ассоциативным*. Ассоциативный массив объявляется с помощью фигурных скобок { }, имена ключей завершаются символом : (двоеточие):

```
var earth = { name: "Земля", period: 365.25, radius: 6378};
```

Ассоциативный массив, так же как и обычный, является объектом типа *Array* и может быть объявлен с помощью конструктора:

```
var earth = new Array();
```

К элементам ассоциативного массива можно обратиться по имени — свойству через символ . (точка) или квадратные скобки []. Обращение по числовому индексу не разрешается:

```
earth.name = "Земля";
earth.radius = 6378;
alert(earth.name); // вывод Земля
alert(earth["radius"]); // вывод 6378
```

Для перебора всех ключей ассоциативного массива используется цикл *for* и команда *in*:

```
for (key in earth) {
    alert("Ключ "+key+" значение " + earth[key]);
}
```

Функции

Функции являются основными строительными блоками программы. Часть

функций, такие как вышеуказанные *alert(<сообщение>)* или *Math.sqrt(x)*, являются встроенными в объектную модель Javascript и не требуют объявления.

Пользовательские функции требуют обязательного объявления:

```
function <имя_функции> ([<аргументы>,...]) {  
    <тело функции>  
}
```

Функция вызывается по ее имени, например:

```
// объявление функции  
function sum(a, b) { // сумма двух чисел  
    return (a+b); // возвращаемое значение  
}  
var c = sum(2,3); // вызов функции  
alert("результат:" + c); // результат: 5
```

При вызове функции *sum* переменным *a* и *b* присваиваются значения 2 и 3. Команда *return* возвращает значение функции в основную программу. Возвращаемое значение присваивается переменной *c* и выводится на экран.

Если аргументы не заданы, то они принимают значения *undefined*. Пример:

```
var c = sum(2); // аргумент b опущен  
alert("результат:" + c); // результат: NaN (2 + undefined)
```

Необязательные аргументы должны размещаться в конце списка. Для задания значения аргумента по умолчанию, используется оператор *||* (ИЛИ):

```
// объявление функции  
function sum(a, b) { // сумма двух чисел  
    b = b || 0; // аргумент по умолчанию, значение 0  
    return (a+b);  
}  
var c = sum(2); // вызов функции  
alert("результат:" + c); // результат: 2
```

Если команда *return* указана без параметра, то функция возвращает значение *undefined*. Команда *return* может быть указана в любом месте функции.

Все переменные, объявленные внутри функции, являются *локальными*. Область их действия распространяется только на заданную функцию, из основной программы они не видны. Аргументы функции также являются локальными переменными.

Переменные, объявленные вне функции, являются *глобальными*. Область их действия распространяется на всю программу, они видны из любой части программы. Локальные переменные перекрывают глобальные. Пример:

```
var a = 0; // глобальная переменная  
var b = 1; // глобальная переменная  
// описание функции
```

```
function f(c) { // локальная переменная
  var a = 2; // локальная переменная
  alert("a=" + a); // вывод локальной переменной a=2
  alert("b=" + b); // вывод глобальной переменной b=1
  alert("c=" + c); // вывод локальной переменной c=3
}
f(3); // вызов функции, с присваивается 3
alert("a=" + a); // вывод глобальной переменной a=0
alert("b=" + b); // вывод глобальной переменной b=1
alert("c=" + c); // ошибка: глобальная переменная c не определена
```

Другой способ объявления функции — функциональное выражение:

```
<имя_функции> = function(<аргументы>,...) {
  ... тело функции ...
}
```

Например:

```
var sum = function(a, b) { // сумма двух переменных
  return (a+b);
}
var c = sum(2,3);
alert(c); // вывод 5
```

Вызываются функциональные выражения так же, как и обычные функции. Функциональные выражения должны объявляться до их использования, для обычных функций это правило не обязательно.

Функциональное выражение, которое не присваивается переменной называется *анонимной функцией*. Анонимная функция вызывается в том месте, где она описана. Пример:

```
function ask(question, yes, no) { // описание функции
  if (confirm(question)) yes();
  else no();
}
ask("Вы согласны?", // вызов функции
  function() { alert("Да"); }, // анонимная функция
  function() { alert("Нет"); } // анонимная функция
);
```

Объекты

Для описания пользовательского объекта используются фигурные скобки { }, в которых перечисляются свойства, подобно ассоциативному массиву:

```
person = { name: "Иван", family: "Иванов", age: 20 };
```

Также для описания объекта можно использовать встроенный класс Object:

```
person = new Object();
person.name = "Иван";
```

```
person.family = "Иванов";  
person.age = 20;
```

Для инициализации объектов можно использовать функцию-конструктор, в которой можно также описать публичные свойства и методы класса с помощью специальной ссылки на класс *this*:

```
function Person(name, family, age) {  
    this.name = name;  
    this.family = family;  
    this.age = age;  
    this.getName = function() { alert(this.name); }  
}
```

После описания класса экземпляры объектов можно задавать с помощью команды *new*:

```
var alex = new Person("Алексей", "Сидоров", 22);  
alert(alex.age); // вывод 22  
alex.getName(); // вывод Алексей
```

Для добавления нового свойства заданного объекта достаточно указать его в программе:

```
alex.group = 144; // появилось новое свойство group
```

Если свойства не существует, то при обращении к нему вернется значение *undefined*:

```
alert(alex.sename); // undefined
```

Для проверки существования свойства можно использовать команду *in*:

```
if ("sename" in alex) alert("Отчество:" + alex.sename);
```

Для удаления свойств заданного объекта можно использовать команду *delete*:

```
delete alex.group; // удалено свойство group
```

Для удаления объекта целиком используется указатель на пустую строку *null*:

```
alex = null; // удаление объекта alex
```

Для добавления новых свойств и методов, общих для всех объектов класса можно их явно указать в программе с помощью свойства *prototype*:

```
Person.prototype.group = 144; // у всех объектов свойство group=144  
Person.prototype.show = function() { // новая функция show()  
    alert("ФИО:" + this.family + " " + this.name);  
}
```

Простое функциональное наследование класса можно указать в конструкторе, вызвав функцию *call* базового класса:

```
function Student(name, family, age, group) {  
    Person.call(this, name, family, age); // конструктор базового класса
```

```

    this.group = group;
    this.show = function() { alert("Студент группы "+group); }
}
var alex = new Student("Алексей", "Сидоров", 22, 144);
alex.show(); // вывод Студент группы 144

```

В данном примере класс *Student* наследует свойства и методы вышеописанного класса *Person*, добавляя новое свойство *group* и метод *show()*.

Также можно указать отношение наследования между классами с помощью свойства *prototype*:

```

function Student(name, family, age, group) {
    Person.apply(this, arguments); // вызов конструктора базового класса
    this.group = group;
}
Student.prototype = Object.create(Person.prototype);
var alex = new Student("Алексей", "Сидоров", 22, 144);

```

Конструктор базового класса *Person* вызывается с помощью специальной функции *apply*, которая передает ей ссылку на текущий класс *this* и полный список аргументов *arguments*. Обратите внимание, что перед *Object* отсутствует команда *new*.

Копирование объектов, а также массивов, осуществляется по ссылке. Это означает, что при присвоении объекта нескольким переменным и изменении одной из них будут изменены и все остальные, так как объект, на который они указывают, изменен. Пример:

```

var alex = new Person("Алексей", "Сидоров", 22); // конструктор
var green = alex; // присвоение green
green.age = 25; // изменение возраста
alert(alex.age); // вывод 25 (объект изменен)

```

Таким образом для осуществления полноценного копирования необходимо использовать явное присвоение всех элементов объекта:

```

for (key in alex) {
    green[key] = alex[key];
}

```

Приватные свойства и методы в классе описываются с помощью локальных переменных и вложенных функций в конструкторе:

```

function Person(name, family, age) { // конструктор
    var kinder = false; // приватное свойство
    this.name = name; // публичное свойство
    this.family = family; // публичное свойство
    this.age = age; // публичное свойство
    function setKinder() { // приватный метод
        kinder = (age < 16);
    }
}

```

```

    }
    this.isKinder = function() { // публичный метод
        setKinder(); // вызов приватного метода
        if (kinder) alert("Да");
        else alert("Нет");
    }
}
var alex = new Person("Алексей", "Иванов", 22);
alex.isKinder(); // вывод Нет

```

Обработка исключений

При выполнении некоторых операций могут возникнуть ошибки времени выполнения. Для их перехвата используется механизм обработки исключений. Механизм обработки исключений реализован с помощью операторов:

```
try { <программа> } catch(e) { <обработка> } [ finally { <завершение> } ]
```

Блок <программа> представляет собой исходный код, в котором осуществляется перехват ошибки. Блок <обработка> служит для обработки ошибки. Необязательный блок <завершение> представляет собой команды, выполняемые после блоков <программа> и <обработка> в любом случае.

Пример:

```

try {
    var x = prompt("введите число");
    var q = x*a;
    alert("Результат:"+q);
} catch(e) {
    alert(e.message);
}

```

При попытке выполнить программу будет выведено сообщение *a is not defined*, поскольку переменная *a* неопределена. Описание ошибки хранится в свойстве *message* объекта *Error*.

Генерацию исключений можно вызвать принудительно с помощью команды *throw*:

```

try {
    var x = prompt("введите число");
    if (x<0) throw "Число должно быть больше 0";
    var q = Math.sqrt(x);
    alert("Корень:"+q);
} catch(e) {
    alert(e);
}

```

В этом случае в переменную *e* записывается непосредственно генерируемое сообщение.

Объекты Javascript

Язык Javascript является объектно-ориентированным и использует объекты для своей работы. Фактически, массивы, строки и даже числа являются встроенными объектами *Array*, *String*, *Number* и *Boolean*. Встроенными объектами также являются даты *Date*. Кроме того, существуют глобальные объекты, такие как *Math*, *JSON* и др.

Объект Number. Представляет собой обычное число. Основные методы:

Number(<параметр>) — представляет собой конструктор числа. В качестве параметра может передаваться любое значение. Пример:

```
var a = new Number("12.24"); // a = 12.24
```

toFixed(n) — преобразование к действительному числу с *n* значащими цифрами после точки:

```
var a = 3.1415926;  
a.toFixed(3); // 3.142
```

toExponential(n) — преобразование к числу в экспоненциальной форме с *n* значащими цифрами после точки:

```
var a = 86421;  
a.toExponential(3); // 8.642e+4
```

toPrecision(n) — преобразование числа с *n* общим количеством значащих цифр:

```
var a = 123.456;  
a.toPrecision(4); // 123.5
```

toString(x) — преобразование числа в строковое представление в системе счисления по основанию *x*:

```
var a = 12;  
a.toString(16); // символ c
```

Объект String. Представляет собой текстовую строку, заключенную в кавычки или апострофы. При необходимости, строка может содержать управляющие последовательности: \n — перевод строки, \DDD — десятичный код символа \xHH — шестнадцатичный код символа, \uNNNN — символ в кодировке Unicode, \' - экранирование апострофа, \" - экранирование кавычки, \\ - экранирование обратной косой черты. Основные методы и свойства:

String(<параметр>) - представляет собой конструктор строки. В качестве параметра инициализации может передаваться любое значение. Пример:

```
var str = new String(12.45);  
alert(str.length); // вывод 5
```

Свойство *length* определяет длину строки в символах:

```
"Привет!".length; // 7
```

Обращение по индексу строки $[i]$ возвращает символ на позиции i . Нумерация позиций начинается с 0:

```
"Привет!"[3]; // символ "в"
```

Метод `charCodeAt(i)` возвращает десятичный код символа строки на позиции i (в кодировке Unicode):

```
"Привет!".charCodeAt(3); // 1074
```

`fromCharCode(x1,...,xN)` — возвращается текстовая строка с кодами символов $x1,...,xN$:

```
String.fromCharCode(1055,1088,1080,1074,1077,1090,33); // Привет!
```

`concat(<строка>)` — сложение (конкатенация) строк. Действует как оператор + (плюс):

```
"Иван".concat("Иванович"); // ИванИванович
```

`indexOf(<подстрока>[, i])` — поиск позиции подстроки в заданной строке, начиная с позиции i (по умолчанию 0) от начала строки. Если подстрока не найдена, функция возвращает значение -1:

```
alert("Математика".indexOf("а")); // вывод 1
```

`lastIndexOf(<подстрока>[, i])` — поиск позиции подстроки в заданной строке, начиная с позиции i (по умолчанию 0) от конца строки. Если подстрока не найдена, функция возвращает значение -1:

```
alert("Математика".lastIndexOf("а")); // вывод 9
```

`substr(i [, <длина>])` — возвращает подстроку строки начиная от позиции i заданной длины. В случае, если длина не указана или превышает длину строки, подстрока обрезается до конца строки.

```
var a = "Математика";  
alert(a.substr(2,4)); // вывод тема
```

`substring(i [, j])` — возвращает подстроку строки начиная от позиции i до j (не включая его). Если позиция j не указана, то возвращает подстроку от позиции i до конца.

```
var a = "Математика";  
alert(a.substring(2,6)); // вывод тема
```

`slice(i [, j])` — подобно функции `substring` возвращает подстроку строки начиная от позиции i до j (не включая его). В отличие от `substring`, может иметь отрицательное значение второго параметра j — в этом случае, индекс отсчитывается от конца строки:

```
var a = "Математика";  
alert(a.slice(2,-4)); // вывод тема
```

`split(<разделитель>[, n])` — разбивает строку на массив из n элементов по заданному разделителю. Количество элементов массива — необязательный

параметр. Если в качестве разделителя указывается пустая строка, то исходная строка разбивается на массив отдельных символов. Пример:

```
var a = "до конца";  
var b = a.split(" "); // Массив ["до", "конца"]  
var c = a.split("",4); // Массив ["д","о","","к"]
```

toLowerCase() - приводит строку к нижнему регистру:

```
var a = "Привет";  
alert(a.toLowerCase()); // вывод: привет
```

toUpperCase() - приводит строку к верхнему регистру:

```
var a = "мур";  
alert(a.toUpperCase()); // вывод: МИР
```

Оборачивающие методы позволяют содержимое строки обернуть в HTML теги:

big() - увеличение шрифта (*<big>строка</big>*);

small() - уменьшение шрифта (*<small>строка</small>*);

bold() - полужирный (*строка*);

italic() - курсив (*<i>строка</i>*);

sub() - подстрочный индекс (*_{строка}*);

sup() - надстрочный индекс (*^{строка}*);

fontcolor(<цвет>) - цвет шрифта (*строка*);

fontsize(<разм>) - размер шрифта (*строка*);

link(<ссылка>) - гиперссылка (*строка*).

Объект Array. Представляет собой массив. Основные свойства и методы:

Array(<n>) - конструктор массива из *n* элементов. Если параметр *n* является числом, то резервируется память для элементов массива. Все элементы принимают значения *undefined*. Для инициализации элементов массива их необходимо передать в качестве параметров:

```
var a = new Array(5); // массив из 5 пустых элементов  
var b = new Array(1,2,3); // массив из 3 элементов, b = [1,2,3]
```

Обращение к элементам массива производится по их индексу *[i]*. Нумерация элементов начинается с 0:

```
var b = [1, 2, 3];  
alert(b[1]); // вывод 2
```

Свойство *length* определяет размер массива в элементах:

```
var b = [1, 2, 3];  
alert(b.length); // вывод 3
```

Уменьшение свойства *length* приводит к уменьшению размера массива и освобождению памяти от элементов. Аналогично, увеличение свойства *length*

резервирует дополнительную память под элементы. Также увеличение свойства *length* происходит автоматически при обращении к несуществующим индексам.

```
var b = [1, 2, 3, 4, 5]; // размер массива 5
b.length = 3; // размер массива 3, b = [1, 2, 3]
alert(b); // вывод 1,2,3
b[4] = 5; // размер массива снова 5, b = [1, 2, 3, undefined, 5]
alert(b); // вывод 1,2,3,,5
```

Методы, не изменяющие исходный массив:

concat(<массив>) - объединение массивов в один. Возвращается объединенный массив. Пример:

```
var a = [1, 2, 3];
var b = new Array("Привет", "мир");
var c = a.concat(b); // c = [1, 2, 3, "Привет", "мир"]
```

join(<разделитель>) - объединение всех элементов массива в одну строку с указанным разделителем:

```
var a = new Array(1, 2, 3);
var str = a.join(""); // str = "123"
```

slice(i [, j]) — создание нового массива из исходного с индексами элементов от *i* до *j* (не включая его). Если параметр *j* не задан, то новый массив создается до конца исходного массива. При отрицательных значениях второго параметра *j* индекс отсчитывается от конца исходного массива:

```
var a = [1, 3, 2, 4, 5];
var b = a.slice(1,-1); // b = [3, 2, 4]
```

Методы, изменяющие исходный массив:

push(<элемент>) - добавление элемента в конец массива.

pop() - извлечение последнего элемента массива.

unshift(<элемент>) - добавление элемента в начало массива.

shift() - извлечение первого элемента массива. Примеры:

```
var a = [1, 2, 3];
a.push(5); // добавление элемента в конец, a = [1, 2, 3, 5]
var b = a.pop(); // извлечение последнего элемента, b = 5, a = [1, 2, 3]
a.unshift(0); // добавление элемента в начало, a = [0, 1, 2, 3]
var c = a.shift(); // извлечение первого элемента, b = 0, a = [1, 2, 3]
```

Из-за особенностей реализации сдвиги *shift* и *unshift* выполняются значительно медленнее, чем операции со стеком *push* и *pop*.

reverse() - изменяет порядок элементов в массиве на противоположный:

```
var a = [1, 2, 3, 4, 5]
a.reverse(); // массив a = [5, 4, 3, 2, 1]
```

sort(<функция>) - сортировка элементов массива с помощью пользовательской функции сравнения. Если функция не задана, то используется строковая сортировка по возрастанию:

```
var a = [1, 8, 15, 6, 3];  
a.sort(); // массив a = [1, 15, 3, 6, 8]
```

При использовании пользовательской функции сравнения в качестве сортируемых элементов массива используются два ее аргумента (например, *a* и *b*), а ее имя передается в качестве аргумента функции *sort*. Функция сравнения в зависимости от параметров должна возвращать одно из трех значений: меньше 0 — если параметр *b* следует за *a*, равен 0 — если параметры равны, больше 0 — если параметр *a* следует за *b*. Пример:

```
var a = [1, 8, 15, 6, 3];  
function comp(a, b) { return (a-b); }  
a.sort(comp); // массив a = [1, 3, 6, 8, 15];
```

splice(i, len [, <список>]) - срез массива, вырезает из исходного массива *len* элементов, начиная от *i*, возвращает их, и заменяет их в исходном массиве новым списком элементов, если он задан. Пример:

```
var a = ["в", "час", "вечерний", "прохладный"];  
var x = a.splice(1, 2, "день", "осенний");  
alert(x); // вывод: час,вечерний  
alert(a); // вывод: в,день,осенний,прохладный
```

Объект Date. Представляет собой дату. Основные свойства и методы:

Date(<дата>) - конструктор даты. Имеются различные способы инициализации даты:

- строка формата ISO 8601 Extended "YYYY-MM-DDTHH:mm:ss.msZ" (год, месяц, день, T - символ-разделитель, часы, минуты, секунды, миллисекунды, Z - символ для GMT или часовое смещение +hhmm);
- укороченная строка ISO "YYYY-MM-DD" (год, месяц, день);
- число msUTC (число миллисекунд POSIX от 01.01.1970 г. GMT+0);
- массив [YYYY, NM, DD, HH, mm, ss] (NM - порядковый номер месяца, начиная с 0).
- укороченный массив [YYYY, NM, DD].

Примеры:

```
var a = new Date("2015-12-26"); // 26 декабря 2015 года  
var b = new Date("2016-04-01T12:30:00"); // 01.04.2016 г., 12 ч 30 мин  
var d = new Date(1461917448231); // 29.04.2016 г., 11:10:48  
var c = new Date(2016, 03, 12, 16, 45, 00); // 12.04.2016 г., 16 ч 45 мин
```

Для установки текущего значения даты и времени параметры в конструкторе можно не указывать:

```
var a = new Date(); // текущая дата и время
```

Для извлечения параметров из даты используются следующие методы:

getFullYear() - получение номера года (от 1970 и далее);

getMonth() - получение порядкового номера месяца (от 0 — января до 11 - декабря);

getDate() - получение числа месяца (от 1 до 31);

getDay() - получение номера дня недели (от 0 — воскресения до 6 - субботы);

getHours() - получение часа (от 0 до 23);

getMinutes() - получение минут (от 0 до 59);

getSeconds() - получение секунд (от 0 до 59);

getMilliseconds() - получение локального времени в миллисекундах POSIX.

Пример:

```
var a = new Date("2016-04-01T12:30:00");
```

```
var year = a.getFullYear(); // 2016
```

```
var month = a.getMonth(); // 3
```

```
var d = a.getDate(); // 1
```

```
var hours = a.getHours(); // 12
```

Все методы извлекают данные для текущей временной зоны. Для мирового времени GMT+0 имеются аналогичные функции с префиксом UTC, например *getUTCHourse()* и т.д. Кроме того, существуют методы, использующие мировое время:

getTime() - получение мирового времени в миллисекундах POSIX GMT+0;

getTimezoneOffset() - возвращение разницы между локальным и мировым временем в минутах:

```
var h = new Date().getTimezoneOffset(); // -180 для Москвы (3 часа)
```

Для установки параметров даты используются соответствующие методы с префиксом *set*:

setFullYear(YYYY) - установка номера года;

setMonth(NM) - установка порядкового номера месяца (0 — январь);

setDate(DD) - установка числа месяца;

setDay(day) - установка номера дня недели (0 — воскресенье);

setHours(HH) - установка часа;

setMinutes(mm) - установка минут;

setSeconds(ss) - установка секунд;

setMilliseconds(ms) - установка локального времени в миллисекундах POSIX.

При установке значений, в общем случае, остальные исходные параметры даты

не изменяются. Однако если значения выходят за пределы диапазона, то параметры автоматически корректируются. Пример:

```
var a = new Date(); // текущее дата и время
a.setFullYear(2015); // установлен 2015 год
a.setMonth(1); // установлен месяц февраль
a.setDate(29); // поскольку в 2015 году февраль не високосный,
alert(a); // будет выведена дата 01.03.2015 и текущее время
```

Все методы устанавливают данные для текущей временной зоны. Для мирового времени GMT+0 имеются аналогичные функции с префиксом UTC, например `setUTCHour(HH)` и т.д. Кроме того, имеется метод, использующий мировое время:

`setTime(msUTC)` - установка мирового времени в миллисекундах POSIX GMT+0.

Для строкового преобразования даты и времени используются следующие методы:

`toString()` - преобразование даты и времени в текстовую строку. Вид строки зависит от версии браузера, например:

```
var a = new Date("2016-04-01"); // установка даты
alert(a.toString()); // вывод Fri Apr 01 2016 03:00:00 GMT +0300 MSK
```

`toDateString()` - преобразование только даты в текстовую строку.

`toTimeString()` - преобразование только времени в текстовую строку.

`toLocaleString([<язык>, <опции>])` - преобразование даты и времени в текстовую строку в локализованном формате. Язык и ассоциативный массив опций — необязательные параметры, устанавливающие формат строки (не поддерживается Microsoft Internet Explorer). Пример:

```
var a = new Date("2016-04-01"); // установка даты
var ops = {
    day: 'numeric',
    month: 'long',
    year: 'numeric'
}
alert(a.toLocaleString("ru",ops)); // вывод 1 апреля 2016 г.
```

`Date.parse(<дата>)` - возвращает число миллисекунд POSIX на заданную дату. Дата указывается в вышеуказанном формате ISO 8601 Extended. Пример:

```
var a = Date.parse("2016-04-01T12:45:12+0300");
alert(a); // вывод 1459503912000
```

При использовании объектов `Date` в числовом контексте возвращается количество миллисекунд. Это свойство часто используется для анимации элементов страницы, однако лучше применить отдельный метод `now()`:

`Date.now()` - возвращает число миллисекунд POSIX. Этот метод работает

быстрее, чем *getTime()*, так как при этом сам объект *Date* не создается. Его можно использовать даже для профилирования программ. Пример:

```
var a = Date.now();
for (var x = 1, s = 1; x < 100000; x++) s += 1/(x*x); // сумма ряда 1/x2
var b = Date.now();
alert("сумма:"+s+", "+(b - a)+" ms"); // 2.644..., 3 ms
```

Обратите внимание, что оператор *new* перед *Date* в примере не ставится.

Управление временем. Профилирование.

В программах, требующих профилирования, вместо объекта *Date* лучше использовать специальное свойство *performance* (не поддерживается Microsoft Internet Explorer):

performance.now() - возвращает число миллисекунд POSIX. Метод работает быстрее, чем *Date.now()*, и точнее в 1000 раз:

```
var a = performance.now();
for (var x = 1, s = 1; x < 100000; x++) s += 1/(x*x); // сумма ряда 1/x2
var b = performance.now();
alert("сумма:"+s+", "+(b - a)+" ms"); // 2.644..., 2.315 ms
```

Для управления временем в Javascript имеется несколько полезных глобальных методов, которые с успехом применяются при анимации объектов.

<переменная> = setTimeout(<функция>, <задержка>) - устанавливает пользовательскую функцию, которая будет выполнена с заданной задержкой (в миллисекундах). Пример:

```
var func = function() { alert("1 секунда"); };
setTimeout(func, 1000); // вывод сообщения через 1 секунду
```

clearTimeout(<переменная>) - отменяет выполнение пользовательской функции, установленной методом *setTimeout*. Переменная представляет собой идентификатор таймера *setTimeout*. Пример:

```
var timer = setTimeout(function() { alert("1 секунда") }, 1000);
clearTimeout(timer);
```

В данном случае сообщение не будет выведено.

<переменная> = setInterval(<функция>, <задержка>) - устанавливает пользовательскую функцию, которая будет периодически выполняться с заданной задержкой (в миллисекундах). Пример:

```
var time = 0;
var func = function() {
    time++;
    alert(time+" секунда");
};
setInterval(func, 1000); // вывод сообщения каждую секунду
```

В отличие от метода *setTimeout*, пользовательская функция в данном примере будет выполняться больше одного раза, увеличивая каждый раз счетчик секунд на 1.

clearInterval(<переменная>) - отменяет выполнение пользовательской функции, установленной методом *setInterval*. Переменная представляет собой идентификатор таймера *setInterval*. Пример:

```
var time = 0;
var func = function() {
    time++;
    alert(time+" секунда");
    if (time >= 5) clearInterval(timer); // отмена выполнения
};
var timer = setInterval(func, 1000); // вывод 5 сообщений через секунду
```

Объект Math. Является глобальным объектом. Представляет собой библиотеку, которая включает наиболее распространенные математические константы и функции. Основные свойства:

E — константа *e* (2.71828...);

PI — константа *π* (3.14159...);

Основные методы:

abs(x) — модуль числа *x*;

acos(x) — арккосинус числа *x* (угол в радианах);

asin(x) — арксинус числа *x* (угол в радианах);

atan(x) — арктангенс числа *x* (угол в радианах);

ceil(x) — округление числа *x* до целого в большую сторону;

cos(x) — косинус угла *x* (в радианах);

exp(x) — экспонента числа *x*;

floor(x) — округление числа *x* до целого в меньшую сторону;

log(x) — натуральный логарифм числа *x*;

max(a, b) — максимальное значение *a* и *b*;

min(a, b) — минимальное значение *a* и *b*;

pow(x, y) — вычисление функции *x* в степени *y*;

random() - вычисление случайного числа в диапазоне от 0 до 1;

round(x) — округление числа *x* до ближайшего целого;

sin(x) — синус угла *x* (в радианах);

sqrt(x) — квадратный корень числа *x*;

tan(x) — тангенс угла *x* (в радианах);

Все константы и функции возвращают значения с двойной точностью. Примеры:

```
var sqare = r * r * Math.PI; // площадь круга с радиусом r
var y = Math.sin(x); // функция y = sin(x);
```

Регулярные выражения и обработка строк.

Регулярные выражения являются мощным инструментом обработки данных и предоставляют основные операции поиска/замены строк. Регулярное выражение не является встроенным типом данных. Для объявления регулярного выражения используется так называемый *паттерн* — строка, содержащая символы поиска/замены и специальные конструкции, заключенный в символах / (слеш). Например:

```
var reg = /ab\d/; // строка, содержащая символы a, b и любую цифру
```

Объект RegExp. Обеспечивает обработку регулярных выражений в соответствии со стандартом Perl Compatible Regular Expressions (PCRE). Основные методы:

RegExp(<паттерн> [, <ключи>]) - является базовым конструктором. Паттерн представляет собой шаблон поиска/замены (регулярное выражение). Ключи представляют опциональную строку из любой комбинации символов:

g — глобальный поиск (свойство *global*);

i — поиск не зависит от регистра (свойство *ignoreCase*);

m — многострочный поиск (свойство *multiline*).

Наиболее распространенные шаблоны

Символ	Значение
\w	любой алфавитно-цифровой символ (лат.), аналог [A-Za-z0-9_]
\d	любой символ цифры, аналог [0-9]
\s	пробельный символ
.	любой символ
\	отмена специального значения последующего символа, например \. - символ десятичной точки
^	начало строки
\$	конец строки
\n	символ перевода строки
\W	любой не алфавитно-цифровой символ (лат.), аналог [^A-Za-z0-9_]
\D	любой нецифровой символ, аналог [^0-9]
\S	любой непробельный символ
\b	граница слова

+	повторение предыдущего символа 1 и более раз
*	повторение предыдущего символа 0 и более раз
?	после символа: повторение предыдущего символа 0 или 1 раз; после квантификаторов *, + задает нежадный поиск следующего символа
a b	один из символов a или b
(abc)	группирующие скобки (результат сохраняется в массиве)
[abc]	набор - один из символов a, b или c
[a-z]	набор - один из символов диапазона от a до z
[^abc]	набор - любой символ, кроме a, b или c
(?:abc)	группировка без сохранения результата
{n}	повторение предыдущего символа n раз
{n,}	повторение предыдущего символа n и более раз
{n,m}	повторение предыдущего символа от n до m раз
\1... \9	ссылка на одну из запомненных группировок

Примеры использования паттернов:

```
var ip_addr = /^d{1,3}\.d{1,3}\.d{1,3}\.d{1,3}$/; // IP адрес
var url = /^www\.\w+\.ru$/; // домен 2 уровня в рунете
var email = /^w+@w+\.\w{2,4}$/; // адрес электронной почты
var str = /\bmup\b/; // слово "мур"
```

test(<строка>) - проверка результатов регулярного выражения. Поиск начинается со свойства *lastIndex* объекта *RegExp* (если оно установлено).
Пример:

```
if (/s/.test("привет, мур")) {
    alert("строка содержит пробелы");
}
```

exec(<строка>) - проверка совпадений регулярного выражения. Содержимое записывается в массив. Если совпадений не найдено, возвращается *null*. Пример поиска слогов "ма" и "ка" в строке "математика":

```
var reg = /[мк]a/g;
var str = "математика";
while ((res = reg.exec(str)) != null) {
    var msg = "Найдено: " + res[0] + ". ";
    msg += "Позиция в строке: " + reg.lastIndex;
    alert(msg);
}
```

В данном примере выводится 3 совпадения на 2, 6 и 10 позициях строки.

Вышеописанный класс *String* также имеет методы поиска/замены строк. Аргументом этих методов являются регулярные выражения:

search(<паттерн>) - поиск подстроки по регулярному выражению. Метод аналогичен методу *test()* объекта *RegExp*. Возвращает позицию подстроки в строке или -1, если подстрока не найдена. Пример:

```
var str = "привет, мир";
var pos = str.search(/s/);
if (pos!=-1) {
    alert("строка содержит пробел на позиции "+pos);
}
```

match(<паттерн>) - проверка совпадений по регулярному выражению. Метод аналогичен методу *exec()* объекта *RegExp*. Возвращает массив совпадений строки по шаблону. Пример:

```
var reg = /[мк]a/g;
var str = "математика";
res = str.match(reg);
alert(res); // вывод ма,ма,ка
```

<новая_строка> = <строка>.replace(<паттерн>, <замена>) - замена подстроки, найденной в исходной строке в соответствии с регулярным выражением. Пример:

```
var res = "математика".replace(/^мате/, "инфор");
alert(res); // вывод информатика
```

Отладка программы

Для отладки Javascript программы и вывода значений переменных обычно используется функция *alert*, выводящее текстовое сообщение. Однако в некоторых случаях вывод окна влияет на поведение сайта и такой способ является нежелательным.

Гораздо более удобным и гибким инструментом является вывод с помощью встроенного метода *console.log(<сообщение>)*. С помощью этого метода можно выводить сообщения в отладочную панель браузера (для этого она должна быть предварительно включена). Пример:

```
console.log("Тестовое сообщение");
```

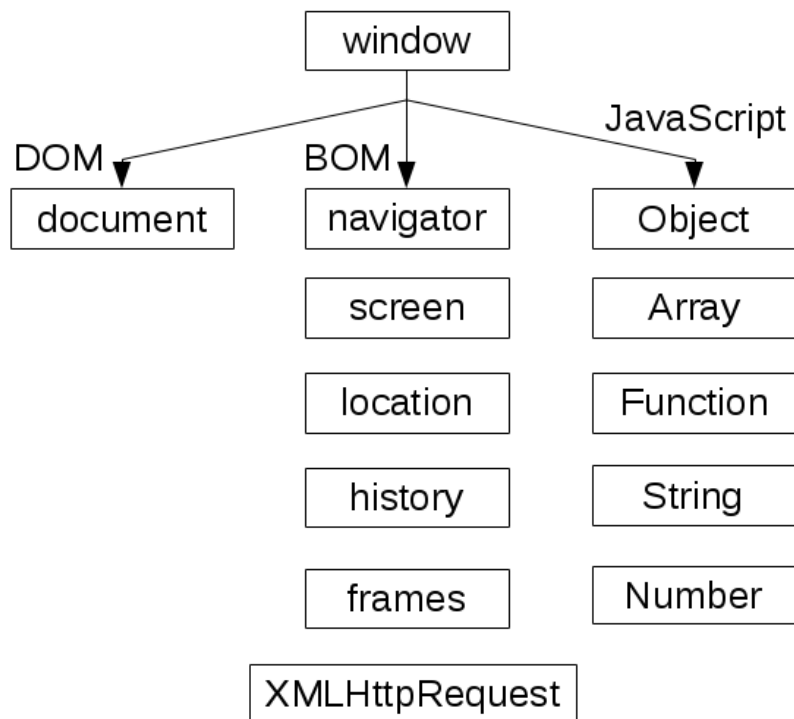
Для включения отладочной панели в большинстве браузеров используется комбинация клавиш *<Ctrl> + <Shift> + <J>*. Кроме этого, для отладки программ в браузере Mozilla Firefox используется расширение Firebug, которое включается клавишей *<F12>*.

Для вывода значений объектов и массивов программы также можно использовать метод *console.dir(<переменная>)*. В отличие от метода

console.log, этот способ позволяет вывести структуру объектной переменной со всеми значениями полей и наименованиями методов.

3.2 Объектная модель браузера

Javascript использует стандартную объектную модель, вершиной которой является объект *window*. Он также является главным объектом всей иерархии Javascript.



Объектная модель браузера (Browser Object Model, BOM) состоит из следующих основных объектов:

1) *window* - окно браузера. Основные свойства:

window.innerWidth - ширина окна браузера;

window.innerHeight - высота окна браузера;

window.opener - ссылка на родительское окно;

window.status - содержимое панели статуса браузера;

window.screenX и *window.screenY* - координаты левого верхнего угла окна браузера относительно экрана;

window.pageXOffset и *window.pageYOffset* - координаты смещения скроллинга HTML страницы в браузере;

Основные методы объекта *window*:

window.open(<URL>, <win> [, params]) - открытие нового окна, где:

URL - адрес отображаемого HTML документа;

win - имя создаваемого окна (или *_blank* - для нового окна);

params - текстовая строка с параметрами окна.

К объекту окна можно также обратиться с помощью инициализированной переменной. Пример:

```
var params="menubar=yes,location=yes,resizable=yes,scrollbars=yes,  
    status=yes,width=500,height=450";  
var win = window.open("http://yandex.ru", "Yandex", params);  
win.focus(); // получить фокус нового окна
```

window.close() - закрытие окна;

window.blur() - сделать окно неактивным;

window.focus() - сделать окно активным;

window.print() - распечатать содержимое окна;

window.moveTo(x,y) - перемещение ЛВУ окна в позицию с координатами (x,y);

window.resizeTo(w,h) - изменение размера окна, где *w* - ширина, *h* - высота;

Поскольку *window* является глобальным объектом, то к нему относятся и все глобальные методы, такие как *alert()*, *prompt()* и др, а также . Например:

```
window.alert("Сообщение"); // вывод сообщения
```

2) *navigator* - объект свойств браузера и операционной системы. Основные свойства:

navigator.appName - имя браузера;

navigator.appVersion - версия браузера;

navigator.userAgent - полная информация о браузере;

navigator.platform - информация об операционной системе пользователя.

3) *screen* - объект экрана. Основные свойства:

screen.width - ширина экрана;

screen.height - высота экрана;

screen.colorDepth - глубина цвета;

4) *location* - URL адрес загруженной HTML страницы. Основные свойства:

location.href - строка URL;

location.hostname - домен URL адреса;

location.pathname - путь относительно корня сайта;

location.search - строка запроса;

Основные методы объекта *location*:

location.assign(<url>) - загрузить документ по данному URL;

location.reload([true]) - перезагрузить документ по текущему URL. Если установлен параметр *true*, то документ перезагружается с сервера, иначе - браузер будет использовать свой кэш.

location.replace(<url>) - замена текущего документа на страницу по

указанному URL. В отличие от использования *assign()*, страница не сохраняется в истории браузера;

location.toString() - строковое представление URL;

5) *history* - объект списка посещенных страниц браузером. Основные свойства:

history.length - количество элементов списка;

Основные методы объекта *history*:

history.back() - загрузить предыдущий URL;

history.forward() - загрузить последующий URL;

history.go(<index>) - перейти на указанный URL в списке по индексу *<index>*. Например, выполнение команды *history.go(-1)* аналогично команде *history.back()* (перейти на предыдущий адрес).

3.3 Доступ к элементам страницы

Объект *document* хранит объектную структуру HTML документа (Document Object Model, DOM). С помощью него можно получить доступ к отдельным HTML тегам, добавлять и изменять их содержимое, а также определять поведение этих элементов.

Основные свойства объекта *document*:

document.cookie - возвращает ключики, связанные с HTML документом. Это ассоциативный массив вида *ключ=значение*, хранящийся в браузере пользователя. Обычно используется для хранения пользовательских настроек для каждого конкретного сайта или отдельной страницы.

document.head - возвращает тег заголовка *<head>*.

document.title - возвращает заголовок страницы *<title>*.

document.forms - возвращает массив, содержащий все формы на странице.

document.images - возвращает массив изображений на HTML странице.

document.links - возвращает массив гиперссылок на HTML странице.

document.lastModified - возвращает время последнего изменения документа.

document.documentElement - возвращает корневой тег документа *<html>*.

document.body - возвращает тег основного тела документа *<body>*.

Основные методы объекта *document*:

document.createElement(<meг>) - создает HTML элемент по имени тега;

document.getElementById(<Id>) - возвращает HTML элемент по его идентификатору (атрибуту *id*);

document.getElementsByName(<имя>) - возвращает массив HTML элементов по имени (атрибуту *name*).

document.getElementsByTagName(<тег>) - возвращает массив HTML элементов по имени тега.

document.getElementsByClassName(<класс>) - возвращает массив HTML элементов по имени класса (атрибуту *class*).

document.querySelector(<селектор>) - возвращает первый подходящий HTML элемент по его CSS селектору. Параметр *<селектор>* описывается так же, как в стилевой таблице. Также он может включать несколько CSS селекторов, разделенных запятыми. Пример (возвращает элемент *div* с атрибутом *id="test"*):

```
var el = document.querySelector("div#test");
```

document.querySelectorAll(<селектор>) - возвращает массив HTML элементов по их CSS селектору.

document.write(<строка>) и *document.writeln(<строка>)* - запись HTML строки в поток вывода браузера.

Пример:

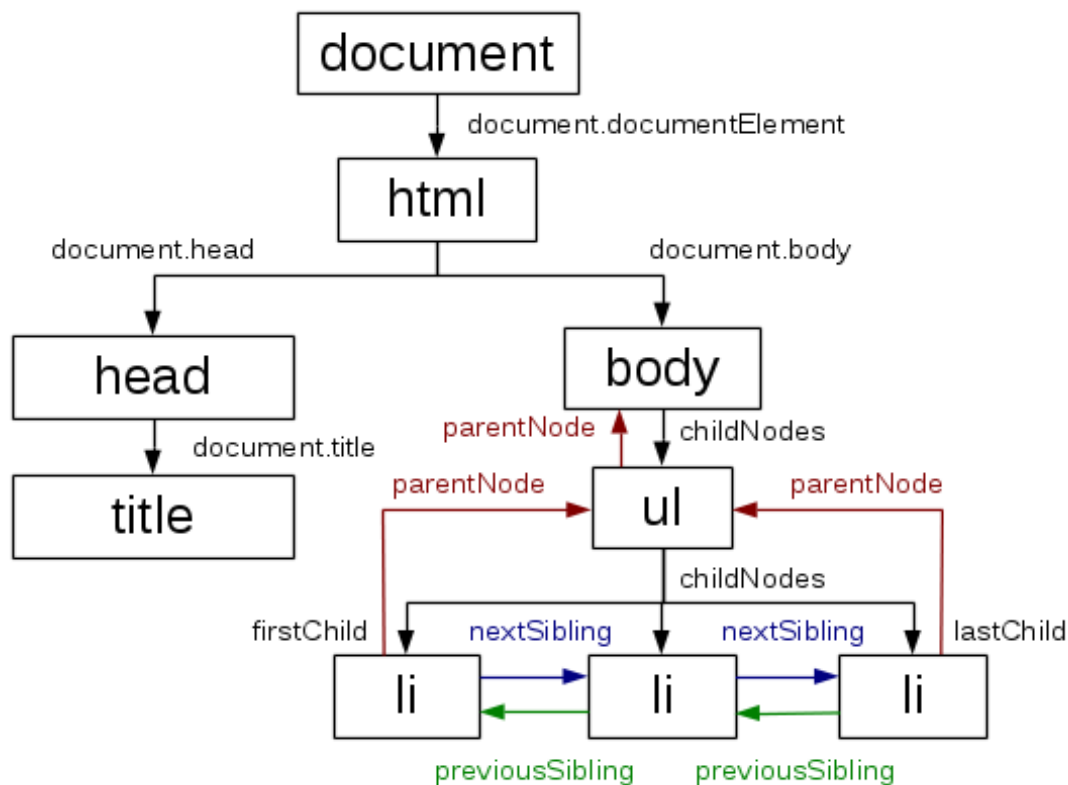
```
<div id="test"></div>
<script>
    var dd = document.getElementById("test");
    dd.innerHTML = "Тестовые данные";
</script>
```

После выполнения скрипта тег *<div>* будет иметь вид:

```
<div id="test">Тестовые данные</div>
```

Свойства и методы элементов DOM

Для каждого HTML элемента в DOM создается соответствующий объект. Все объекты DOM имеют набор стандартных свойств и методов и образуют древовидную иерархию. Кроме того, некоторые объекты (например объекты формы) имеют дополнительные свойства и методы, определяющиеся их назначением.



Общие основные свойства элементов DOM:

attributes - возвращает массив атрибутов;

childNodes - возвращает массив потомков;

parentNode - возвращает родительский элемент;

firstChild - возвращает первого потомка;

lastChild - возвращает последнего потомка;

nextSibling - возвращает последующий соседний элемент;

previousSibling - возвращает предыдущий соседний элемент;

nodeName или *tagName* - имя узла (тега);

nodeType - тип узла (1 - элемент, 2 - атрибут, 3 - текстовый узел);

nodeValue или *data* - получить (установить) значение текстового узла;

innerHTML - HTML содержимое элемента (без тега);

outerHTML - HTML содержимое элемента (вместе с тегом);

textContent - текстовое содержимое элемента (без тегов);

className - получить/установить имя класса элемента;

id - получить/установить идентификатор элемента;

style - получить/установить CSS стиль элемента

Текстовый узел является прямым потомком элемента, поэтому для обращения к его содержимому необходимо обратиться к свойству *childNodes*:

```
var text = document.body.childNodes[0].data;  
alert(text); // вывод содержимого тега <body>.
```

Общие основные методы элементов DOM:

getAttribute(<атрибут>) - получить значение атрибута;

setAttribute(<атрибут>,<значение>) - установить значение атрибута;

removeAttribute(<атрибут>) - удалить атрибут;

appendChild(<элемент>) - добавление элемента потомка;

removeChild(<элемент>) - удаление элемента потомка;

hasChildNodes() - возвращает *true*, если узел имеет потомков;

focus() - делает элемент активным;

blur() - делает элемент неактивным;

click() - активирует обработчик нажатия правой кнопки мыши на элементе.

Пример использования методов DOM:

```
<ul id="test"></ul>  
<script>  
    var dd = document.getElementById("test");  
    var el = Array();  
    for (var i=0; i<3; i++) {  
        el[i] = document.createElement("li");  
        el[i].innerHTML = "место " + (i+1);  
        dd.appendChild(el[i]);  
    }  
</script>
```

После выполнения скрипта список `` на HTML странице будет иметь вид:

```
<ul id="test">  
    <li>место 1</li>  
    <li>место 2</li>  
    <li>место 3</li>  
</ul>
```

Специальные свойства элементов DOM

Кроме общих свойств многие элементы DOM имеют также специальные свойства. Использование этих свойств позволяет обращаться к HTML атрибутам соответствующих тегов, получать и устанавливать их значения. Например, для обращения к файлу изображения, устанавливаемому с помощью атрибута *src* тега *img* используется соответствующее свойство *img.src* и т.д.:

```
  
<script>  
    function change() { // функция изменения изображения
```



```
        document.images[0].src = "picture2.png";
    }
</script>
```

Для элементов HTML таблиц используются следующие свойства и методы:

table.rows - возвращает массив строк таблицы;
table.createCaption() - создает тег заголовок таблицы *<caption>*;
table.deleteCaption() - удаляет тег заголовок таблицы *<caption>*;
table.createTHead() - создает тег шапку таблицы *<thead>*;
table.deleteTHead() - удаляет тег шапку таблицы *<thead>*;
table.createTFoot() - создает тег подвал таблицы *<tfoot>*;
table.deleteTFoot() - удаляет тег подвал таблицы *<tfoot>*;
table.insertRow(i) - добавляет строку таблицы в позицию *i*;
table.deleteRow(i) - удаляет *i* строку из таблицы;
tr.cells - возвращает массив ячеек таблицы;
tr.rowIndex - возвращает текущую позицию строки в таблице;
tr.insertCell(i) - добавляет ячейку строки в позицию *i*;
tr.deleteCell(i) - удаляет *i* ячейку из строки;
td.cellIndex - возвращает текущую позицию ячейки в строке.

Пример:

```
var table = document.getElementById("idTable");
var row = table.insertRow(0); // вставка пустой строки <tr> в начало
var cell1 = row.insertCell(0); // вставка пустой ячейки <td> в позицию 0
cell1.innerHTML = "новая ячейка";
```

Большая часть специальных свойств HTML элементов, необходимых для обработки данных будет рассмотрена в п. 3.4.

Использование стилей

Свойство *style* элемента DOM является объектом и позволяет изменять внешний вид элемента в соответствии с CSS правилами. Большинство CSS атрибутов представлены соответствующими строковыми свойствами элемента *style*. Для обращения к этим свойствам используется следующее правило: если CSS атрибут состоит из нескольких слов, разделенных дефисом, то соответствующее свойство *style* также будет составлено из этих слов, но записанных подряд. При этом все слова, следующие после дефиса, должны начинаться с заглавной буквы. Например, CSS атрибуту *font-weight* для указанного элемента в javascript будет соответствовать свойство *element.style.fontWeight*, атрибуту *text-align* будет соответствовать свойство *element.style.textAlign* и др. Используя свойство *style* вместе с обработкой событий можно сделать динамическую стилизацию страницы, например,

скрыть или показать некоторые элементы на странице в зависимости от выбора того или иного пункта меню, изменить цветовую гамму элементов и др.

Пример использования объекта *style*:

```
document.body.style.backgroundColor = "red"; // установить красный фон  
var dd = document.getElementById("test");  
dd.style.visibility = "hidden"; // скрыть элемент с идентификатором test
```

Полный перечень свойств объекта *style* приведен в документации javascript [].

Обработка событий

Управление событиями на странице является одной из важнейших возможностей javascript, необходимых для создания интерактивного интерфейса веб-приложений.

События - это функции обработчики, которые могут быть привязаны к элементам HTML страниц. Обработчики выполняются после того, как произойдет их активация. Разные типы событий имеют разные активирующие действия.

Основные виды событий:

onblur - элемент стал неактивным;

onchange - содержимое элемента изменено;

onclick - на элементе произведен щелчок мыши;

ondblclick - на элементе произведен двойной щелчок мыши;

onerror - при загрузке элемента произошла ошибка;

onfocus - элемент стал активным;

onkeydown - нажата клавиша;

onkeypress - произошло нажатие на клавиатуру;

onkeyup - нажатая клавиша отпущена;

onload - элемент полностью загружен;

onmousedown - на элементе нажата клавиша мыши;

onmouseout - курсор мыши вышел за пределы элемента;

onmouseover - курсор мыши наведен на элемент;

onmouseup - на элементе отпущена клавиша мыши;

onreset - форма сброшена;

onresize - изменен размер документа;

onscroll - содержимое элемента прокручено;

onselect - текст элемента выделен;

onsubmit - форма передана на сервер;

onunload - страница закрыта (для тега *<body>*).

Чтобы задать обработчик необходимо связать его с HTML элементом. Для этого существует несколько способов:

1) Через соответствующий атрибут HTML тега *on<событие>*:

```
<div id="test" onclick="alert('Клик')">Нажми меня</div>
```

Данный способ наиболее простой, но смешивает javascript код с HTML разметкой, поэтому он недостаточно эффективный.

2) Через свойство элемента DOM *on<событие>* в javascript:

```
document.getElementById("test").onclick = function() { alert('Клик') }
```

Этот способ позволяет вынести всю обработку в javascript, но при этом на каждое событие можно задать только один обработчик. Способ используется в большинстве простых программ.

3) С помощью отдельного метода, рекомендованного консорциумом W3C *addEventListener(<событие>, <обработчик>[, <фаза>])*:

```
document.getElementById("test").addEventListener("click", message);  
function message() { alert('Клик') }
```

Это наиболее гибкий способ задания обработчиков, он используется для сложных приложений. Для удаления обработчика можно использовать метод *removeEventListener(<событие>, <обработчик>[, <фаза>])*:

```
document.getElementById("test").removeEventListener("click", message);
```

Если параметр *<фаза>* установлен в *true*, то при срабатывании во вложенных элементах обработчик можно вызвать в фазе "захвата", иначе - в фазе "всплывания" (см. ниже).

В старых версиях браузера Microsoft Internet Explorer (до 9) вместо *addEventListener* и *removeEventListener* использовались аналогичные функции *attachEvent* и *removeEvent*.

За информацию о произошедших событиях отвечает специальный DOM объект *events*, который можно передать в функцию обработчика и использовать его свойства. События хранятся в отдельных атрибутах *events*. Список атрибутов событий:

altKey - нажата клавиша *<Alt>* во время вызова события;

button - нажата клавиша мыши (0 - левая, 1 - средняя, 2 - правая) во время вызова события;

clientX - горизонтальные координаты указателя мыши относительно границ документа во время вызова события;

clientY - вертикальные координаты указателя мыши относительно границ документа во время вызова события;

ctrlKey - нажата клавиша *<Ctrl>* во время вызова события;

screenX - горизонтальные координаты указателя мыши относительно границ экрана во время вызова события;

screenY - вертикальные координаты указателя мыши относительно границ экрана во время вызова события;

shiftKey - нажата клавиша *<Shift>* во время вызова события;

target - возвращает элемент DOM, который вызвал событие;

type - возвращает имя события.

Используя эти атрибуты из объекта *events* можно извлечь много полезной информации о его состоянии. Объект *events* всегда передается в функцию обработчика первым параметром. Например:

```
function message(event) {  
    alert("Имя события:" + event.type);  
}
```

Порядок срабатывания событий

На одно и то же событие может реагировать не только указанный DOM элемент, но и элементы, в которые он вложен. Это свойство часто используется при обработке большого количества вложенных однотипных элементов. В этом случае обработчик можно повесить на родительский тег и ловить все события, происходящие в потомках. После отработки событий потомков, будут отработаны события на родительском теге. Этот способ называется фазой "всплытия", он действует в браузерах по умолчанию. Если какой-либо обработчик хочет остановить всплытие и не выпускать событие наружу, для объекта *events* используется метод *stopPropagation()*:

```
element.onclick = function(event) {  
    event.stopPropagation()  
}
```

При использовании другого способа обработки - фазы "захвата" события будут вызываться и обрабатываться в обратной последовательности: от родительского тега - к потомкам.

Консорциумом W3C определена универсальная модель обработки вложенных событий, в которой сначала используется фаза "захвата", а затем "всплытия". Таким образом, разработчик веб-приложения может самостоятельно определить, в какой фазе должен выполняться тот или иной обработчик с помощью установки параметра *<фаза>* функции *addEventListener*.

Для некоторых событий определены стандартные действия браузера, например, вызов контекстного меню по нажатию правой кнопки мыши. Для отмены действия браузера по умолчанию используется метод *preventDefault()*. После этой команды можно задавать свой обработчик события:

```
element.onclick = function(event) {  
    event.preventDefault();
```

```
    // код обработчика события  
}
```

3.4 Обработка форм

Существует три способа взаимодействия пользовательского интерфейса веб-приложения (frontend) с серверной частью веб-приложения (backend):

- с помощью параметров URL адреса (*GET* запрос);
- с помощью элементов форм;
- с помощью асинхронного запроса AJAX (см. п.3.5).

В первом случае, для взаимодействия пользователя с веб-приложением используются гиперссылки, содержащие параметры URL строки, например:

```
<a href="http://domain.ru/index.php?login=ivan&password=1234">  
Авторизация</a>
```

В данном случае взаимодействие с веб-приложением осуществляется с помощью *GET* запроса по адресу *http://domain.ru/index.php* с параметрами *login=ivan* и *password=1234*. Гиперссылки используются в случае, когда все параметры имеют фиксированные значения.

Для взаимодействия пользователя с веб-приложением с помощью интерактивных элементов форм: списков, текстовых полей ввода, кнопок и др., последние размещаются или прикрепляются на форме. В этом случае может использоваться как *GET* (по умолчанию), так и *POST* метод запроса. URL адрес приложения указывается с помощью атрибута *action*.

Перед отправкой формы на сервер данные можно проверить на корректность ввода с помощью специальной функции - обработчика, связанного с кнопкой ввода. Этот способ называется валидацией содержимого формы, производимой, как правило, на стороне клиента.

Доступ к свойствам и методам элементов форм осуществляется с помощью интерфейса DOM:

```
document.<имя_формы>.<имя_элемента>.<свойства>
```

Данные формы, введенные пользователем, отправляются на сервер с помощью метода *submit()* применяемого к форме. Метод вызывается автоматически при нажатии кнопки типа *submit* или при нажатии клавиши *<Enter>* в текстовом поле ввода. Также метод *submit()* можно вызвать программным способом в обработчике формы. Пример валидации формы:

```
<form name="form1" action="http://domain.ru/index.php">  
    <label>Логин <input type="text" name="login"></label>  
    <label>Пароль <input type="password" name="passwd"></label>  
    <input type="button" value="Ввод" onclick="go()">  
</form>  
<script>  
    function go() {
```

```
    if (document.form1.login.value=="") {  
        alert("логин не введен"); return false; }  
    if (document.form1.passwd.value=="") {  
        alert("пароль не введен"); return false; }  
    document.form1.submit(); // отправить данные на сервер  
}  
</script>
```

Основные свойства и методы объекта *form*:

form.elements - массив всех элементов формы;

form.length - количество элементов формы;

form.name - имя формы;

form.<имя_элемента> - обращение к элементу формы;

form.action - адрес URL программы обработчика на сервере;

form.submit() - отправка данных формы на сервер;

form.reset() - сброс элементов формы в исходное состояние.

Для всех элементов форм можно обратиться к родительскому объекту формы с помощью ссылки: *<имя_элемента>.form*

Объект ввода *input* содержит следующие основные свойства и методы:

input.type - тип элемента ввода;

input.value - значение элемента ввода;

input.size - размер элемента ввода;

input.name - имя элемента ввода;

input.accept - значение атрибута *accept* (для *<input type="file">*);

input.checked - значение атрибута *checked* (для *<input type="checkbox">* и *<input type="radio">*);

input.defaultChecked - возвращает *true*, если элемент выбран по умолчанию (для *<input type="checkbox">* и *<input type="radio">*);

input.maxLength - значение атрибута *maxlength* (для *<input type="text">* и *<input type="password">*);

input.readOnly - возвращает *true*, если содержимое поля нельзя изменить (для *<input type="text">* и *<input type="password">*);

input.select() - выделение текста элемента (для *<input type="text">* и *<input type="password">*).

Кроме того, к строковым свойствам элементов формы можно применять строковые функции или операции сравнения, например:

```
if (document.form1.passwd.length < 6) {  
    alert("Слишком короткий пароль"); return false;
```

```

}
if (document.form1.passwd.value != document.form1.passwd2.value) {
    alert("Пароли не совпадают"); return false;
}

```

Объект ввода текста *textarea* содержит следующие основные свойства и методы:

textarea.value - текст элемента ввода;

textarea.name - имя элемента ввода;

textarea.defaultValue - текст элемента ввода по умолчанию;

textarea.disabled - значение атрибута *disabled*;

textarea.readOnly - значение атрибута *readonly*;

textarea.rows - количество строк элемента (значение атрибута *rows*);

textarea.cols - количество позиций элемента (значение атрибута *cols*);

textarea.select() - выделение текста элемента.

Объект списка *select* содержит следующие основные свойства и методы:

select.length - количество пунктов в выпадающем списке;

select.multiply - значение атрибута *multiply*;

select.name - имя объекта списка;

select.options - массив всех элементов списка;

select.selectedIndex - позиция выбранного элемента списка (начиная с 0);

select.add(<элемент>[, <позиция>]) - добавление элемента в выпадающий список (по умолчанию, элемент добавляется в конец списка);

select.remove(<позиция>) - удаление элемента из выпадающего списка.

Элемент списка *option* содержит следующие основные свойства и методы:

option.index - текущая позиция элемента в списке (начиная с 0);

option.selected - значение атрибута *selected*;

option.value - значение элемента списка.

Пример валидации списка:

```

<form name="form1" action="http://domain.ru/index.php">
  <label>Владение языками
    <select name="language" multiple>
      <option value="EN">английский</option>
      <option value="DE">немецкий</option>
      <option value="FR">французский</option>
    </select>
  </label>

```

```


</form>
<script>
    function go() {
        var select = document.form1.language;
        var c = false; // признак выбора
        for (var i=0; i<select.length; i++) { // цикл по всем опциям
            if (select.options[i].selected) c = true;
        }
        if (!c) { alert("Ни один язык не выбран"); return false; }
        document.form1.submit(); // отправить данные на сервер
    }
</script>

```

3.5 Асинхронная передача данных

Основой взаимодействия пользователя с веб-приложением является его интерфейс, который формируется на основе элементов HTML страниц. В период развития интернет технологий в конце прошлого века основной проблемой являлось обновление части данных на странице. Для корректного отображения интерфейса всю HTML страницу приходилось перегружать целиком, нагружая при этом браузер, сервер и каналы передачи данных, что значительно замедляло работу веб-приложения. Использование технологии фреймов для этой цели себя не оправдало из-за сложности организации взаимодействия фреймовых окон и громоздкости всей конструкции. Технология встраиваемых страниц SSI (*Server Side Including*) оказалась неудобной в настройке и конфигурировании и с развитием веб-программирования необходимость в ней отпала. Одновременно с этим появилась насущная необходимость обновления отдельных элементов без перезагрузки всей HTML страницы.

Для частичного обновления информации на странице был разработан стандарт обмена данными AJAX (*Asynchronous Javascript And Xml*). Для обмена данными с сервером с помощью AJAX используется Javascript объект *XmlHttpRequest*, который умеет отправлять запрос и получать ответ с сервера.

Обмен данными с сервером в AJAX может быть как синхронный, так и асинхронный. При синхронном обмене обработчик будет ожидать ответа от сервера после каждого запроса. При асинхронном обмене после запроса выполнение Javascript сценариев на стороне клиента не прерывается, а ответ от сервера вызывает событие *onreadystatechange*, которое затем перехватывается клиентским обработчиком. Синхронная обработка данных в веб-приложениях применяется крайне редко.

Основные методы объекта *XmlHttpRequest*:

new() - создание объекта. Используется в качестве инициализации AJAX.

open(<метод>,<адрес>,<асинхр>[,<логин>,<пароль>]) - настройка параметров соединения с сервером. Параметр *<метод>* задает один из двух методов запроса *GET* или *POST*. *<адрес>* представляет собой URL адрес запроса. Параметр *<асинхр>* задает способ синхронизации: *true* - асинхронный, *false* - синхронный. Дополнительные параметры *<логин>* и *<пароль>* при необходимости задают аутентификационные данные доступа к серверу. Метод не открывает соединение. Пример:

```
var xhr = new XMLHttpRequest(); // создается HTTP запрос
xhr.open('GET', 'http://domain.ru/phones.json', true); // асинхронный метод
send([<запрос>])
```

 - отправка запроса на сервер. Дополнительный параметр *<запрос>* является телом запроса и используется с методом *POST*. В случае использования метода *GET*, параметры запроса передаются в URL строке метода *open()*. Пример:

```
var xhr = new XMLHttpRequest(); // создается HTTP запрос
xhr.open('GET', 'http://domain.ru/phones.json', true);
xhr.send(); // запрос отправляется на сервер
```

abort() - прерывание выполнения запроса.

onreadystatechange() - асинхронное событие, вызываемое несколько раз в ходе сеанса запроса, по числу возможных состояний. Событие связывается с обработчиком ответа. Текущее состояние запроса определяется свойством *readyState*:

```
xhr.onreadystatechange = function() {
  if (xhr.readyState != 4) return; // запрос не завершен
  if (xhr.status != 200) {
    alert(xhr.status + ': ' + xhr.statusText); // запрос неуспешен
  } else {
    // запрос успешен...
    alert(xhr.responseText); // вывод ответа сервера
  }
}
```

setRequestHeader(<имя>,<знач>) - устанавливает заголовок запроса, где *<имя>* - наименование заголовка, *<знач>* - его значение. Отменить установленный заголовок нельзя, так же как и установить некоторые виды заголовков, контролируемых браузером. Пример:

```
xhr.setRequestHeader('Content-Type', 'application/json');
```

getResponseHeader(<имя>) - получает заголовок ответа, где *<имя>* - наименование заголовка (кроме заголовка Set-Cookie):

```
xhr.getResponseHeader('Content-Type');
```

getAllResponseHeaders() - получает все заголовки ответа, разделенные символами перевода строки. Заголовок возвращается в виде одной строки.

Основные свойства объекта *XmlHttpRequest*:

readyState - текущее состояние запроса (0 - начальное состояние, 1 - вызвана функция *open*, 2 - получены заголовки ответа, 3 - загружается тело ответа, 4 - запрос завершен). Обработка ответа сервера допустима только в случае завершенного запроса (*readyState=4*).

status - код ответа сервера (стандартные коды: 200 - запрос успешен, 302 - документ перемещен, 401 - клиент не авторизован, 403 - запрос отклонен, 404 - документ не найден, 500 - внутренняя ошибка сервера и т.д.).

statusText - текстовое описание статуса сервера.

responseText - текстовый ответ сервера (содержание запрашиваемого документа).

responseXML - ответ сервера в формате XML (если запрашивался документ в формате XML).

timeout - задает максимальную продолжительность асинхронного запроса (в миллисекундах). При превышении этого времени генерируется событие *ontimeout*, например:

```
xhr.timeout = 30000;  
xhr.ontimeout = function() {  
    alert("Запрос превысил 30 секунд")  
}
```

Преимущества AJAX

Экономия трафика. Использование AJAX позволяет значительно сократить трафик при работе с веб-приложением, так что вместо загрузки всей страницы загружается только изменившаяся часть, или только данные в формате JSON или XML.

Уменьшение нагрузки на сервер. AJAX позволяет снизить нагрузку на сервер в несколько раз за счет уменьшения трафика и числа HTTP соединений.

Улучшение отклика интерфейса. При загрузке частичных данных через AJAX пользователь сразу получает результат своих действий без обновления страницы.

Недостатки AJAX

Отсутствие интеграции с браузерами. Динамически создаваемые страницы не регистрируются браузером в истории посещения страниц, поэтому не работает кнопка «Назад», также невозможно сохранение закладок на просматриваемые материалы. Эти проблемы решаются с помощью скриптов.

Динамически загружаемое содержимое не доступно поисковикам. Старые методы индексации содержимого и учета статистики сайтов становятся неактуальными из-за того, что поисковые машины не могут выполнять JavaScript. Эта проблема решается с помощью организации альтернативного доступа к содержимому сайта для поисковых машин.

Усложнение клиентской части веб-приложений. Перераспределяется логика

обработки данных — происходит выделение и частичный перенос на сторону клиента процессов получения и форматирования данных. В целом это усложняет контроль целостности форматов и типов и приводит к усложнению и удорожанию веб-приложений.

Требуется включенный JavaScript в браузере. JavaScript может быть выключен из соображений безопасности, кроме того, AJAX страницы труднодоступны неполнофункциональным браузерам и поисковым роботам.

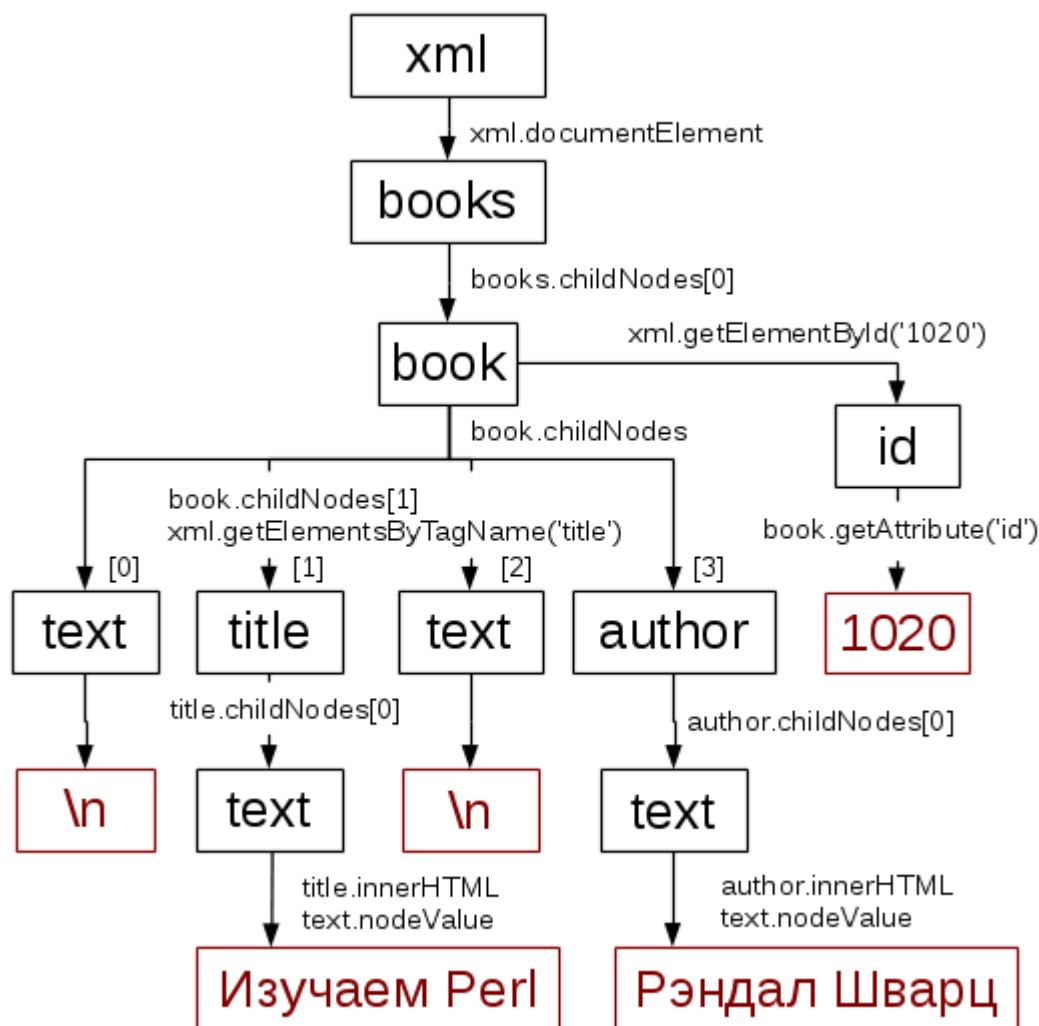
Риск несанкционированного использования кроссдоменных запросов. Результат работы AJAX запроса может являться JavaScript кодом. Если методы объекта XMLHttpRequest действуют только в пределах одного домена, то для тега `<script>` такого ограничения нет, что может привести к различным коллизиям, в том числе .

В целом AJAX удобен для программирования интерактивных веб-приложений, административных панелей и других инструментов, которые используют динамические данные.

3.6 Форматы обмена данными

Для обмена данными между серверной и клиентской частями веб-приложения наиболее часто используются следующие форматы: XML (*eXtensible Markup Language*) и JSON (*JavaScript Object Notation*). Данные могут быть приняты от сервера в результате AJAX запроса и объекта *XmlHttpRequest*.

Документ XML представляет собой дерево, состоящее из узлов - элементов. Для доступа к XML узлам используются вышеописанные функции DOM, применяемые ранее к тегам HTML страниц. Одним из важных достоинств представления данных в виде XML является его универсальность. Стандарт обмена данными в формате XML используется при разработке сложных модульных веб-приложений, в стеке технологий Java, при проектировании и использовании веб-сервисов, в серверном протоколе SOAP. Большинство языков программирования содержит встроенные библиотеки обработки XML документов, поддерживающие интерфейс DOM - парсеры.



Недостаток использования формата XML заключается в некоторой избыточности передаваемых данных (каждый XML узел представляет собой пару текстовых тегов), а также трудоемкость и ресурсоемкость разбора данных XML формата с помощью функций DOM или SAX (Simple API XML).

Пример использования XML данных

Содержание файла books.xml на сервере:

```

<?xml version="1.0"?>
<books>
  <book id="1020">
    <title>Изучаем Perl</title>
    <author>Рэндал Шварц</author>
  </book>
  <book id="1021">
    <title>HTML, скрипты и стили</title>
    <author>Дунаев В.</author>
  </book>
</books>

```

Обработка подгружаемого списка из XML файла:

```

<ol id="books" onclick="select()">Список книг</ol>
<script>
function select() {
    var xhr = new XMLHttpRequest();
    xhr.open('GET', 'http://domain.ru/books.xml', false);
    xhr.send(); // выполнение запроса
    if (xhr.status != "200") { // обработка ошибки
        alert("Ошибка сервера"); return false;
    }
    var xml = xhr.responseXML;
    var ol = document.getElementById("books");
    var book = Array();
    for(var i=0; i < xml.childNodes.length; i++) {
        var title = xml.getElementsByTagName("title")[i].innerHTML;
        var author = xml.getElementsByTagName("author")[i].innerHTML;
        book[i] = document.createElement("li");
        book[i].innerHTML = author.italics() + title;
        ol.appendChild(book[i]);
    }
}
</script>

```

В результате после нажатия на текстовый элемент мы получаем вывод списка книг из заданного XML файла.

Использование формата JSON

Альтернативой использования стандарта XML является представление данных в формате JSON. Это удобный краткий текстовый формат описания Javascript объектов. Данные, передаваемые от серверной части веб-приложения, конвертируются в объектные переменные Javascript и могут непосредственно использоваться в программных обработчиках на стороне клиента.

Данные в формате JSON (в соответствии с RFC 4627) представляют собой набор терм, содержащих:

- Javascript объекты { ... };
- массивы [...];
- строки, заключенные в двойные кавычки;
- числа;
- логические значения true/false;
- пустое значение null.

В основе представления JSON данных являются Javascript объекты, содержащие свойства и их значения. Содержание файла books.json на сервере:

```

{
    "book": [
        {
            "id": 1020,
            "title": "Изучаем Perl",

```

```

        "author": "Рэндал Шварц"
    },
    {
        "id": 1021,
        "title": "HTML, скрипты и стили",
        "author": "Дунаев В."
    }
]
}

```

В отличие от объектов Javascript, все атрибуты JSON данных должны представлять собой текстовые строки, заключенные в кавычки. Апострофы не допускаются. Такое представление короче и проще, чем XML, и потому завоевало большую популярность у разработчиков. Для чтения и разбора данных JSON в языке Javascript используется глобальный объект **JSON**.

Основные методы объекта JSON:

JSON.parse(<строка>[,<функция>(ключ,значение)]) - преобразование текстовой строки JSON в объект Javascript. Дополнительный параметр **<функция>(ключ, значение)** вызывает функцию для каждого свойства объекта, которая должна вернуть измененное значение.

JSON.stringify(<объект>[,<массив>]) - преобразование (сериализация) объекта Javascript в текстовую строку формата JSON. Дополнительный параметр **<массив>** содержит массив заданных свойств для сериализации объекта.

Пример использования данных JSON:

```

<ol id="books" onclick="select()">Список книг</ol>
<script>
function select() {
    var xhr = new XMLHttpRequest();
    xhr.open('GET', 'http://domain.ru/books.json', false);
    xhr.send(); // выполнение запроса
    if (xhr.status != "200") { // обработка ошибки
        alert("Ошибка сервера"); return false;
    }
    var json_str = xhr.responseText;
    var ol = document.getElementById("books");
    var obj = JSON.parse(json_str, function(key,val) {
        if (key=="author") val = val.italics();
        return val;
    });
    var book = Array();
    for(var i=0; i < obj.book.length; i++) {
        var title = obj.book[i].title;
        var author = obj.book[i].author;
    }
}

```

```

    book[i] = document.createElement("li");
    book[i].innerHTML = author + title;
    ol.appendChild(book[i]);
  }
}
</script>

```

После нажатия на текстовый элемент мы, как и в предыдущем примере, получаем вывод списка книг из заданного файла формата JSON.

3.7 Библиотека JQuery

Одной из самых популярных Javascript библиотек, использующихся в разработке веб-приложений, является библиотека JQuery. Если CSS отделяет визуализацию от структуры HTML, то JQuery отделяет поведение от структуры HTML документа. Эта библиотека позволяет значительно упростить и ускорить написание Javascript кода, сделать его более прозрачным и понятным. jQuery позволяет создавать анимацию, обработчики событий, значительно облегчает выбор элементов в DOM и создание AJAX запросов. Также эта библиотека является кроссбраузерной, что позволяет не заботиться о кроссбраузерной совместимости Javascript кода.

Для использования библиотеки JQuery в своих проектах необходимо скачать ее с официального сайта <https://code.jquery.com> и добавить ее на HTML страницу веб-приложения. Существует два варианта библиотеки JQuery: для разработчиков (uncompressed) и для готовых приложений (minified). Последний вариант имеет минимальный объем кода за счет удаления комментариев и пробелов из исходников и отличается от первой дополнительным расширением .min в имени файла.

Для добавления библиотеки JQuery на HTML страницу веб-приложения используется вставка скрипта в заголовок страницы в виде отдельного файла, например (для предварительной загрузки и локального использования):

```
<script src="jquery-2.2.4.min.js"></script>
```

или (для использования ссылки Google):

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.2.4/jquery.min.js">
</script>
```

В приведенном примере используется сжатая версия 2.2.4 библиотеки JQuery. Версии периодически обновляются, поэтому за этим нужно следить. На официальном сайте также содержится подробное описание функций API и возможностей, предоставляемых библиотекой.

Один из наиболее частых вариантов использования библиотеки JQuery, позволяющий избежать преждевременное выполнение кода до полной загрузки страницы, с использованием оборачивающей безымянной функции:

```
$(document).ready(function() {
```

// обработка данных

});

Вся работа с JQuery производится с помощью функции `$()`, что является синонимом *JQuery*.

Работу с JQuery можно разделить на 2 типа:

- Получение JQuery объекта с помощью функции `$()`. Например, передав в нее CSS-селектор, можно получить JQuery объект всех элементов HTML, попадающих под критерий и далее работать с ними с помощью различных методов JQuery объекта. В случае, если метод не должен возвращать какого-либо значения, он возвращает ссылку на JQuery объект, что позволяет вести цепочку вызовов методов.
- Вызов глобальных методов объекта `$`, например, итераторов для массива.

Таким образом, стандартный синтаксис выполнения команд JQuery:

`$(<селектор>).<метод>(<параметры>)`

Например, вызов `$()` функции со строкой селектора CSS возвращает объект *JQuery*, содержащий некоторое количество элементов HTML страницы. Эти элементы затем можно обработать методами JQuery:

```
$(document).ready(function() {  
    $("div.test").add("p.quote").addClass("blue").slideDown("slow");  
});
```

В данном примере находятся все элементы *div* с классом *test*, а также все элементы *p* с классом *quote*, и затем к ним добавляется класс *blue* и применяется метод *slideDown* - визуальный слайдер с параметром *slow*. В результате возвращается ссылка на исходный объект `$("div.test")`.

Все методы в JQuery делятся на следующие группы:

- методы для манипулирования DOM;
- методы для оформления элементов;
- методы для создания AJAX запросов;
- методы для создания эффектов;
- методы для привязки обработчиков событий.

Селекторы

С помощью CSS селекторов выбираются элементы на странице для последующего применения к ним определенных действий. Параметры селекторов в целом соответствуют синтаксису стандарта CSS, хотя и имеют некоторые отличия. Основные виды селекторов, поддерживаемые JQuery:

`$("*")` - выделить все элементы на странице;

`$("p")` - выделить все абзацы в тексте;

`$(".par")` - выделить все элементы с классом `class="par"`;

`$("#par")` - выделить элемент с идентификатором `id="par"`;

`$(this)` - выделить текущий элемент;

`$("p.par")` - выделить все абзацы с классом *class="par"*;

`$("p#par")` - выделить абзац с идентификатором *id="par"*;

`$(".head,.foot")` - выделить все элементы с классами *class="head"* и *class="foot"*;

`$("[src]")` - выделить все элементы, имеющие атрибут *src*;

`$("[src='1.jpg']")` - выделить все элементы с атрибутом *src="1.jpg"*;

`$("[src!='1.jpg']")` - выделить все элементы с атрибутом *src* не равным *'1.jpg'*;

`$("[src^='1.']")` - выделить все элементы с атрибутом *src* и началом *'1.'*;

`$("[src$='.jpg']")` - выделить все элементы с атрибутом *src* и окончанием *'jpg'*;

`$("[src*='jpg']")` - выделить все элементы с атрибутом *src*, содержащим *'jpg'*;

`$("ol#list li")` - выделить все элементы *li* внутри нумерованного списка *ol* с идентификатором *id="list"*;

`$("p > div")` - выделить все элементы *div* внутри родительского абзаца *p*;

`$(":input")` - выделить все элементы *input*;

`$(":button")` - выделить все элементы *input* с атрибутом *type="button"*;

`$(":text")` - выделить все элементы *input* с атрибутом *type="text"*;

`$(":password")` - выделить все элементы *input* с атрибутом *type="password"*;

`$(":radio")` - выделить все элементы *input* с атрибутом *type="radio"*;

`$(":checkbox")` - выделить все элементы *input* с атрибутом *type="checkbox"*;

`$(":file")` - выделить все элементы *input* с атрибутом *type="file"*;

`$("div:first")` - выделить первый элемент *div*;

`$("div:last")` - выделить последний элемент *div*;

`$("li:even")` - выделить все четные элементы списка *li*;

`$("li:odd")` - выделить все нечетные элементы списка *li*;

`$("li:eq(2)")` - выделить третий элемент списка *li* (нумерация с 0);

`$("li:lt(2)")` - выделить первый и второй элемент списка *li* (индекс меньше 2);

`$("li:gt(1)")` - выделить все элементы списка *li*, начиная с 3 (индекс больше 1);

`$(":header")` - выделить все заголовки *h1 - h6*;

`$(":animated")` - выделить все анимированные элементы;

`$(":contains('hello')")` - выделить все элементы, содержащие строку *'hello'*;

`$(":empty")` - выделить все элементы, не имеющие потомков;

`$(":hidden")` - выделить все скрытые элементы;

`$(":visible")` - выделить все видимые элементы.

Обработчики событий

После выбора элементов, можно задать обработчики событий и привязать их к данным элементам. Общий синтаксис обработчика:

```
$(<селектор>).<событие>(function() { ...код обработчика... } );
```

Например, для изменения текста раздела *div* по нажатию на кнопку *button* определим следующий обработчик:

```
$(document).ready(function() {  
    $(".button").click(function() { $(".div#par").html("новый текст"); });  
});
```

Основные методы событий:

click([<функция>]) - устанавливает обработчик клика мыши (или запускает его, при опущенном параметре *<функция>*);

dblclick([<функция>]) - устанавливает/запускает обработчик двойного клика мыши;

mouseenter([<функция>]) - устанавливает/запускает обработчик появления курсора мыши в области элемента (не обладает свойством всплытия);

mouseleave([<функция>]) - устанавливает/запускает обработчик выхода курсора мыши из области элемента (не обладает свойством всплытия);

hover(<функция1>[,<функция2>]) - устанавливает обработчики двух событий *mouseenter* и *mouseleave*. Если задан один параметр *<функция1>*, то он является обработчиком двух событий одновременно.

mousedown([<функция>]) - устанавливает/запускает обработчик нажатия кнопки мыши;

mouseup([<функция>]) - устанавливает/запускает обработчик отпускания кнопки мыши;

mousemove([<функция>]) - устанавливает/запускает обработчик перемещения мыши;

toggle(<функция1>,<функция2>[,<функция3>,...]) - устанавливает переключение между двумя и более обработчиками, вызываемыми по нажатию кнопки мыши.

trigger(<событие>) - вызывает заданное событие по его имени. Пример:

```
$(".form").trigger("submit"); // отправка данных формы на сервер
```

keydown([<функция(объект)>]) - устанавливает/запускает обработчик нажатия клавиши. Код клавиши передается в свойстве *объект.which*;

keyup([<функция(объект)>]) - устанавливает/запускает обработчик отпускания клавиши. Код клавиши передается в свойстве *объект.which*;

keypress([<функция(объект)>]) - устанавливает/запускает обработчик ввода символа с клавиатуры. Код символа передается в свойстве *объект.which*;

focus([<функция>]) - устанавливает/запускает обработчик получения фокуса элемента;

blur([<функция>]) - устанавливает/запускает обработчик потери фокуса элемента;

change([<функция>]) - устанавливает/запускает обработчик изменения элемента. Событие активируется, когда элемент ввода изменяет свое значение после получения фокуса. Пример:

```
$(":password").change( function() {  
    if ($(this).val().length < 6) {  
        alert("Пароль меньше 6 знаков!");  
    }  
});
```

resize([<функция>]) - устанавливает/запускает обработчик изменения размера элемента. Пример:

```
$(window).resize( function() {  
    alert("размер окна изменен");  
});
```

scroll([<функция>]) - устанавливает/запускает обработчик прокрутки области просмотра документа;

select([<функция>]) - устанавливает/запускает обработчик выделения текста в текстовом поле. Пример:

```
$(":input").select( function () {  
    alert("Текст был выделен");  
});
```

submit([<функция>]) - устанавливает/запускает обработчик отправки формы на сервер. Пример:

```
$("form").submit( function () {  
    if ($(":password").val().length < 6) {  
        alert("Пароль слишком короткий"); return false;  
    } return true;  
});
```

Содержание и стилизация элементов

text([<текст>]) - получает/устанавливает текстовое значение элемента.

html([<разметка>]) - получает/устанавливает HTML значение элемента.

append(<текст>) - добавляет текст в конец элемента. Пример:

```
$("div").append("текст в конец элемента");
```

prepend(<текст>) - добавляет текст в начало элемента.

after(<текст>) - добавляет текст после элемента.

before(<текст>) - добавляет текст до элемента.

remove([<класс>]) - удаляет выбранные элементы вместе с потомками. Если указан класс, то удаляются только элементы с указанным классом. Пример:

```
$("#p").remove(".test"); // удаление абзацев с классом test
```

empty() - удаляет содержание и потомки выбранного элемента.

attr(<атрибут>[,<значение>]) - получает/устанавливает значение атрибута выбранных элементов. Если элемент не имеет атрибута, возвращается значение *undefined*. Пример:

```
var title = $("#em").attr("title"); //получить значение атрибута title mega em  
$("#div").text(title); // установить содержимое mega div
```

removeAttr(<атрибут>) - удаляет заданный атрибут выбранных элементов.

addClass(<класс>) - добавляет новый класс к выбранным элементам. Пример:

```
$("#em").addClass("red"); // установить класс red для mega em
```

hasClass(<класс>) - проверяет наличие класса для выбранных элементов. Если указанный класс установлен, то возвращается *true*.

removeClass([<класс>]) - удаляет заданный класс для выбранных элементов. Если параметр *<класс>* не задан, то для выбранных элементов удаляются все классы.

toggleClass(<класс>[,<усл>]) - добавляет или удаляет (переключает) класс для выбранных элементов. Необязательный логический параметр *<усл>* может использоваться в качестве дополнительного условия: *true* - добавить класс, *false* - удалить класс. Пример:

```
$("#button").click(function() { // добавляет или удаляет класс red  
    $("#div").toggleClass("red");  
}); //
```

css(<свойство>[,<знач>]) - получить/установить CSS свойство для выбранных элементов. Если устанавливается несколько свойств, то параметры могут быть переданы в формате JSON. Пример:

```
$("#p").css( { "color": "blue", "background-color", "yellow" } );  
// установить синий цвет шрифта и желтый фон для абзацев
```

Визуальные эффекты

hide([<скорость>]) - скрыть элемент. Дополнительный параметр *<скорость>* может принимать значения *slow*, *fast* или число в миллисекундах. Пример:

```
$("#hide").click(function(){  
    $("#p").hide(); // скрыть все вложенные абзацы  
});
```

show([<скорость>]) - показать элемент. Метод обратный *hide*().

toggle([<скорость>]) - переключить видимость элемента. Аналогично применению функций *hide* и *show*.

fadeIn([<скорость>]) - проявление элемента. В отличие от метода *show*(), производится анимация свойства прозрачности (*opacity*).

fadeOut([<скорость>]) - исчезновение элемента. Производится анимация свойства прозрачности (*opacity*).

fadeTo([<скорость>],[<прозрачность>]) - проявление/исчезновение элемента. Производится анимация свойства прозрачности (*opacity*) до заданного значения.

fadeToggle([<скорость>]) - поочередное проявление или исчезновение элемента.

slideUp([<скорость>]) - плавное скрывание элемента по высоте.

slideDown([<скорость>]) - плавное разворачивание элемента по высоте.

slideToggle([<скорость>]) - плавное разворачивание/сворачивание элемента по высоте. С помощью этого метода можно организовать выпадающее меню:

```
$("#menu").click(function() { $("#list").slideToggle(500) });
```

animate(<свойства>,[<скорость>]) - применение пользовательской анимации. Список CSS свойств задается в виде javascript объекта. В отличие от метода *css*(), значения этих свойств могут быть указаны в виде констант *hide*, *show*, *toggle* или относительных единиц. Например:

```
$('#div').animate( // убрать прозрачность и уменьшить высоту на 50px  
    { opacity: "hide", height: "-=50" }, 5000  
);
```

Асинхронные запросы

load(<URL>) - загрузка содержимого с помощью AJAX запроса. Параметр <URL> содержит адрес запроса. Например:

```
$("#div").click(function(){  
    $(this).load('example.html'); // загрузить файл в текущий блок  
});
```

get(<URL>[,<obj>]) - выполнение GET запроса с помощью AJAX. Дополнительный параметр *obj* содержит параметры GET запроса:

```
$(this).get('example.php', { id: 1}); // параметр id=1
```

post(<URL>[,<obj>]) - выполнение POST запроса с помощью AJAX. Дополнительный параметр *obj* содержит параметры POST запроса.

ajax(<obj>) - выполнение запроса с помощью AJAX (универсальный метод). Параметр *obj* может содержать следующие поля:

- *async* — асинхронность запроса, по умолчанию true;

- *contentType* — тип возвращаемого содержимого, по умолчанию *application/x-www-form-urlencoded*;
- *data* — передаваемые данные (строка или объект);
- *dataType* — тип возвращаемых данных (*xml*, *html*, *script*, *json* или *text*);
- *type* — тип запроса GET или POST;
- *url* — URL запрашиваемой страницы;
- *username* — логин для подключения;
- *password* — пароль для подключения;
- *beforeSend* — функция-обработчик перед началом запроса;
- *error* — функция-обработчик ошибки;
- *success* — функция-обработчик успешного запроса;
- *complete* — функция-обработчик после завершения запроса;

Пример использования метода AJAX и вывод результата:

```
$(this).ajax( {
  url: 'example.json',
  dataType: 'json',
  success( function(data, status) {
    alert('Данные загружены в javascript объект data');
  });
});
```

getJSON(<URL>[,<obj>]) - выполнение запроса на получение объекта JSON с помощью AJAX. Дополнительный параметр *obj* содержит параметры запроса.

Сервисные функции

each(<obj>, <func(i[,val])>) - итератор, вызывающий заданную функцию для каждого элемента массива с индексом *i* или заданным свойством объекта.

Пример:

```
var obj = { 'one': 1, 'two': 2 };
$('div').each( obj, function(key, val) {
  alert(key + ': ' + val);
});
```

grep(<obj>, <func(elem[,i])>) - фильтрация элементов массива по заданным критериям. Пример:

```
var a = [1,2,5,9,16,25,8,9,6,3];
a = $.grep(a, function(elem, i) {
  return (elem == i*i); // a = [9,16,16]
});
```

extend(<res>,<obj1>[...<objN>]) - объединение двух и более объектов в один результирующий *<res>*. Пример:

```
var a = { 'one': 1 }; var b = { 'two': 2 };
var c = $.extend({}, a, b); // c = { 'one': 1, 'two': 2 }
```

map(<obj>, <func(elem[,i])>) -замена каждого элемента массива в соответствии с заданной функцией. Пример:

```
var a = ['a', 'b', 'c'];  
a = $.map(a, function(elem) {  
    return (elem.toUpperCase()); // a = ['A', 'B', 'C']  
});
```

parseHTML(str) — преобразование HTML строки в Javascript объект:

```
html = $.parseHTML('Hello, <br>world!');
```

parseJSON(str) — преобразование строки JSON в Javascript объект:

```
json = $.parseJSON('{ id: 1, name: test }');
```

parseXML(str) — преобразование строки XML в Javascript объект.

Библиотека jQuery в настоящее время активно развивается, в ней появляются новые функции и возможности. Эта библиотека лежит в основе многих проектов и более мощных Javascript фреймворков, таких как AngularJS, Bootstrap и др.

4 Сервер приложений

4.1 Установка сервера приложений

Обычно веб-приложение имеет трехзвенную архитектуру, поэтому для его работы необходим сервер приложений. Установка сервера приложений включает установку веб-сервера, сервера баз данных и интерпретатора серверных сценариев. Из свободного программного обеспечения в настоящее время наиболее популярными являются следующие программные продукты:

- веб-сервер Apache 2.4;
- сервер баз данных MariaDB 10.3 (совместимый с MySQL 5.7);
- интерпретатор языка программирования PHP 7.3.

Конкретные технические характеристики сервера приложений определяются функциональными требованиями, предъявляемыми к веб-приложению: объемом трафика между клиентской и серверной частью (запрос — ответ), производительностью обработки одного запроса пользователя, количеством одновременных запросов к веб-приложению.

Необходимо заметить, что для запуска разрабатываемого веб-приложения в отладочном режиме обычно достаточно домашнего компьютера средней производительности. В разное время сервер приложений тестировался автором на следующих платформах: Windows XP, Windows 7/8 (x32 и x64), Windows 10 x64, FreeBSD, Linux (CentOS, Ubuntu, OpenSUSE, Mint). Для установки данной сборки в учебных целях рекомендуется использовать компьютер с характеристиками не ниже: Intel Pentium Duo Core 2GHz/2 Gb DDR3/100 Gb HDD и операционной системой Windows 7 SP1 x64.

Интегрированные сборки сервера приложений типа Denwer, WAMP, XAMPP, OpenServer и др. использовать в учебных целях не рекомендуются: во-первых, они не всегда содержат нужные версии прикладных библиотек и модулей, а во-вторых, для лучшего понимания принципа работы каждый компонент сервера приложений необходимо уметь устанавливать и настраивать отдельно.

Программное обеспечение сервера приложений рекомендуется настраивать в следующей последовательности:

1. Установка веб-сервера Apache.
2. Установка сервера баз данных MariaDB.
3. Установка интерпретатора языка программирования PHP.
4. Настройка и тестирование сервера приложений.

Установка веб-сервера Apache

Согласно отчету компании Netcraft на май 2019 года, сервер Apache является одним из самых распространенных веб-серверов в мире по размещению сайтов. В настоящее время на нем размещены около 385 млн. активных сайтов (29% от общего числа), 72 млн. из них имеют доменное имя (30% от общего числа). Другим популярным веб-сервером является nginx — на нем размещены более 387 млн. активных сайтов (29% от общего числа), 62 млн. из них имеют доменное имя (26% от общего числа).

Официальным сайтом разработчиков Apache является сайт <http://httpd.apache.org>. Для получения бинарной сборки Apache под Windows рекомендуется использовать сайт <https://www.apachelounge.com/download>.

Если у вас 64-битная версия Windows, то вы можете выбрать как 64-битную, так и 32-битную версию сборки. Главное правило - все компоненты должны быть одной версии. Если у вас 32-битная версия Windows, то все компоненты должны быть 32-битными. Обратите внимание, что сборки Apache 2.4 VC11/VC15 не работают под Windows XP. Рекомендуемые платформы - Windows 7 SP1 x64 или Windows 10. Здесь и далее мы покажем установку веб-сервера на примере сборки Apache 2.4 VC15.

После скачивания подходящего архива (например, httpd-2.4.39-win64-VC15.zip), распакуйте содержимое папки архива Apache24 в отдельный каталог, например, C:\Apache24. В этом случае корнем веб-сервера (*ServerRoot*) будет являться каталог C:\Apache24, а корневой папкой веб-сервера (*DocumentRoot*) будет являться каталог C:\Apache24\htdocs.

Для версии Apache 2.4 VC15 необходимо скачать и установить компонент Microsoft Visual C++ 2017 Redistributable x64 (или более поздний), именуемый как VC15. Если этот компонент в вашей системе не установлен, то его можно скачать со страницы <http://www.apachelounge.com/download> по ссылке *vc_redist_x64*, или по прямой ссылке https://aka.ms/vs/15/release/VC_redist.x64.exe.

Для установки веб-сервера Apache в качестве службы Windows необходимо зайти в консоль под учетной записью Администратора (клавиши *Win+X*) и

выполнить команду инсталляции:

```
>C:\Apache24\bin\httpd.exe -k install
```

Если поступит запрос от файервола в отношении Apache, то нажмите «Разрешить». По окончании установки система также может потребовать отключить контроль учетных записей UAC. Для запуска веб-сервера Apache из консоли необходимо выполнить команду:

```
>C:\Apache24\bin\httpd.exe -k start
```


После запуска веб-сервера можно проверить его работоспособность. Для этого в адресной строке браузера нужно набрать адрес <http://localhost>. В качестве результата вы должны получить HTML страницу с сообщением:

It works!

Это означает, что веб-сервер Apache установлен успешно и работает.

Для остановки веб-сервера Apache из консоли можно выполнить команду:

```
>C:\Apache24\bin\httpd.exe -k stop
```

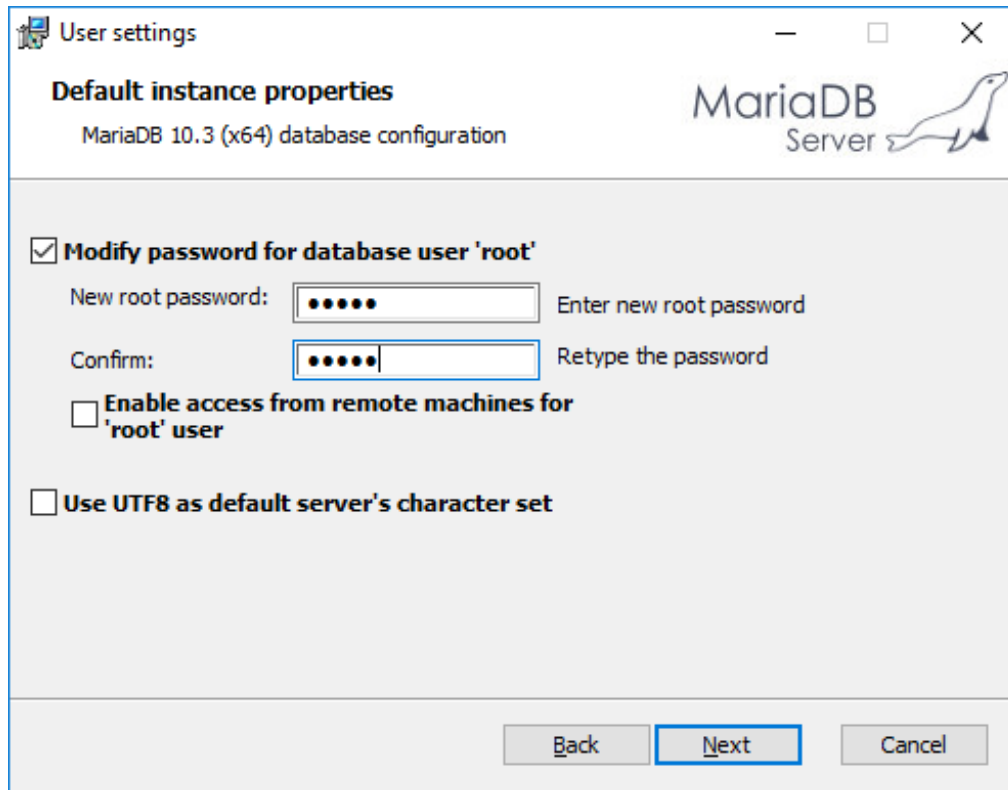
Управление веб-сервером Apache также можно осуществлять с помощью оснастки служб *services.msc*, команд *net* или *sc* (служба Apache2.4). Но намного проще это делать с помощью программы администрирования *ApacheMonitor.exe*, входящей в сборку Apache. По умолчанию эта программа находится в папке C:\Apache24\bin, ее можно добавить в пункт Автозагрузки. Значок запущенной программы, отображающей состояние веб-сервера, будет виден в панели задач 

Установка сервера баз данных MariaDB

Самой популярной в мире СУБД для разработки веб-приложений является MySQL. Это также одна из самых простых и нетребовательных к ресурсам СУБД, отлично зарекомендовавших себя в небольших проектах. Хотя в настоящее время она все еще распространяется на базе открытой лицензии GPL, после приобретения проекта в 2010 году компанией Oracle разработчики выступили с инициативой обеспечения свободного статуса СУБД. Так появилось самостоятельное ответвление MariaDB, совместимое с MySQL (до версии 5.5, включительно). Версия 10 MariaDB стала развиваться независимо от MySQL и в настоящее время включена в множество серверных дистрибутивов Linux, в том числе, RHEL/CentOS 7, чаще всего используемой коммерческими провайдерами. Поэтому рекомендуется устанавливать версию MariaDB.

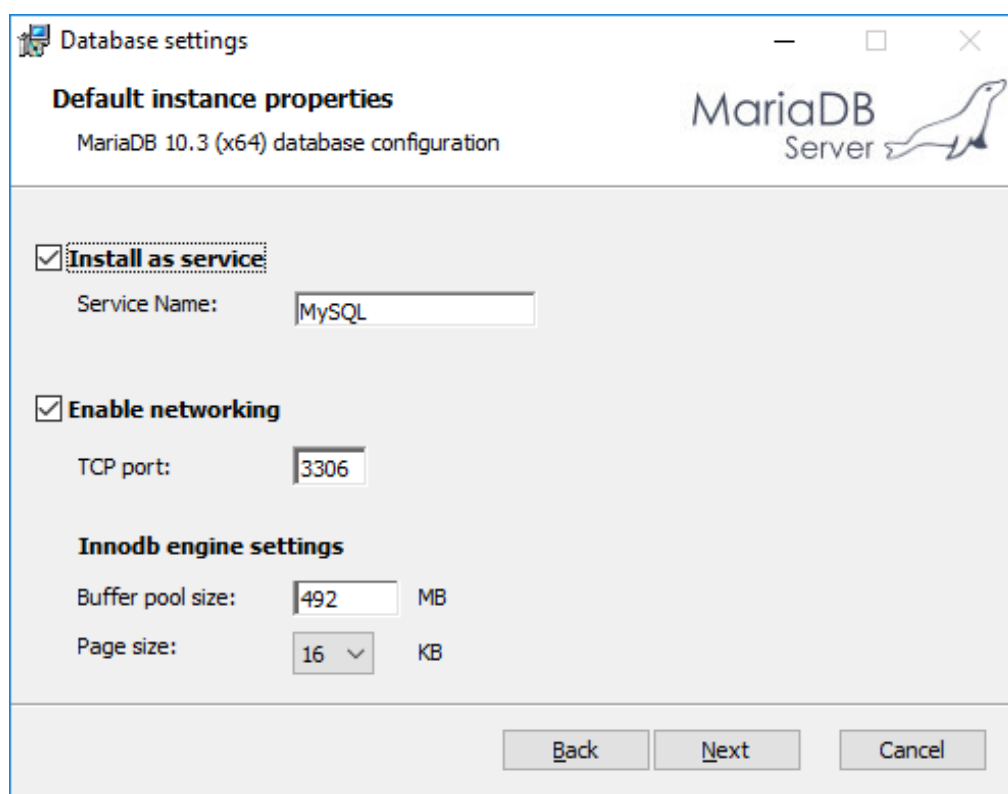
Официальным сайтом разработчиков СУБД MariaDB является сайт <https://mariadb.com>, откуда можно скачать и установить последнюю версию. Разрядность СУБД должна соответствовать разрядности платформы, например, *mariadb-10.3.15-winx64.msi*. После запуска инсталлятора необходимо подтвердить лицензию и выбрать путь установки СУБД (по-умолчанию, каталог C:\Program Files\MariaDB 10.3), затем программа предложит дважды ввести пароль администратора СУБД (*Modify password for database user root*) и

установить кодировку таблиц по умолчанию UTF8:



The screenshot shows the 'User settings' window for MariaDB 10.3 (x64) database configuration. The window has a title bar with standard Windows controls. The main content area is titled 'Default instance properties' and includes the MariaDB logo. There are three checkboxes: 'Modify password for database user 'root'' (checked), 'Enable access from remote machines for 'root' user' (unchecked), and 'Use UTF8 as default server's character set' (unchecked). The 'Modify password' section has two password input fields labeled 'New root password:' and 'Confirm:', both containing masked characters. At the bottom, there are three buttons: 'Back', 'Next' (highlighted with a blue border), and 'Cancel'.

В следующем окне выбираются опции для запуска сервера:



The screenshot shows the 'Database settings' window for MariaDB 10.3 (x64) database configuration. The window has a title bar with standard Windows controls. The main content area is titled 'Default instance properties' and includes the MariaDB logo. There are two checkboxes: 'Install as service' (checked) and 'Enable networking' (checked). The 'Install as service' section has a 'Service Name:' field with 'MySQL' entered. The 'Enable networking' section has a 'TCP port:' field with '3306' entered. Below these, there is a section titled 'InnoDB engine settings' with a 'Buffer pool size:' field set to '492' MB and a 'Page size:' dropdown menu set to '16' KB. At the bottom, there are three buttons: 'Back', 'Next' (highlighted with a blue border), and 'Cancel'.

Важной опцией является установка сервера СУБД как сервиса (Install as service). В этом случае возможно управление запуском СУБД с помощью оснастки служб *services.msc*, команд *net* или *sc* (служба MySQL).

По окончании установки система может потребовать отключить контроль

учетных записей UAC и создаст раздел MariaDB в главном меню «Пуск».

Для облегчения запуска клиента СУБД из командной строки рекомендуется в переменную окружения PATH добавить путь установки *C:\Program Files\MariaDB 10.3\bin*. Для этого нужно найти в поиске пункт «Система», далее последовательно выбрать ссылку в левом меню «Дополнительные параметры системы», вкладку «Дополнительно», кнопку «Переменные среды», выбрать пункт меню «Системные переменные - Path» и нажать кнопку «Изменить». В открывшемся окне нажать кнопку «Создать», добавить путь к каталогу и в завершении нажать «ОК». Для активации переменной PATH консольное окно администратора необходимо перезапустить. После этого полный путь к команде `mysql` в консоли администратора можно не указывать — система найдет его автоматически.

После установки СУБД можно зайти в административную консоль (*Win + X*) и запустить клиент MySQL с помощью команды:

```
>mysql -u root -p
```

На запрос программы необходимо ввести пароль администратора и, если он корректен, вы получите полный доступ к командам СУБД из консоли:

MariaDB>

Используя синтаксис SQL команд *create database*, *create table* и др., вы можете создавать базы данных, таблицы, назначать права пользователям и выполнять любые запросы к СУБД как из консоли, так и из других программ, в соответствии со спецификацией языка MariaDB 10.3.

Для получения помощи по системе команд СУБД в консоли клиента СУБД можно выполнить команду *help* с параметром или без:

MariaDB>help select

Для выхода из консоли клиента СУБД, необходимо вызвать команду *quit*:

MariaDB>quit

Для работы с данными СУБД вместо консоли удобнее использовать специальные приложения, такие как phpMyAdmin (для доступа к СУБД через веб-интерфейс) или HeidiSQL (для локального доступа к СУБД). Ярлык программы HeidiSQL после установки автоматически размещается на рабочем столе.



Установка интерпретатора языка PHP

Самым популярным языком программирования веб-приложений в мире является язык PHP. В 1994 году датский программист Рasmus Лерддорф создал набор скриптов на языке *Perl* для учета посетителей онлайн резюме, обрабатывающий шаблоны HTML документов, и назвал его *Personal Home Page* (персональная домашняя страница или PHP). В дальнейшем сам язык претерпел множество изменений, код интерпретатора неоднократно

переписывался на языке C++, были добавлены функции работы с базами данных, осуществлена поддержка внешних модулей, в версии PHP 5 язык стал полностью объектным. В 2015 году вышла новая версия интерпретатора языка PHP 7. Новая версия позволяет увеличить производительность серверных скриптов до 2 раз и сэкономить до 50% памяти по сравнению с традиционным PHP 5. Это достигается за счет усовершенствованного механизма компиляции и кэширования (новая версия компилятора Zend Engine 3), поддержкой многопоточности и другими усовершенствованиями, что позволяет написанным на нем программам по производительности конкурировать с Java приложениями. Достоинством языка PHP также является простота написания программ, низкий порог вхождения, невысокая стоимость поддержки готовых проектов.

Интерпретатор языка PHP можно скачать и установить с официального сайта по ссылке <https://windows.php.net/download>. Разрядность интерпретатора PHP в нашем случае должна совпадать с разрядностью платформы, а также с версией установленной библиотеки *Microsoft Visual C++ Redistributable*. Официальный релиз PHP имеет два вида сборок: *Thread Safe* (с поддержкой многопоточности) и *Non Thread Safe* (без такой поддержки). Потокбезопасная сборка используется для организации многопоточных приложений с использованием модуля `mod_php` веб-сервера Apache. Другой вариант сборки используется для разработки однопоточных приложений с использованием интерфейса FastCGI и сервера Microsoft IIS.

Таким образом, для использования `mod_php` совместно с Apache скачиваем потокбезопасную сборку (например, *php-7.3.6-ts-Win32-VC15-x64.zip*). Архив следует распаковать в корневой каталог диска, например, в *C:\PHP*, после чего интерпретатор готов к работе. Для запуска библиотек PHP рекомендуется в переменную окружения PATH добавить путь установки *C:\PHP* (см. выше). Для функционирования интерпретатора PHP в составе сервера приложений после установки PHP необходимо произвести его настройку.

4.2 Настройка сервера приложений

Настройка веб-сервера Apache

Для настройки веб-сервера Apache используется основной конфигурационный файл *httpd.conf*, который размещается в каталоге *C:\Apache24\conf*. Конфигурационный файл имеет текстовый формат и редактируется с помощью блокнота или программы Notepad++. Конфигурационный файл содержит набор команд (директив), некоторые из которых имеют XML формат. Каждая директива начинается с новой строки и может содержать один или несколько параметров. Для деактивации команды ее можно закомментировать, поставив перед ней символ # (решетка). Для каждой директивы имеется описание в виде комментария на английском языке в конфигурационном файле, а также страницы руководства на сайте <http://httpd.apache.org/docs/2.4> и в локальном каталоге *C:\Apache24\manual*.

Установка домена и корневой папки сервера

Для установки домена вашего веб-приложения необходимо задать значение для директивы *ServerName*, например:

```
ServerName www.example.ru
```

По умолчанию, строка с этой директивой закомментирована, а сам домен ассоциирован с *localhost*. Однако если вы хотите, чтобы ваше веб-приложение загружалось в браузере по адресу *http://www.example.ru*, как это планируется на рабочем сервере, необходимо ассоциировать IP адрес вашего компьютера с указанным доменом. Для учебного примера можно добавить в текстовый файл *C:\windows\system32\drivers\etc\hosts* значение домена в строку *localhost* через пробел:

```
127.0.0.1 localhost www.example.ru
```

Для редактирования текстовый файл *hosts* нужно открывать в блокноте с правами администратора.

По умолчанию, корневая папка сервера установлена в *C:\Apache24\htdocs*, что не всегда удобно для разработчика. Для установки корневой папки сервера в другой каталог (например, *d:\www*) нужно изменить значение директивы *DocumentRoot* "*\${SRVROOT}/htdocs*" на строку, содержащую путь:

```
DocumentRoot "d:/www"
```

Обратите внимание на использование прямого слэша в файловых путях конфигурации.

Установка прав доступа к содержимому каталогов

В секции *Directory* устанавливаются права доступа к содержимому для отдельного каталога веб-сервера, например:

```
<Directory "${SRVROOT}/htdocs">  
    Options Indexes FollowSymLinks  
    AllowOverride None  
    Require all granted  
</Directory>
```

Директива *AllowOverride*, указанная в данной секции, запрещает использование локальных конфигурационных файлов *.htaccess*. Директива *Options* разрешает индексирование и доступ к содержимому данного каталога. Директива *Require all granted* предоставляет эти права для всех пользователей.

Так как к нашему каталогу *d:\www* необходимо предоставить полный доступ, а каталог *C:\Apache24\htdocs* использоваться не будет, секцию *Directory* можно привести к следующему виду:

```
<Directory "d:/www">  
    Options All  
    AllowOverride All  
    Require all granted
```

</Directory>

Нужно отметить, что каталог данных, указанный в параметре директивы *Directory*, должен совпадать или быть дочерним для корневой папки, указанной в директиве *DocumentRoot*. Исключения составляют каталоги, доступ к которым осуществляется с помощью псевдонимов (директив *Alias* или *ScriptAlias*).

Для всех остальных каталогов сервера используется доступ по-умолчанию, определяемый секцией *<Directory />*.

Загрузка и подключение внешних модулей

Модули веб-сервера Apache размещаются в каталоге *C:\Apache24\modules* в специальном объектном формате DSO. Исключением является модуль PHP, который загружается с помощью библиотеки *php7apache2_4.dll*. Для загрузки модулей во время старта веб-сервера используется директива *LoadModule*:

LoadModule <имя_модуля> <имя_файла>

Например, для подключения модуля *php7* используется директива:

LoadModule php7_module "c:/php/php7apache2_4.dll"

Также для инициализации PHP модуля необходимо добавить директиву для указания каталога местоположения конфигурационного файла *php.ini*:

PHPIniDir "C:/PHP"

Каждый модуль расширяет функциональность веб-сервера и имеет свой набор встроенных команд. Во избежание ошибки конфигурирования веб-сервера при отмене загрузки того или иного модуля, встроенные директивы модуля заключают в условные секции *<IfModule>*, например:

<IfModule dir_module>

DirectoryIndex index.html

</ IfModule>

Это означает, что директива *DirectoryIndex* будет сопоставлять индексный файл как *index.html* только в случае предварительной загрузки модуля *dir_module*. В противном случае веб-сервер не сможет корректно распознать эту команду.

Для правильной работы PHP веб-приложений необходимо в конец директивы *DirectoryIndex* добавить через пробел имя индексного файла *index.php*:

DirectoryIndex index.html index.php

Индексирование позволяет обращаться к каталогу без указания в адресе URL имени файла, подразумевая при этом индексный файл.

Некоторые модули отключены по-умолчанию. Для их подключения необходимо раскомментировать соответствующую директиву *LoadModule*, например:

LoadModule rewrite_module modules/mod_rewrite.so

В этом случае при загрузке Apache подключается модуль *rewrite*, который

позволяет управлять содержимым URL «на лету», осуществляя синтаксический разбор строки запроса с помощью регулярных выражений.

Запуск файлов по расширению

Для запуска файлов в браузере по расширению используется настройка модуля MIME (*Multipurpose Internet Mail Extensions*). Например, для запуска PHP скриптов в секцию `<IfModule mime_module>` необходимо добавить директиву *AddHandler* с именем типа *application/x-httpd-php* и сопоставляемым ему расширением файла *.php*:

```
<IfModule mime_module>  
    AddHandler application/x-httpd-php .php  
</IfModule>
```

Перезапуск сервера и обработка ошибок

После редактирования необходимо сохранить измененный файл *httpd.conf* на диск, затем перезапустить веб-сервер Apache с помощью программы *ApacheMonitor* или команды административной консоли:

```
>C:\Apache24\bin\httpd.exe -k restart
```

Если конфигурационный файл не содержит ошибок, то веб-сервер Apache будет запущен успешно. Если файл *httpd.conf* содержит синтаксические ошибки, то в консоль будет выведено соответствующее сообщение с наименованием ошибки и номером строки, где она была обнаружена, например:

AH00526: Syntax error on line 269 of C:/Apache24/conf/httpd.conf:

Для вывода ошибок веб-приложений используется текстовый файл указанный в директиве *ErrorLog*. По-умолчанию используется журнал ошибок *error.log*, который размещается в каталоге *C:\Apache24\logs*. Формат этого файла и уровень вывода задается директивой *LogLevel*.

Для задания журнала посещений веб-приложения используется текстовый файл, указанный в директиве *CustomLog*. По-умолчанию используется файл *access.log*, размещенный в каталоге *C:\Apache24\logs*. Формат этого файла задается директивой *LogFormat*. Как правило в журнале посещений указывается входящий IP адрес посетителя, запрашиваемый URL, время обращения, объем загруженного ресурса, другие параметры запроса.

Дополнительные конфигурации и виртуальные сервера

Для удобства директивы конфигурации по своему назначению группируются в дополнительные конфигурационные файлы. По-умолчанию, эти файлы размещаются в каталоге *C:\Apache24\conf\extra*. Для подключения дополнительных конфигураций в основном файле *httpd.conf* используется директива *Include* `<имя_файла>`. Путь к дополнительным конфигурациям указывается относительно корня веб-сервера, например:

```
Include conf/extra/httpd-vhosts.conf
```

В результате будет подключен файл, описывающий настройки виртуальных

серверов. Виртуальные сервера используются в случае, когда на одном компьютере размещается сразу несколько веб-приложений, имеющих различные домены (удобный инструмент для разработчиков). В этом случае конфигурация каждого веб-приложения описывается в файле *httpd-vhosts.conf* в отдельных секциях директивы *<VirtualHost>*, например:

```
<VirtualHost 127.0.0.1>
    ServerName www.example1.ru
    DocumentRoot "D:\www\example1"
    ErrorLog "logs/example1-error.log"
    CustomLog "logs/example1-access.log"
</VirtualHost>
```

Директивы *ServerName* и *DocumentRoot* в каждой секции устанавливают имя виртуального домена и корневой каталог документов для него.

В конфигурации виртуального сервера указаны два отдельных лог файла: *example1-error.log* и *example1-access.log*, для журнала ошибок и журнала посещения, соответственно.

Для успешной работы такого веб-приложения необходимо ассоциировать виртуальный домен *example1.ru* с IP адресом 127.0.0.1 (*localhost*), прописав его в файле *C:\windows\system32\drivers\etc\hosts* так же, как это было сделано для корневого домена *www.example.ru*.

Настройка сервера СУБД MariaDB

После установки сервера MariaDB первым делом необходимо настроить подключение и создать новую базу данных для использования ее РНР приложениями. Для этого необходимо зайти в административную консоль, запустить клиент СУБД и ввести пароль администратора:

```
>mysql -u root -p
```

Далее в консоли клиента СУБД необходимо создать новую базу данных, выполнив команду *create database*, например:

```
MariaDB>create database dbname;
```

После этого необходимо создать нового пользователя и задать ему права на доступ к базе данных, например:

```
MariaDB>create user 'wwwuser'@'localhost' identified by '13a24b57';
```

```
MariaDB>grant all privileges on dbname.* to 'wwwuser'@'localhost';
```

В нашем случае создается пользователь *wwwuser* для полного доступа к базе данных *dbname* из домена *localhost* (т.е. из нашего веб-приложения). Пароль пользователя указан как *13a24b57*. Этот логин и пароль также нужно будет прописать явно в веб-приложениях.

В завершении необходимо применить привелегии вновь созданного пользователя:

MariaDB>flush privileges;

Далее осталось создать необходимые таблицы и загрузить в них данные для веб-приложения. Это можно сделать с помощью команд *create table* и *insert*, как в консоли *mysql*, так и с помощью клиентов *HeidiSQL*, *phpMyAdmin* или других аналогичных.

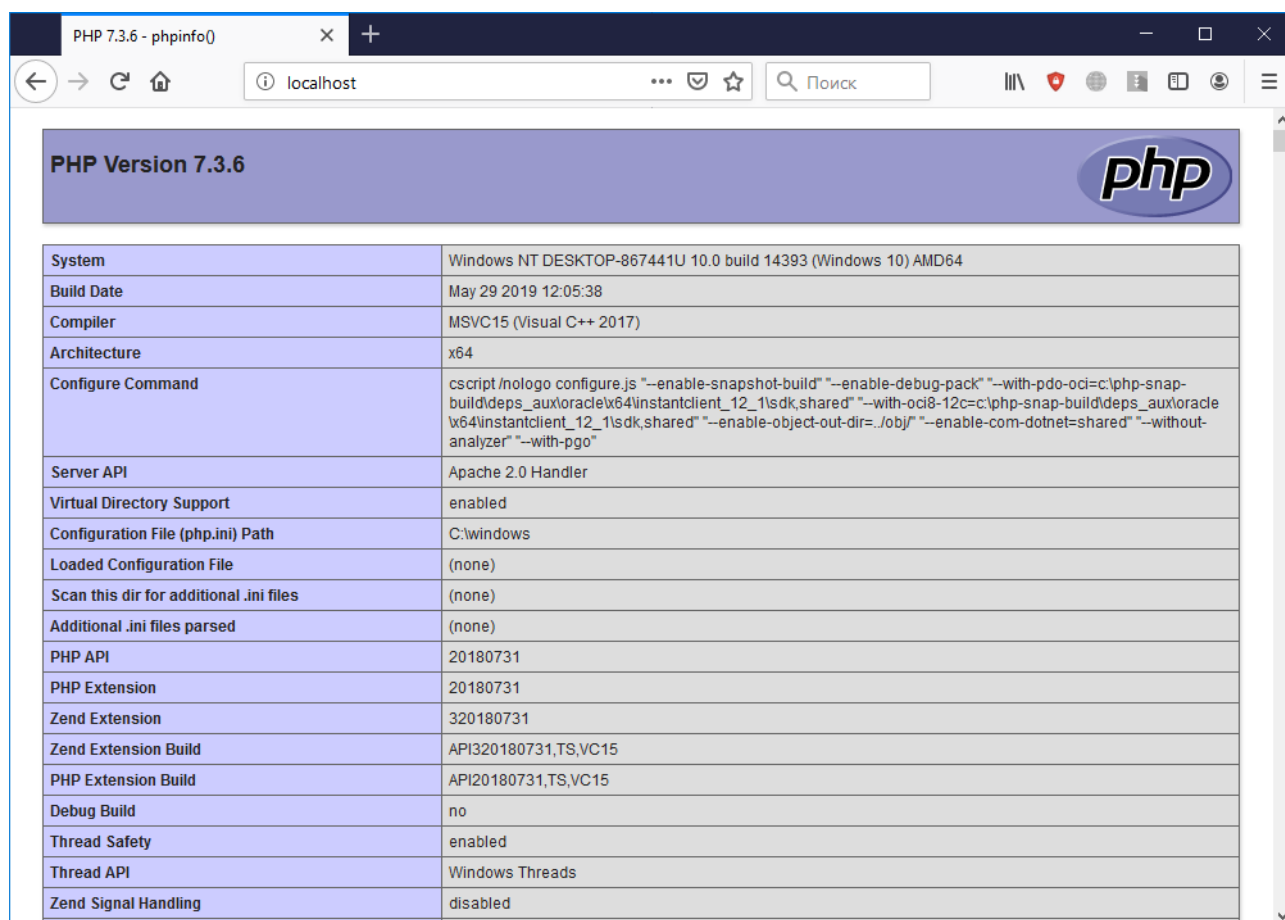
Для проектирования веб-приложений, организации работы СУБД *MariaDB*, необходима настройка системных параметров сервера, находящихся в файле .

Настройка интерпретатора PHP

После установки и настройки веб-сервера *Apache* можно проверить работоспособность модуля *PHP*. Для этого в каталоге *D:\www* с помощью текстового редактора необходимо создать файл *index.php* со следующим содержанием:

```
<?php
    phpinfo();
```

Затем в браузере необходимо открыть ссылку по адресу <http://localhost>. В результате откроется таблица, содержащая информацию об установленной версии *PHP*.



PHP Version 7.3.6	
System	Windows NT DESKTOP-867441U 10.0 build 14393 (Windows 10) AMD64
Build Date	May 29 2019 12:05:38
Compiler	MSVC15 (Visual C++ 2017)
Architecture	x64
Configure Command	cmdscript /nologo configure.js "--enable-snapshot-build"--enable-debug-pack"--with-pdo-oci=c:\php-snap-build\deps_aux\oracle\x64\instantclient_12_1\sdk,shared"--with-oci8-12c=c:\php-snap-build\deps_aux\oracle\x64\instantclient_12_1\sdk,shared"--enable-object-out-dir=.\obj"--enable-com-dotnet=shared"--without-analyzer"--with-pgo
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\windows
Loaded Configuration File	(none)
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20180731
PHP Extension	20180731
Zend Extension	320180731
Zend Extension Build	API320180731,TS,VC15
PHP Extension Build	API20180731,TS,VC15
Debug Build	no
Thread Safety	enabled
Thread API	Windows Threads
Zend Signal Handling	disabled

Для проектирования веб-приложений, организации совместной работы *Apache*, *PHP* и СУБД *MariaDB*, необходимо подключение клиентских библиотек *PHP* и настройка параметров конфигурационного файла *php.ini*.

Для этого копируем конфигурационный файл по умолчанию *php.ini-development* в *php.ini* в каталог установки *C:\PHP* в административной консоли:

```
>copy C:\PHP\php.ini-development C:\PHP\php.ini
```

Далее необходимо открыть файл *php.ini* в текстовом редакторе и установить нужные вам параметры:

```
extension_dir = C:\PHP\ext
```

- устанавливает каталог расширений по-умолчанию,

```
memory_limit = 128M
```

- определяет количество памяти, выделяемое для PHP скрипта,

```
post_max_size = 50M
```

- устанавливает максимальный объем данных, передаваемых методом POST,

```
upload_max_filesize = 50M
```

- устанавливает максимальный размер загружаемого на сервер файла,

```
max_execution_time = 30
```

- устанавливает максимальное время выполнения одного скрипта, и т.д.

Также необходимо раскомментировать строки, содержащие подключение системных библиотек, например:

```
extension=pdo_mysql
```

4.3 Администрирование сервера приложений

5 Серверное программирование на PHP

5.1 Элементы языка PHP

5.2 Обработка входных параметров

5.3 Объектно-ориентированное программирование

6 Разработка веб-приложений

6.1 Паттерн проектирования MVC

6.2 Маршрутизация URL

6.3 Отладка и профилирование

6.4 Механизмы аутентификации и авторизации

6.5 Безопасность веб-приложений