

Министерство образования и науки Российской Федерации
Сибирский государственный аэрокосмический университет
имени академика М. Ф. Решетнева

А. Г. ЗОТИН
М. В. ДАМОВ

РАЗРАБОТКА ИНФОРМАЦИОННЫХ СИСТЕМ В ВЕБ-СРЕДЕ

*Утверждено редакционно-издательским советом университета
в качестве учебного пособия для студентов магистратуры, обучающихся
по направлению подготовки 09.04.02 «Информационные системы и технологии
очной формы обучения*

Красноярск 2018

УДК 004.738.5, 004.43, 004.42 (075.8)
ББК 32.973.4Я.7
3-88

Рецензенты:

кандидат технических наук, доцент Ю.Ю. Якунин
(Сибирский федеральный университет);
кандидат педагогических наук, доцент Ю.Б. Козлова
(Сибирский государственный университет науки и технологий
имени академика М.Ф. Решетнева)

Зотин, А. Г.

3-88 Разработка информационных систем в веб-среде: учебное пособие / А. Г. Зотин, М. В. Дамов ; Сибирский государственный университет. – Красноярск, 2018. – 114 с.

Приведено описание технологий, которые могут использоваться для разработки корпоративных веб-приложений с примерами их использования. Рассмотрены как клиентские (HTML, CSS, JavaScript), так и серверные (PHP, MySQL) технологии. Приведен пример создания сайта на базе системы управления содержимым сайта «1С-Битрикс» с использованием указанных технологий.

Пособие предназначено для студентов магистратуры, обучающихся по направлению 09.04.02 «Информационные системы и технологии» очной формы обучения, а также для всех желающих самостоятельно изучить разработку корпоративных веб-приложений.

УДК 004.738.5, 004.43, 004.42 (075.8)
ББК 32.973.4Я7

© Сибирский государственный университет науки
и технологий имени академика М. Ф. Решетнева, 2018
© Зотин А. Г., Дамов М. В., 2018

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ.....	5
Жизненный цикл информационных систем в web-среде	Ошибка! Закладка не определена.
Основы клиентских технологий (CSS, HTML)	8
HTML.....	8
Структура HTML страницы	8
Объектная модель документа	10
Отношения между узлами.....	11
CSS	12
Способы добавления стилей на страницу.....	12
Стилевые правила CSS	14
Виды селекторов	14
Псевдоклассы и псевдоэлементы	20
Основные свойства	23
Верстка на основе блоков с использованием CSS.....	25
Блочная модель CSS	27
Описание свойств CSS для блочной вёрстки	28
Описание верстки на примере	30
Организация меню	36
Простое меню	36
Горизонтальное меню	37
Вертикальное меню.....	39
Меню содержащее элементы подменю	40
JS	42
Способы подключения и использования JS	43
Обращение к элементам DOM.....	45
Работа с cookie на примере.....	46
Основы серверных технологий (PHP, CGI), модель MVC.	48
Язык PHP	48
PHP: Подключение файлов	48
PHP: Работа POST и GET	50
Передача переменных при помощи метода GET.....	50
Передача переменных в PHP при помощи метода POST	51
Сравнение методов GET и POST.....	52
PHP: Работа с базами данных	52
MySQL.....	52
MySQLi.....	54
PDO	57
Сравнение(особенности) MySQLi и PDO.....	60
MVC	61
Основы форматов передачи данных (XML, CML, JSON)	63

XML и его производные	63
Структура файлов формата XML	63
Пролог XML	63
Тело документа XML	65
Базовые правила XML	67
JSON.....	67
Коммерческие CMS.....	70
Структура CMS Битрикс	70
Реализация прав доступа к информации	74
Создание сайта.....	76
Структура сайта.....	77
Создание шаблона страницы	81
Шаблон дизайна	85
Язык сайта.....	87
Административная панель	88
Создание шаблона дизайна сайта	88
Создание страниц и разделов сайта	95
Управление динамической информацией	101
Библиографический список.....	114

ПРЕДИСЛОВИЕ

Развитие сети Интернет и облачных технологий привело к появлению новых классов приложений и новых принципов распространения общеизвестных приложений. Информационные системы на базе веб-технологий позволяют обрабатывать большие объемы информации и пользоваться ими широкому кругу потребителей.

Существует несколько подходов к созданию подобных приложений: они могут быть созданы с нуля, а также на базе свободных или коммерческих систем управления содержимым сайта.

При создании с нуля можно наиболее гибко управлять будущим функционалом разрабатываемой системы, однако необходимо также полностью самостоятельно реализовывать стандартные функции системы, например модуль управления пользователями, обеспечивать безопасность системы и поддержку пользователей, а также многое другое.

Если информационная система создается на базе свободного программного обеспечения (ПО), то многие стандартные функции, как правило, уже реализованы. В то же время для базового ПО имеется большое количество разработчиков готовых участвовать в разработке новой информационной системы и, таким образом, обеспечить будущее функционирование системы проще. Однако при этом подходе функционал системы придется разрабатывать с использованием множества готовых модулей различных авторов, что может затруднить реализацию и поддержку создаваемого ПО. В то же время необходимо соответствовать стандартам разработки свободного ПО, в том числе лицензионным. Это может затруднить коммерческое использование будущего проекта.

Использование коммерческого фрейворка при соблюдении лицензионной политики исходного разработчика и первоначальных небольших затратах часто более оправдано.

В данном пособии в качестве базы для разработки информационной системы в среде Интернет рассмотрена коммерческая система управления содержимым сайта (CMS – Content Management System) «1С: Битрикс». Эта система обладает обширным сообществом разработчиков, хорошей поддержкой производителя, технической документацией. При наличии лицензии производитель обеспечивает доступ к исходным кодам.

В первой главе пособия даны некоторые сведения о технологиях и средствах, применяемых при разработке сайтов и веб-приложений – рассмотрены как клиентские технологии (HTML5, CSS3, JavaScript), так и серверные (база данных MySQL и язык программирования PHP).

В второй главе пособия рассмотрены приемы проектирования с использованием CMS (создание шаблона из верстки HTML-документа, управление страницами и разделами сайта или web-приложения, а также управление информационными блоками, включая их вывод на страницах сайта, применительно к CMS «Битрикс»).

В конце пособия приведены контрольные вопросы, библиографический список с литературой, рекомендуемой для самостоятельного изучения.

Пособие предназначено для студентов магистратуры по направлению подготовки 09.04.02 «Информационные системы и технологии», а также может быть использовано бакалаврами, обучающимися по направлениям подготовки 09.03.02 «Информационные системы и технологии», 09.03.03 «Прикладная информатика», 09.03.04 «Программная инженерия» при изучении ими дисциплин, посвященных web-технологиям и разработке корпоративных веб-приложений.

Материал, представленный в пособии, может использоваться в качестве основы при выполнении курсовых работ и выпускной квалификационной работы.

Введение

Проектирование информационных систем и сайтов в веб-среде по аналогии с проектированием настольных систем также имеет определенный жизненный цикл.

Работа начинается с заполнения анкетирования заказчика (заполнения брифа проекта), где он должен ответить на заранее подготовленные вопросы о функционале, интерфейсе информационной системы, особенностях предметной области, дополнительных пожеланиях.

На основе брифа составляется техническое задание (ТЗ), где в соответствии с государственными, отраслевыми или корпоративными стандартами отражаются пожелания заказчика.

На основе ТЗ формируется прототип интерфейса информационной системы, который представляет собой страницы с обозначенными на них блоками информации в виде прямоугольниками с подписями

На основе ТЗ и прототипа интерфейса далее создаются макеты в виде изображений страниц интерфейса с использованием фирменного стиля и графических элементов оформления.

Далее на основе ТЗ и макета создается HTML-верстка интерфейса системы. Если есть необходимость, то реализация интерфейса выполняется с использованием JavaScript и мультимедийных объектов.

Создание шаблона и интеграция верстки с использованием CMS являются последним этапом разработки.

При дальнейшем тестировании выполняется первоначальное наполнение информацией и сдача проекта заказчику.

Конечным этапом является этап эксплуатации и сопровождения.

Основы клиентских технологий (CSS, HTML)

HTML

HTML (от англ. HyperText Markup Language – «язык гипертекстовой разметки») – стандартный язык разметки документов во Всемирной паутине. Большинство веб-страниц содержат описание разметки на языке HTML (или XHTML). Язык HTML интерпретируется браузерами и отображается в виде документа в удобной для человека форме. Любой документ на языке HTML представляет собой набор элементов, у которых начало и конец обозначается специальными пометками – тегами.

Элементы документа могут быть пустыми, то есть не содержащими никакого текста и других данных. В таких случаях обычно не указывается закрывающий тег. Примером для такого случая является тег перевода строки – `
..` Кроме того, элементы могут иметь атрибуты, определяющие какие-либо их свойства (например, размер шрифта для элемента `font`). Атрибуты указываются в открывающем теге.

СТРУКТУРА HTML СТРАНИЦЫ

Каждый HTML-документ, отвечающий спецификации HTML какой-либо версии, обязан начинаться со строки декларации версии HTML `<!DOCTYPE>`, которая обычно выглядит примерно так:

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

Данный элемент предназначен для указания типа текущего документа и необходим для того чтобы браузер понимал, как следует интерпретировать текущую веб-страницу. Использование `<!DOCTYPE>` необходимо, поскольку HTML существует в нескольких версиях, кроме того, имеется XHTML (EXtensible HyperText Markup Language, расширенный язык разметки гипертекста), похожий на HTML, но различающийся с ним по синтаксису.

В след за элементом `<!DOCTYPE>` обозначается начало и конец документа при помощи тегов `<html>` и `</html>` соответственно. Внутри этих тегов должны находиться теги заголовка (`<head></head>`) и тела (`<body></body>`) документа.

Рассмотрим используемые строки декларации для HTML документов. Они различаются в зависимости от версии языка, на которого ориентированы. В таблице 1 приведены основные типы документов с описанием их элементов `<!DOCTYPE>`.

Табл. 1. Строки декларации для HTML документов

DOCTYPE	Описание
HTML 4.01	
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">	Строгий синтаксис HTML. (Strict)
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">	Переходный синтаксис HTML. (Transitional)
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd">	В HTML-документе применяются фреймы.
HTML 5	
<!DOCTYPE html>	Для всех документов.

Так строгий синтаксис HTML означает, что документ не содержит элементов, помеченных как «устаревшие» или «не одобряемые». Переходный подразумевает, что в документе могут содержаться устаревшие теги в целях совместимости и упрощения перехода со старых версий HTML. Синтаксис с фреймами (Frameset) аналогичен переходному, но содержит также теги для создания наборов фреймов.

Тег <head> предназначен для хранения элементов, которые помогают браузеру в работе с данными. Также внутри контейнера <head> находятся метатеги используемые для хранения информации предназначенной для браузеров и поисковых систем. В качестве примера тут могут служить механизмы поисковых систем обращаются к метатегам для получения описания сайта, ключевых слов и других данных. Само содержимое тега <head> не отображается напрямую на веб-странице. Исключением является тег <title> устанавливающий заголовок окна веб-страницы.

Два метатега предназначены специально для поисковых серверов: **description** (описание) и **keywords** (ключевые слова). Так большинство поисковых серверов отображают содержимое поля **description** при выводе результатов поиска. В случае если этого тега нет на странице, то поисковый движок просто перечислит первые встречающиеся слова на странице. Метатег **keywords** был предназначен для описания ключевых слов, встречающихся на странице. Но в результате действия людей, желающих попасть в верхние строчки поисковых систем любыми средствами, теперь дискредитирован. В связи с этим многие поисковики пропускают этот параметр.

Для того чтобы сообщить браузеру, в какой кодировке находятся символы веб-страницы, необходимо установить параметр:

```
<meta http-equiv="Content-Type" content="text/html; charset=имя кодировки">.
```

Для операционной системы Windows и кириллицы **charset** обычно принимает значение **utf-8** или **windows-1251**. Если браузер не обнаружит указание кодировки в HTML документе, то он попытается сам определить, какой тип символов используется и выберет необходимую кодировку авто-

матически. Однако браузер не всегда может точно распознать язык веб-страницы.

Перейдем к рассмотрению представления HTML документа в виде объектной модели.

ОБЪЕКТНАЯ МОДЕЛЬ ДОКУМЕНТА

При открытии любого HTML документа браузер предварительно производит разбор его содержимого и на основе этого разбора создает объектную модель HTML документа или более коротко DOM.

Объектная Модель Документа (DOM) – это программный интерфейс (API) для HTML и XML документов. DOM предоставляет структурированное представление документа и то как эта структура может быть доступна из програм, позволяя изменять содержимое, стиль и структуру документа. Представление DOM состоит из вложенных друг в друга иерархически расположенных объектов, которые называются узлами. Каждый узел в структуре представляет располагающийся на странице HTML элемент. DOM полностью поддерживает объектно ориентированное представление веб страницы, делая возможным её изменение при помощи языка описания сценариев на подобие JavaScript. По существу он соединяет веб страницу с языками описания сценариев либо языками программирования. Рассмотрим пример HTML документа и DOM, которая бы была создана браузером на основе его кода (рис. 1):

```
<!DOCTYPE html>
<html>
  <head>
    <title>HTML DOM</title>
  </head>
  <body>
    <h1>HTML DOM.</h1>
    <p style="color:green">Привет всем.</p>
  </body>
</html>
```

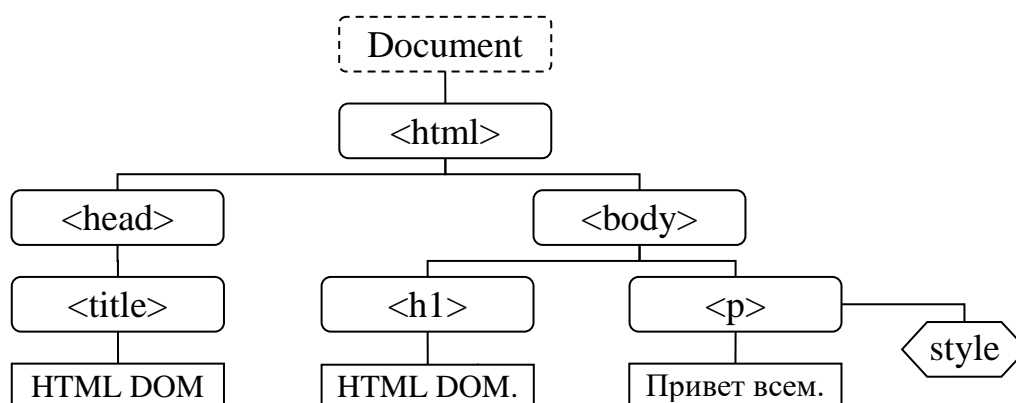


Рис. 1. Схема документа DOM

Все фигуры, изображенные на картинке, являются объектами (или узлами). Разными фигурами на изображение отмечены узлы разного **типа**. Так пунктирной линией отмечен узел **Document**. Любое обращение к DOM должно начинаться с обращения к данному узлу. Прямоугольником со скругленными краями отмечены **элементные узлы**. Для каждого HTML элемента на странице браузер создает соответствующий элементный узел. Содержимое элементов хранится в **текстовых узлах**. Текстовые узлы отмечены на нашей схеме прямоугольниками. Для каждого атрибута HTML элемента создается **атрибутный узел**. Атрибутный узел показан на схеме шестигранником.

Отношения между узлами

Узлы в объектной структуре связаны друг с другом. Существует несколько специальных терминов для описания отношений между узлами:

Родительский узел (parent node) – родительским узлом по отношению к рассматриваемому объекту является узел, в который вложен рассматриваемый объект. На нашей схеме по отношению к узлам `<h1>` и `<p>` узлом `<body>` является родительским. Для узла `<title>` родительским является узел `<head>`.

Узлы-потомки (child node) – узлом-потомком по отношению к рассматриваемому объекту является узел, который вложен в рассматриваемый объект. На нашей схеме по отношению к узлу `<body>` узлы `<h1>` и `<p>` являются потомками. Для узла `<head>` потомком является `<title>`.

Узлы-братья (sibling node) – узлы находящиеся на одинаковом уровне вложенности по отношению к их родительскому узлу. На нашей схеме узлами-братьями являются `<body>` и `<head>`, `<p>` и `<h1>`.

Самый верхний узел в DOM называется **корневым**. На нашей схеме корневым является `<html>` (т.к. объект document не является частью DOM).

CSS

CSS (Cascading Style Sheets – каскадные таблицы стилей) – технология описания внешнего вида документа, написанного языком разметки. Преимущественно **CSS** используется как средство оформления веб-страниц в формате **HTML** и **XHTML**, но может применяться с любыми видами документов в формате **XML**.

СПОСОБЫ ДОБАВЛЕНИЯ СТИЛЕЙ НА СТРАНИЦУ

Для добавления стилей на веб-страницу существует несколько способов, которые различаются своими возможностями и назначением. Рассмотрим их подробнее. При отображении страницы таблица стилей может быть получена из различных источников. В случае авторских стилей (т.е. когда информация стилей, предоставляемая автором страницы) в качестве источников могут выступать:

- Внешние (linking) таблицы стилей, то есть имеется отдельный файл .css, на который делается ссылка в документе.
- Вложенные (embedding) стили, подразумевающие наличие блоков CSS внутри самого HTML-документа.
- Встроенные (inline) стили, которые указывается в атрибуте style элемента страницы.

При использовании внешних стилей описание селекторов и их значений располагается в отдельном файле, а для связывания документа с этим файлом применяется тег `<link>`. Данный тег помещается в контейнер `<head>`:

```
<head>
  <meta charset="utf-8">
  <title>Стили</title>
  <link rel="stylesheet" href="mysite.css">
  <link rel="stylesheet" href="http://www.my.ru/main.css">
</head>
```

Значение атрибута тега `<link>` – **rel** остаётся неизменным независимо от кода, как приведено в данном примере. Значение **href** задаёт путь к CSS-файлу, он может быть задан как относительно, так и абсолютно. Следует отметить, что таким образом можно подключать таблицу стилей, которая находится на другом сайте.

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

```

<head>
  <meta charset="utf-8">
  <title>Глобальные стили</title>
  <style>
    H1 {
      font-size: 120%;
      color: #333366;
    }
  </style>
</head>

```

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

```

<body>
  <p style="font-size: 120%; color: #cd66cc">Пример</p>
</body>

```

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

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

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

Возможно выполнение соединения несколько CSS файлов, т.е. в текущую стилевую таблицу можно импортировать содержимое других CSS-файлов. Для этих целей используется директива **@import**. Указанные в директиве CSS-файлы загружаются и присоединяются к тому CSS в котором она встретилась. Общий синтаксис соединения CSS файлов следующий:

```

@import url("имя файла") типы носителей;
@import "имя файла" типы носителей;

```

После ключевого слова **@import** указывается путь к стилевому файлу одним из двух приведенных способов – с помощью **url** или без него. Пример фрагмента файла CSS с директивой **@import**:

```

@import url(css/base.css);
@import url(css/inner.css);

```

```
@import url(css/profile.css);  
/* далее идут обычные CSS-правила */
```

Следует помнить, что если в CSS есть директива **@import**, то она должна находиться в самом начале таблицы (перед всеми правилами). В противном случае браузер может ее проигнорировать. Поскольку директива **@import** представляет собой CSS-конструкцию, она должна находиться либо в CSS-файле, либо внутри тега `style`. Использовать **@import** во встроенных стилях нельзя!

СТИЛЕВЫЕ ПРАВИЛА CSS

Стилевые правила записываются в своём формате, отличном от HTML. Основным понятием выступает селектор – это некоторое имя стиля, для которого добавляются параметры форматирования. В качестве селектора выступают теги, классы и идентификаторы. Общий способ записи имеет следующий вид:

селектор {*свойство1*: значение1 значение2; *свойство2*: значение;}

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

Виды селекторов

Селектор служит для того, чтобы однозначно определить элемент в html документе, к которому желаем применить тот или иной стиль CSS. Существует несколько видов CSS селекторов. Перейдем к рассмотрению этих видов и примеров их применения.

Универсальный селектор, задает стиль всем элементам документа. Этот селектор обозначается символом звездочки (*). Например, следующий фрагмент стиля определяет, что все элементы будут иметь красный цвет:

```
* {color: red;}
```

Селектор элементов(тегов), в качестве селектора здесь выступает имя тега, для которого необходимо изменить свойства. При описания стиля используется общая форма описания стилей где в качестве селектора указывается имя необходимого тега:

```
Тег { свойство1: значение; свойство2: значение; ... }
```

Например:

```
h1 {  
    font-family:Verdana;  
    font-size:12px;  
    color:#666;  
}
```

В данном случае все заголовки первого уровня в документе будут иметь шрифт Verdana, размер 12px и серый цвет.

Селекторы классов. Данный вид селектора позволяет назначить стиль независимо от типа элемента. Однако для работы такого селектора нужно добавить в HTML код элемента атрибут **class** со значением, которое будет использовано при создании CSS правила. Классы применяют, когда необходимо определить стиль для индивидуального элемента веб-страницы или задать разные стили для одного тега.

При использовании совместно с тегами синтаксис для классов будет следующий:

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

Имена классов должны начинаться с латинского символа и могут содержать в себе символ дефиса (-) и подчеркивания (_). Использование русских букв в именах классов недопустимо.

Можно, также, использовать классы и без указания тега. Синтаксис в этом случае будет следующий.

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

К любому тегу одновременно можно добавить несколько классов, перечисляя их в атрибуте **class** через пробел. В таком случае к элементу применяется стиль, описанный в правилах для каждого класса. Бывают ситуации, когда для тега добавляются нескольких классов содержать одинаковые стилевые свойства, но с разными значениями. В таком случае будет использоваться значение свойства у класса, который в коде описан ниже.

Селекторы по идентификатору (ID селекторы) определяет уникальное имя элемента, которое используется для изменения его стиля. Синтаксис в этом случае следующий:

```
#Имя_идентификатора { свойство1: значение; ... }
```

При описании селектора по идентификатору вначале указывается символ решётки (#), а затем идет имя идентификатора. В отличие от классов идентификаторы должны быть уникальны, т.е. встречаться в коде документа только один раз. Описание идентификатора элемента происходит аналогично классам, но в качестве ключевого слова у тега используется атрибут **id**.

Как и при использовании классов, идентификаторы можно применять к конкретному тегу. Синтаксис при этом будет следующий:

Тег#Имя_идентификатора { свойство1: значение; ... }

Вначале указывается имя тега, затем без пробелов символ решётки и название идентификатора.

Контекстные селекторы, применяются только в определённом контексте. Например, при создании веб-страницы мы хотим задать определённый стиль для тега только в том случае, когда он располагается внутри контейнера <p>. Применяя контекстный селектор можно установить стиль для тега находящегося внутри другого. При записи в CSS стиле контекстный селектор состоит из простых селекторов разделенных пробелом:

Тег1 Тег2 { ... }

В этом случае стиль будет применяться к Тегу2 когда он размещается внутри Тег1, как показано ниже.

```
<Тег1>
  <Тег2> ... </Тег2>
</Тег1>
```

Контекстные селекторы не обязательно содержат только один вложенный тег. В зависимости от ситуации допустимо применять два и более последовательно вложенных друг в друга тегов.

Более широкие возможности контекстные селекторы дают при использовании идентификаторов и классов. Это позволяет устанавливать стиль только для того элемента, который располагается внутри определённого класса.

При таком подходе легко управлять стилем одинаковых элементов, вроде изображений и ссылок, оформление которых должно различаться в разных областях веб-страницы. Например, для HTML кода:

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset="utf-8">
    <title>Контекстные селекторы</title>
    <style>
      A {
        color: green; /* Зеленый цвет текста для всех ссылок */
      }
      .menu {
        padding: 7px; /* Поля вокруг текста */
        border: 1px solid #333; /* Параметры рамки */
        background: #fc0; /* Цвет фона */
      }
      .menu A {
        color: navy; /* Темно-синий цвет ссылок */
      }
    </style>
  </head>
  <body>
    <div class="menu">
      <a href="#">Ссылка</a>
    </div>
  </body>
</html>
```



```

    }
  </style>
</head>
<body>
  <div class="menu">
    <a href="http://site.ru/1.html">ссылка 1</a> |
    <a href="http://site.ru/2.html">ссылка 2</a>
  </div>
  <p>
    <a href="http://site.ru/add.html">Другие материалы </a>
  </p>
</body>
</html>

```

Будет получен результат, который отображен на рис. 2

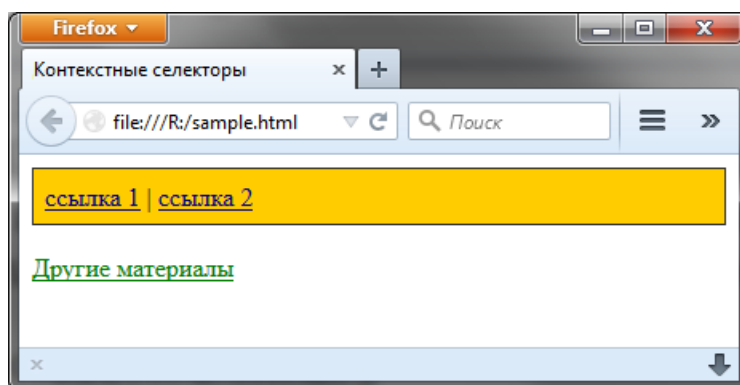


Рис. 2. Работа контекстных селекторов

В данном примере используется два типа ссылок. Первая ссылка, стиль которой задаётся с помощью селектора **A**, будет действовать на всей странице, а стиль второй ссылки (**.menu A**) применяется только к ссылкам внутри элемента с классом **menu**.

Дочерние селекторы – селекторы, определяемые для элементов находящихся прямо внутри родительского элемента согласно дерева элементов. Синтаксис таких селекторов будет следующим:

Селектор 1 > Селектор 2 { Описание правил стиля }

Стиль применяется к Селектору 2, но только в том случае, если он является дочерним для Селектора 1.

По своей логике дочерние селекторы похожи на контекстные селекторы. Однако между ними существует принципиальная разница. Стиль к дочернему селектору применяется только в том случае, когда он является прямым потомком, т.е. непосредственно располагается внутри родительского элемента в то время как для контекстного селектора допустим любой уровень вложенности. Рассмотрим пример:

```

<!DOCTYPE HTML>
<html>
<head>
  <meta charset="utf-8">
  <title>Дочерние селекторы</title>
  <style>
    DIV I { /* Контекстный селектор */
      color: green; /* Зеленый цвет текста */
    }
    P > I { /* Дочерний селектор */
      color: red; /* Красный цвет текста */
    }
  </style>
</head>
<body>
  <div>
    <p><i>Lorem ipsum dolor sit amet</i>, consectetur
    adipiscing elit, sed diam nonummy nibh euismod tincidunt
    ut laoreet <i>dolore magna</i> aliquam erat volutpat.</p>
  </div>
</body>
</html>

```

Проанализировав код примера можно заметить, что на тег `<i>` действуют одновременно два правила: контекстный селектор (тег `<i>` расположен внутри `<div>`) и дочерний селектор (тег `<i>` является дочерним по отношению к `<p>`). При этом правила являются равносильными, поскольку все условия для них выполняются и не противоречат друг другу. В подобных случаях применяется стиль, который расположен в коде ниже, поэтому курсивный текст отображается красным цветом (рис. 3).

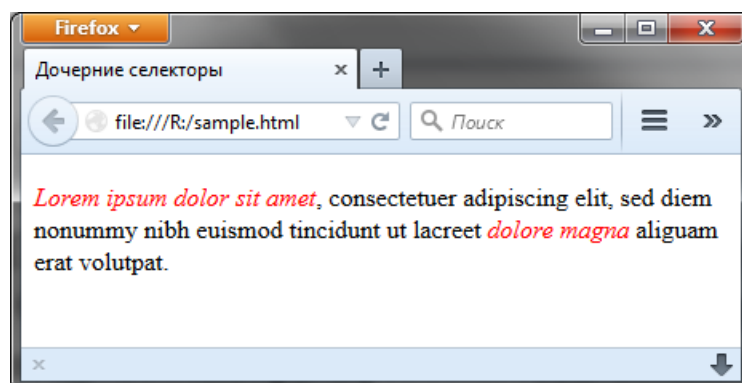


Рис. 3. Работа дочерних селекторов

Однако если поменять правила местами и поставить `DIV I` ниже, то цвет текста изменится с красного на зеленый (рис. 4).

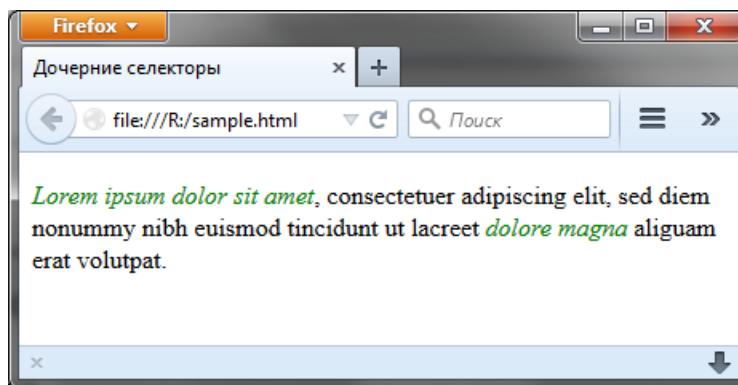


Рис. 4. Работа дочерних селекторов

Соседние селекторы, используются для управления стилем соседних элементов. Так, например если есть фрагмент HTML кода:

```
<p>Lorem <b>ipsum </b> dolor sit amet, <i>consectetuer</i>
adipiscing <strong>elit</strong>.</p>
```

```
<p>Lorem ipsum dolor sit amet, <i>consectetuer</i> adipiscing
elit.</p>
```

то в нем соседними будут являться теги `` и `<i>`, а также `<i>` и ``. При этом теги `` и `` не будут считаться соседними элементами поскольку между ними расположен контейнер `<i>`.

Для задания стиля соседних элементов используется символ плюс (+), устанавливаемый между двумя селекторами. В общем случае синтаксис соседнего селектора будет следующим:

Селектор 1 + Селектор 2 { Описание правил стиля }

При написании селектора пробелы вокруг плюса не обязательны, стиль при такой записи применяется к Селектору 2, но только в том случае, если он является соседним для Селектора 1 и следует сразу после него.

В случае если применить стиль:

```
B + I {
    color: red; /* Красный цвет текста */
}
```

к рассмотренному выше фрагменту html кода, то происходит изменение цвета текста для содержимого контейнера `<i>`, когда он располагается сразу после контейнера ``.

Селекторы атрибутов, применяются для выбора элементов на основании их атрибутов, а так же значений этих атрибутов. Среди селекторов данного вида выделяют два основных класса подвида – простой селектор атрибута и селектор учитывающий атрибут со значением

В простом селектор атрибута устанавливает стиль для элемента, если задан специфичный атрибут для тега. Его значение в данном случае не важно. Синтаксис описания такого селектора следующий.

```
[атрибут] { Описание правил стиля }  
Селектор[атрибут] { Описание правил стиля }
```

Стиль применяется к тем тега, внутри которых добавлен указанный атрибут. Пробел между именем селектора и квадратными скобками не допускается.

Использование селектора учитывающий атрибут со значением задает стиль для элемента в том случае, если задано определённое значение специфичного атрибута. Синтаксис следующий:

```
[атрибут="значение"] { Описание правил стиля }  
Селектор[атрибут="значение"] { Описание правил стиля }
```

В первом случае стиль применяется ко всем тега, которые содержат указанное значение. А во втором – только к определённым селекторам. Бывает ситуации когда необходимо учитывать часть значения атрибута, в таком случае будет несколько изменяться форма записи. Примеры форм записи в зависимости от учета значения атрибута:

– значение атрибута начинается с определённого текста

```
[атрибут^="значение"] { Описание правил стиля }  
Селектор[атрибут^="значение"] { Описание правил стиля }
```

– значение атрибута оканчивается определённым текстом

```
[атрибут$="значение"] { Описание правил стиля }  
Селектор[атрибут$="значение"] { Описание правил стиля }
```

– значение атрибута содержит указанный текст

```
[атрибут*="значение"] { Описание правил стиля }  
Селектор[атрибут*="значение"] { Описание правил стиля }
```

– одно из нескольких значений атрибута

```
[атрибут~="значение"] { Описание правил стиля }  
Селектор[атрибут~="значение"] { Описание правил стиля }
```

Использование одного из нескольких значений атрибута будет полезно в том случае, когда идет перечисление значение, например имен классов. Так стиль применяется в том случае, если у атрибута имеется указанное значение или оно входит в список значений, разделяемых пробелом.

Псевдоклассы и псевдоэлементы

Псевдоклассы определяют динамическое состояние элементов, которое изменяется в зависимости от действий пользователя, а также положение

ние в дереве документа. Можно сказать что псевдоклассы это атрибуты, назначаемые строго к селекторам с намерением определить реакцию или состояние для данного селектора. При использовании псевдоклассов браузер не перегружает текущий документ, поэтому с помощью псевдоклассов можно получить разные динамические эффекты на странице. Синтаксис использования псевдоклассов в CSS стилях следующий:

Селектор:Псевдокласс { Описание правил стиля }

Вначале указывается селектор, к которому добавляется псевдокласс, затем следует двоеточие, после которого идёт имя псевдокласса.

Допускается применять псевдоклассы к именам идентификаторов или классов (A.menu:hover {color: green}), а также к контекстным селекторам (.menu A:hover {background: #fc0}). Если псевдокласс указывается без селектора впереди (:hover), то он будет применяться ко всем элементам документа.

Условно все псевдоклассы делятся на три группы:

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

Наибольший интерес представляет группа псевдоклассов определяющих состояние элементов. Примером использования таких псевдоклассов может служить текстовая ссылка, меняющая свой цвет при наведении на неё курсора мыши.

К этой группе относятся псевдоклассы, которые распознают текущее состояние элемента и применяют стиль только для этого состояния. Возможны следующие виды состояний:

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

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

focus – применяется к элементу при получении им фокуса. В качестве примера может служить текстовое поле формы. Получение фокуса будет означать, что курсор установлен в поле, и с помощью клавиатуры можно вводить в него текст

hover – псевдокласс активизируемый, когда курсор мыши находится в пределах элемента, но при этом щелчка по нему не происходит.

visited – псевдокласс применяемый к посещённым ссылкам.

Помимо псевдоклассов в CSS существует понятие псевдоэлементов. Псевдоэлементы – это селекторы, которые определяют область элементов,

изначально отсутствующих в дереве документа. Например, первая буква слова и первая строка абзаца. Эти элементы реально не существуют в виде HTML кода, но они присутствуют на странице – создаются искусственно с помощью CSS. Им можно задавать стиль, также как и элементам HTML.

Ключевое отличие между псевдоклассами и псевдоэлементами заключается в том, что первые определяют именно состояние элементов, которые уже существуют на странице, а вторые создают области (искусственные элементы), которых изначально на веб-странице не было. Но и те и другие отсутствуют в исходном коде документа. В общем виде синтаксис записи псевдоэлементов выглядит следующим образом:

```
элемент:псевдоэлемент { Описание правил стиля } //CSS2.1  
элемент::псевдоэлемент { Описание правил стиля } //CSS3
```

Так при оформлении текста используют следующие псевдоэлементы:

first-letter – определяет стиль первого символа в тексте элемента, к которому добавляется. К этому псевдоэлементу могут применяться только стилевые свойства, связанные со свойствами шрифта, полями, отступами, границами, цветом и фоном.

first-line – задает стиль первой строки форматированного текста. Длина этой строки зависит от таких факторов как используемый шрифт, размер окна, ширина блока, языка и т.д. В правилах стиля допустимо использовать только свойства, относящиеся к шрифту, изменению цвета текста и фона.

Рассмотрим следующий пример – фрагмент HTML кода страницы:

```
<style>  
  p:first-letter{ color:red; }  
  p:first-line{ color:blue; }  
</style>  
...  
<p>Это текст параграфа, и первую букву мы выделили красным  
цветом.</p>
```

Результат примера, показанном на рис. 5.

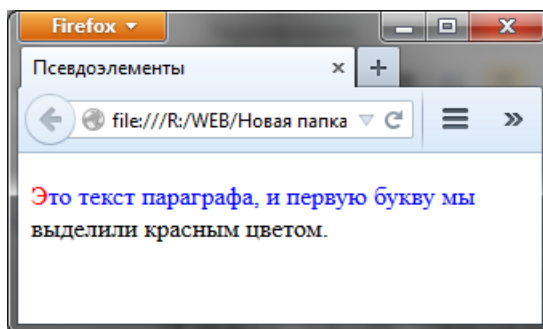


Рис. 5. Работа псевдоклассов

Также существуют псевдоэлементы **before** и **after** позволяющие добавлять содержимое до и после элемента, к которому были применены. Эти псевдоэлементы работают совместно со свойством **content**. Для псевдоэлементов **before** и **after** характерны следующие особенности:

- при добавлении к блочному элементу, значение свойства **display** может быть только: **block**, **inline**, **none**, **list-item**. Все остальные значения будут трактоваться как **block**.

- при добавлении к встроенному элементу, **display** ограничен значениями **inline** и **none**. Все остальные будут восприниматься как **inline**.

Для **before** также свойственна следующая особенность – он наследует стиль от элемента, к которому он добавляется.

Основные свойства

При работе со стилями CSS следует учитывать использования механизма наследования и группирования.

Под наследованием в CSS понимается перенос правил форматирования для элементов, находящихся внутри других. Такие элементы являются дочерними, и они наследуют некоторые стилевые свойства своих родителей, внутри которых располагаются. Чтобы определить, наследуется значение стилевого свойства или нет, требуется заглянуть в справочник по свойствам CSS и посмотреть там.

Наследование позволяет задавать значения некоторых свойств единожды, определяя их для родителей верхнего уровня.

Рассмотрим следующий пример

```
<div style="color:green">Первый слой  
  <div style="background:#cccccc">Второй слой внутри первого  
    <div style="color:red">Третий слой внутри второго</div>  
  </div>  
</div>
```

В первом блоке `div` устанавливается зеленый цвет текста. После того как в него помещается еще один блок, но со стилем задающим цвет фона. Можно наблюдать, что при отображении второго блока `div` изм цвет текста останется зеленым, поскольку наследуется от родительского элемента, а цвет фона задается оттенком серого. Наследуемые свойства можно переопределить, применив индивидуальное правило для нужного элемента. Так в примере для третьего элемента `div` переопределен цвет текста на красный (рис. 6).

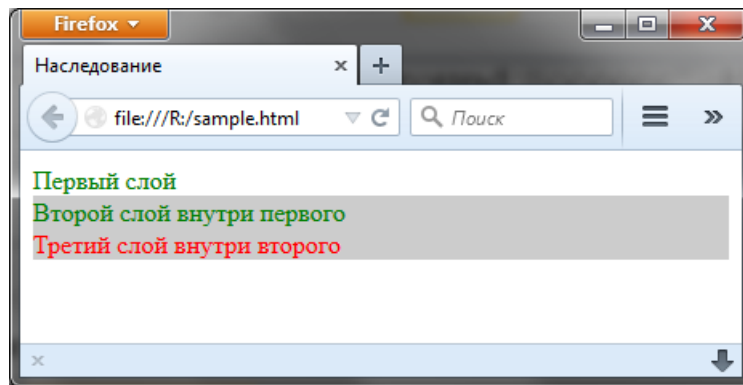


Рис. 6. Работа наследования

При создании стиля для сайта, когда одновременно используется множество селекторов, возможно появление повторяющихся стилевых правил. Чтобы не повторять дважды одни и те же элементы, их можно сгруппировать для удобства представления и сокращения кода.

Селекторы группируются в виде списка тегов, разделенных между собой запятыми. В группу могут входить не только селекторы тегов, но также идентификаторы и классы. Общий синтаксис следующий:

Селектор 1, Селектор 2, ... Селектор N { Описание стиля }

При такой записи правила стиля применяются ко всем селекторам, перечисленным в одной группе.

В следующем примере показана обычная запись, где для каждого селектора приводится свой набор стилевых свойств:

```
H1 {
  font-family: Arial, Helvetica, sans-serif;
  font-size: 160%;
  color: #003;
}
H2 {
  font-family: Arial, Helvetica, sans-serif;
  font-size: 135%;
  color: #333;
}
H3 {
  font-family: Arial, Helvetica, sans-serif;
  font-size: 120%;
  color: #900;
}
P {
  font-family: Times, serif;
}
```

По этому примеру можно заметить, что стиль для тегов заголовков содержит одинаковое значение **font-family**. Использование группирования

позволяет установить одно свойство сразу для нескольких селекторов. В таком случае фрагмент примера, настраивающий стили заголовка, будет выглядеть следующим образом:

```
h1, h2, h3 {
    font-family: Arial, Helvetica, sans-serif;
}
h1 {
    font-size: 160%;
    color: #003;
}
h2 {
    font-size: 135%;
    color: #333;
}
h3 {
    font-size: 120%;
    color: #900;
}
```

В рассмотренном фрагменте **font-family** единое для всех селекторов и применяется сразу к нескольким тегам, а индивидуальные свойства уже добавляются к каждому селектору отдельно.

ВЕРСТКА НА ОСНОВЕ БЛОКОВ С ИСПОЛЬЗОВАНИЕМ CSS

При использовании верстки на основе блоков структура страницы формируется из блоков, в качестве которых в HTML4 выступает тег `<div>` не имеющий никаких свойств отображения по умолчанию. С помощью стилей блоки позиционируются на странице определенным образом, формируя каркас, который уже затем наполняют содержимым. При этом все параметры блока (размеры, расположение, отступы, оформление и т.п.) определяются с помощью CSS. С появлением HTML5 были введены новые теги блоков, используемые при верстке:

header – задает «шапку» сайта или раздела, в которой обычно располагается заголовок;

article – задает содержание сайта вроде новости, статьи, записи блога, форума или др.

aside – определяет блок сбоку от контента для размещения рубрик, ссылок на архив, меток и другой информации. Такой блок, как правило, называется «сайдбар» или «боковая панель».

nav – задает блок навигации по сайту

section – группирует тематические блоки, т.е. задает раздел документа, который может включать в себя заголовки, шапку, подвал и текст. Элементы **section** могут быть вложены друг в друга.

footer – задаёт «подвал» сайта или раздела, в нём может располагаться имя автора, дата создания документа, контактная и правовая информация и т.п

figure – используется для группирования любых элементов, например, изображений и подписей к ним.

Введенные в HTML 5 новые теги используются также, как и обычные теги `div`. Единственная разница заключается в том, что данные теги семантически разделяют страницу. Другими словами, можете представить страницу таким образом, что станет сразу понятно про что она. В результате поисковые системы лучше будут обрабатывать страницу и соответственно при выполнении поисковых запросов это будет учитываться.

Разметка блоков с помощью HTML 4 осуществляется путем присвоения блокам `<div>` идентификаторов и классов (рис. 7):

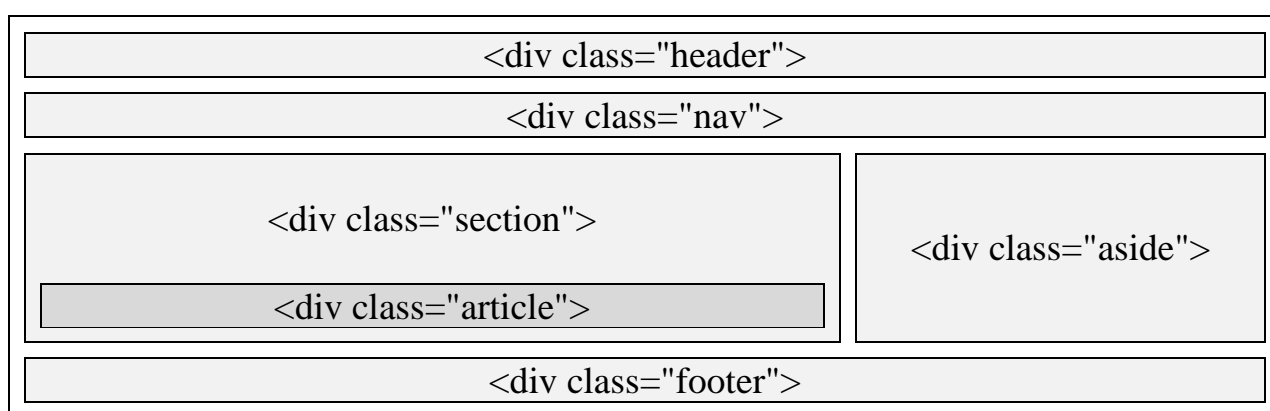


Рис. 7. Разметка блоков

В HTML 5 используются специализированные блоки (семантическая разметка). По умолчанию, во всех браузерах новые элементы будут инлайновыми, поэтому необходимо при помощи CSS задавать желаемые свойства отображения (рис. 8).

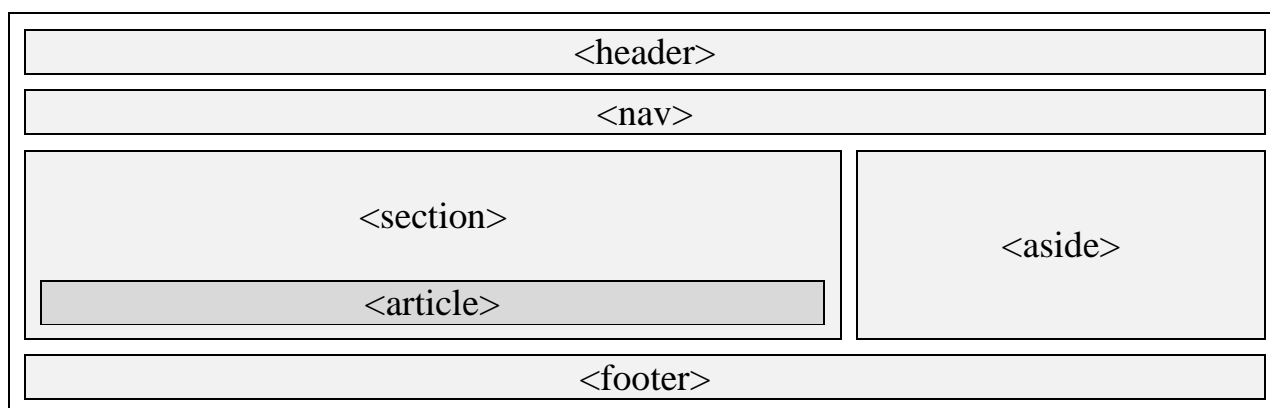


Рис. 8. Разметка блоков

Можно сказать, что HTML5 имеет семантический чистый код, а структура HTML5 более легкая, гибкая и функциональная.

Блочная модель CSS

В CSS используется модель представления визуально отображаемых элементов документа в виде прямоугольных блоков, возможно, вложенных друг в друга. Каждый такой блок имеет информационное содержимое (текст, изображения и т.д.), а также может иметь отступы (padding), границы (border) и поля (margin). Схематично эта модель представления показана на рис. 9.

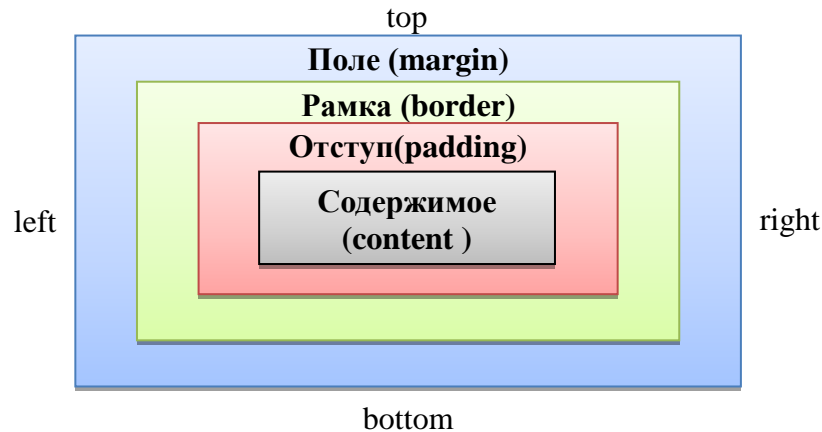


Рис. 9. Блочная модель CSS

Соответствующие свойства CSS позволяют оформить блок желаемым образом. Зона **padding** окружающая зону **content** используется для задания величины отступа содержимого элемента (content) от его границы (border). Она может быть разбита на четыре части, которые могут оформляться независимо друг от друга: padding-top, padding-right, padding-bottom, padding-left. В общем виде синтаксис настройки свойства padding выглядит следующим образом:

padding: [значение | проценты] {1, 4} | inherit

Разрешается использовать одно, два, три или четыре значения, разделяя их между собой пробелом. В зависимости от количества значений будет устанавливаться различные параметры применения:

1. – поля устанавливаются одновременно от каждого края элемента.
2. – первое значение устанавливает поля от верхнего и нижнего края, второе – от левого и правого.
3. – первое значение задает поле от верхнего края, второе – одновременно от левого и правого края, а третье – от нижнего края.
4. поочередно устанавливается поля от верхнего, правого, нижнего и левого края.

Свойство **margin** устанавливает величину отступа от каждого края элемента. Отступом в данном случае является пространство от границы текущего элемента до внутренней границы его родительского элемента.

Если у элемента нет родителя, отступом будет расстояние от края элемента до края окна браузера с учетом того, что у самого окна по умолчанию тоже установлены отступы. Синтаксис и особенности установки значений аналогичны свойству **padding**.

Свойство **border** позволяет одновременно установить толщину (**border-width**), стиль (**border-style**) и цвет (**border-color**) границы вокруг элемента. Значения могут идти в любом порядке, разделяясь пробелом, браузер сам определит, какое из них соответствует нужному свойству.

Для установки границы только на определенных сторонах элемента, используются соответствующие свойства: **border-top**, **border-bottom**, **border-left**, **border-right**. Синтаксис установки свойства:

```
border: [border-width || border-style || border-color] inherit
```

Примеры видов границ можно увидеть на рис. 10:

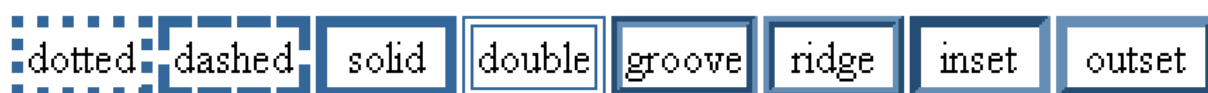


Рис. 10. Виды границ в CSS

Ознакомившись с блочной моделью CSS перейдем к рассмотрению свойств, которые будут полезны при верстке HTML страницы.

Описание свойств CSS для блочной вёрстки

Базовые свойства в CSS используемые для блочной вёрстки это **width**, **height** определяющие ширина и высота блока. Обычно значения для этих свойств задаются в пикселях или процентах. В случае если высота у блока не задана явно, то она будет определена автоматически в зависимости от высоты содержимого в блоке.

Следующее свойство, которое будет рассмотрено это **position**. Оно устанавливает способ позиционирования элемента относительно окна браузера или других объектов на веб-странице. У данного свойства возможны следующие значения: **absolute**, **fixed**, **relative**, **static** и **inherit**.

absolute – указывает на то, что элемент абсолютно позиционирован. При этом другие элементы отображаются на веб-странице словно абсолютно позиционированного элемента и нет. Положение элемента задается свойствами **left**, **top**, **right** и **bottom**, также на положение влияет значение свойства **position** родительского элемента. Так, если у родителя значение **position** установлено как **static** или родителя нет, то отсчет координат ведется от края окна браузера. Если у родителя значение **position** задано как **fixed**, **relative** или **absolute**, то отсчет координат ведется от края родительского элемента.

fixed – по своему действию схоже со значением **absolute**, но в отличие от него привязывается к указанной свойствами **left**, **top**, **right** и **bottom**

точке на экране и не меняет своего положения при прокрутке веб-страницы.

relative – положение элемента устанавливается относительно его исходного места. Добавление свойств **left**, **top**, **right** и **bottom** изменяет позицию элемента и сдвигает его в ту или иную сторону от первоначального расположения.

static – элементы отображаются как обычно. При этом использование свойств **left**, **top**, **right** и **bottom** не приводит к каким-либо результатам.

inherit – наследует значение родителя.

Важным свойством для выполнения верстки является свойство **float**. Оно определяет, по какой стороне будет выравниваться элемент, остальные же элементы будут обтекать его с других сторон. Когда значение свойства **float** равно **none**, элемент выводится на странице как обычно. Допускается, что одна строка обтекающего текста может быть на той же линии, что и сам элемент. Возможно выравнивание элемента по левому краю (значение **left**), в таком случае все остальные элементы, вроде текста, обтекают его по правой стороне. Также можно установить и обратное выравнивание – по правому краю (значение **right**), соответственно остальные элементы будут обтекать его по левой стороне

При верстке достаточно часто используется многоцелевое свойство **display**, определяющее, как элемент должен быть показан в документе. Список возможных значений этого свойства, понимаемый разными браузерами очень короткий – **block**, **inline**, **inline-block**, **list-item** и **none**. Все остальные допустимые значения поддерживаются браузерами выборочно. Рассмотрим эти значения:

block – элемент показывается как блочный. Применение этого значения для встроенных элементов, например тега ``, заставляет его вести подобно блокам. При таком способе отображения происходит перенос строк в начале и в конце содержимого.

inline – элемент отображается как встроенный. Использование блочных тегов, таких как `<div>` и `<p>`, автоматически создает перенос и показывает содержимое этих тегов с новой строки. Значение **inline** отменяет эту особенность, поэтому содержимое блочных элементов начинается с того места, где окончился предыдущий элемент.

inline-block – это значение генерирует блочный элемент, который обтекается другими элементами веб-страницы подобно встроенному элементу. По сути такой элемент по своему действию похож на встраиваемые элементы (вроде тега ``). При этом его внутренняя часть форматируется как блочный элемент, а сам элемент рассматривается как встроенный.

list-item – элемент выводится как блочный и добавляется маркер списка.

none – временно удаляет элемент из документа. Занимаемое им место не резервируется и веб-страница формируется так, словно элемента и не было. Изменить значение и сделать вновь видимым элемент можно с помощью скриптов, обращаясь к свойствам через объектную модель. В

этом случае происходит переформатирование данных на странице с учетом вновь добавленного элемента.

При выполнении верстки возможно использование свойства **z-index** позволяющего регулировать виртуальную глубину (слой) расположения объекта. Это свойство работает только для элементов, у которых значение **position** задано как *absolute*, *fixed* или *relative*. В качестве значения у данного свойства используются целые числа (положительные, отрицательные и ноль). Чем больше значение, тем выше находится элемент по сравнению с теми элементами, у которых оно меньше. При равном значении **z-index**, на переднем плане будет находиться тот элемент, который в HTML коде описан ниже.

Ознакомившись с основными свойствами перейдем к рассмотрению примера верстки на основе блоков.

Описание верстки на примере

Есть несколько видов верстки, которая отличается визуально – это фиксированная и резиновая виды верстки. Рассмотрим их основные особенности.

Фиксированная верстка означает возможность основного контейнера сайта иметь одинаковую ширину независимо от разрешения монитора. Если экран меньше чем фиксированный размер основного контейнера, то появляется боковой скроллинг (прокрутка влево-вправо).

Резиновая верстка – возможность основного контейнера сайта растягиваться в ширину от и до указанных минимальных и максимальных размеров. Резиновая верстка может быть без указания минимального или максимального размера. Однако если не указывать хотябы минимальный размер то страница может не очень хорошо выглядеть т.е. она будет открыта на экранах меньше того на который рассчитывал дизайнер, то статичные объекты могут наплывать друг на друга.

В **резиновой верстке** в качестве значений в большинстве случаев используют проценты или другие относительные единицы, в связи с чем расположение блоков будет растягиваться и сжиматься в зависимости от разрешения экрана. Это довольно распространенный метод так как позволяет не задумываться о корректном отображении сайта.

Предположим, что необходимо создать верстку для сайта, содержащего следующие элементы:

- баннер в самом верху;
- шапку сайта, на которой расположено горизонтальное меню;
- левый блок, содержащий дополнительное меню и некоторую информацию;
- область контента в которой отображаются некоторые данные визуально отделяемые друг от друга;
- блок с правовой информацией.

Сформируем макет размещения блоков для понимания того, как следует настраивать стили. Схематично размещение элементов согласно макета будет выглядеть следующим образом (рис. 11):

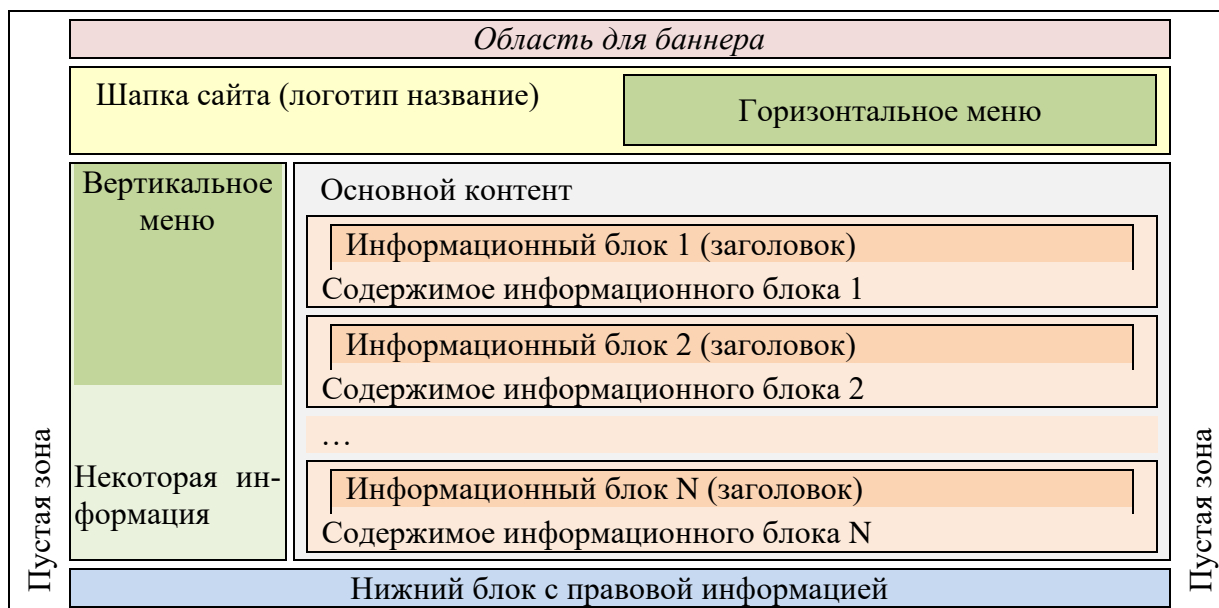


Рис. 11. Схема размещения элементов страницы

Для реализации этого макета воспользуемся возможностями HTML5 и CSS. В общем виде HTML код страницы будет выглядеть следующим образом:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset=utf-8>
    <title>Пример верстки</title>
  </head>
  <body>
    <header class="top_banner">
      баннер
    </header>
    <header class="site_header">
      Шапка сайта (логотип название)
      <nav class="top_menu">
        Горизонтальное меню
      </nav>
    </header>
    <div id="main_wrapper">
      <aside>
        <nav class="left_menu">
          Вертикальное меню
        </nav>
```

```

</aside>
<section>
    Основной контент
    <article class="infoblock">
        <header class="infoblock_head">
            Информационный блок 1
        </header>
        Содержимое информационного блока 2
    </article>
    <article class="infoblock">
        <header class="infoblock_head">
            Информационный блок 2
        </header>
        Содержимое информационного блока 2
    </article>
</section>
</div>
<div id="clearD"></div>
<footer>
    copyright
</footer>
</body>
</html>

```

Если просмотреть данную страницу через браузер, то мы увидим, что элементы просто размещены на странице в форме текста (рис. 12).

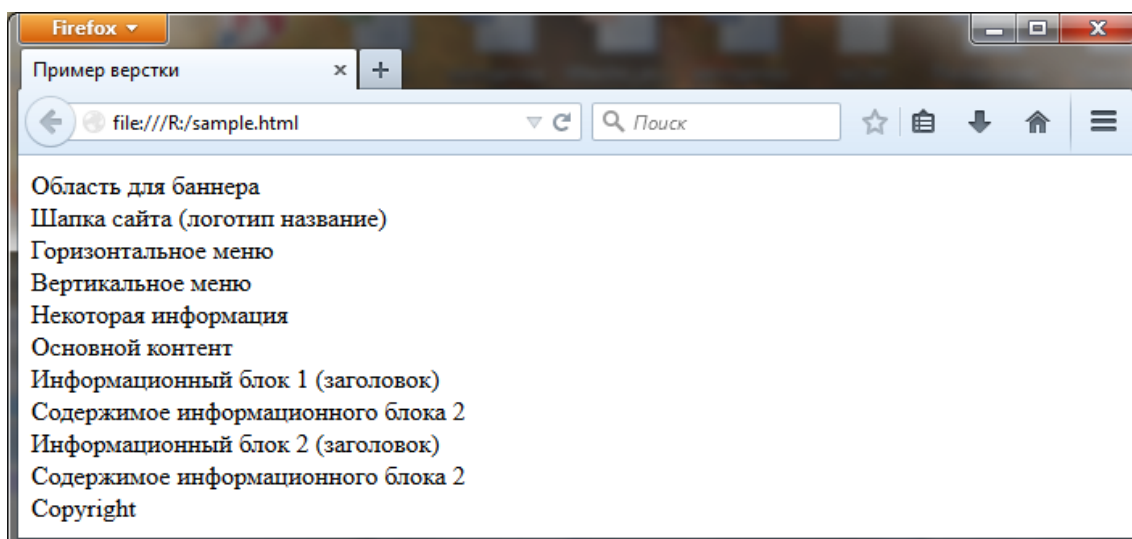


Рис. 12. Вид страницы в браузере

Для того чтобы задать расположение элементов воспользуемся стилями CSS. Поскольку в браузерах используемые HTML 5 теги имеют вид отображения inline необходимо добавить следующий стиль CSS:

```
header, nav, section, article, aside, footer {
```



```
display: block }
```

Для начала зададим свойства для блоков занимающих горизонтальное пространство. При этом воспользуемся подходом верстки использующего «резиновые» макеты, т.е. в таком случае зададим минимальные размеры в пикселах, а отступы от краев экрана и ширину элементов в процентах. Настроим стиль следующим образом:

```
header.top_banner, header.site_header, footer, #main_wrapper {  
    margin: 5px auto; min-width:600px; width:94%; }
```

Теперь настроим для этих элементов их визуальное отображение:

```
header.top_banner, footer{ text-align:center; }
```

```
header.top_banner{ height: 35px; padding-top:15px;  
    background-color:#F2DCDC;}
```

```
header.site_header{ height: 50px; background-color:#FFFFB4; }
```

```
footer{ height: 20px; background-color:RGB(198,217,240); }
```

на текущем этапе верстки наш макет выглядит следующим образом (рис. 13):

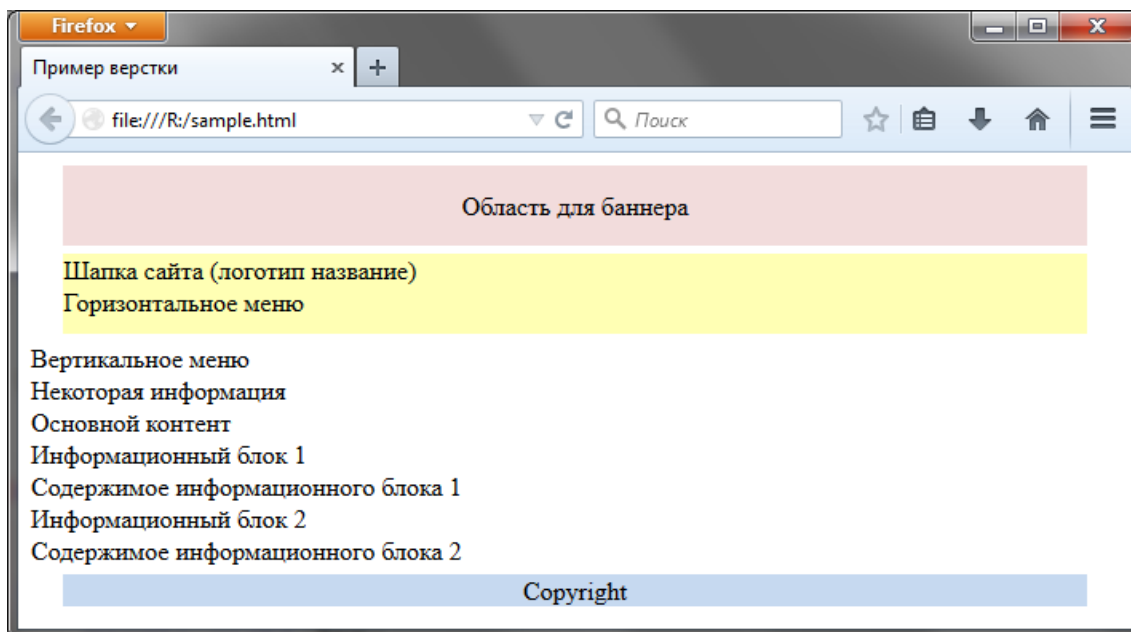


Рис. 13. Вид страницы в браузере

Зададим параметры стилей для блоков aside и section:

```
aside { float:left; background-color:#C6D9C8;  
    margin-right:5px; width:160px;}
```

```
section{ padding:3px; background-color: #ECECEC; }
```

После добавление этих стилей в файл и отображении страницы в браузере получим следующий результат (рис. 14).

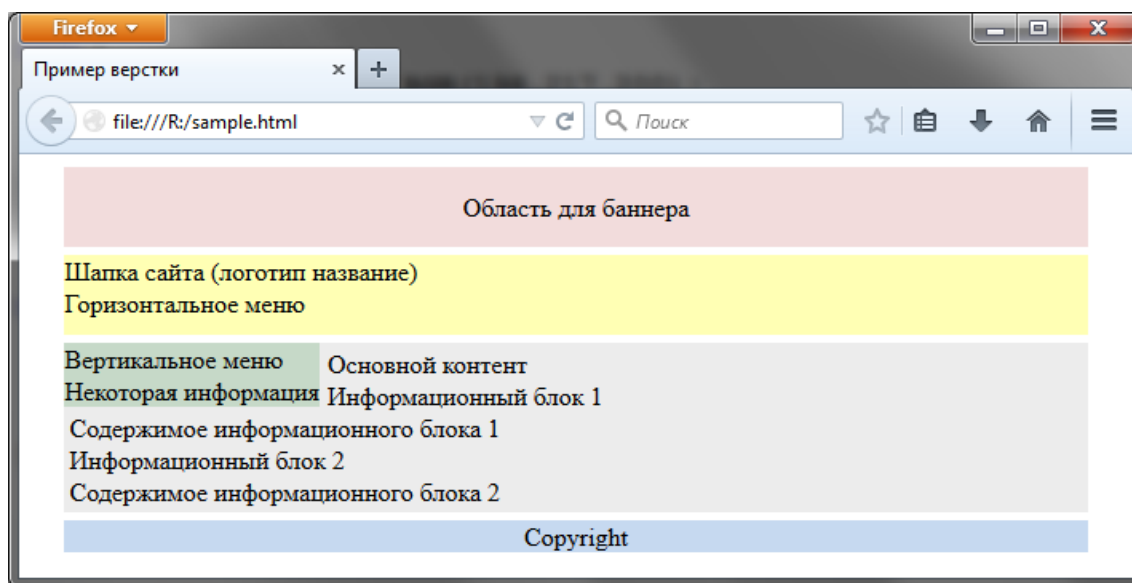


Рис. 14. Вид страницы в браузере

Как можно заметить, область блока **section** обтекает область **aside** как планировалось, но при этом содержимое блоков не выстраивается как предполагалось в макете. Для того чтобы область блока **section** отображалась в виде единой колонки воспользуемся свойством **overflow**. Данное свойство управляет отображением содержания блочного элемента. Назначим ему значение **hidden** для того чтобы скрыть не нужные фрагменты области.

```
section{ padding:3px; background-color: #ECECEC;
  overflow:hidden; }
```

В результате мы увидим разметку (рис. 15) похожую на то, что предполагает макет.

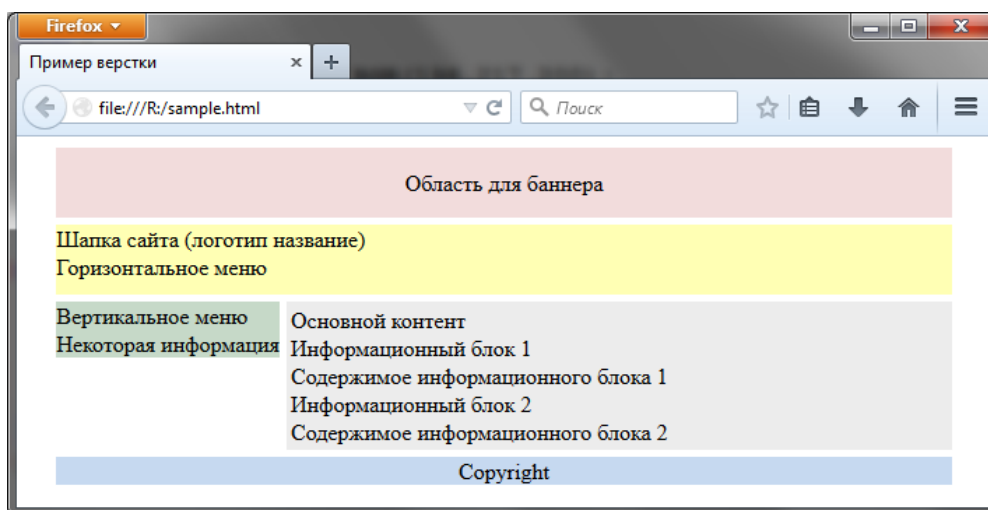


Рис. 15. Вид страницы в браузере

Для элемента div с id равным clearD задаем свойство отменяющее обтекание элемента одновременно с правого и левого края. Это необходимо для выравнивания высоты расположения нижнего блока.

```
#clearD { clear:both; }
```

Остается только настроить области для меню и информационных блоков. Для этого зададим следующие CSS стили:

```
article.infoblock {background-color:#F4CD8C; margin:10px 15px;
padding:2px 8px; border:#C90 solid thin;
border-top-left-radius:15px;
border-bottom-right-radius:5px; }
```

```
header.infoblock_head { padding:0px 8px; text-align:center; }
```

```
nav { background-color: #9DB678; text-align:center; }
```

```
nav.top_menu { float:right; width:280px; height:25px;
padding:8px; margin-top:5px; margin-right:10px;
border-top-left-radius:15px;
border-bottom-right-radius:5px; }
```

```
nav.left_menu { margin:5px 4px; margin-right:10px;
height:80px; }
```

Применив эти стили, мы получим HTML страницу в том виде в котором она представлена на рис. 16.

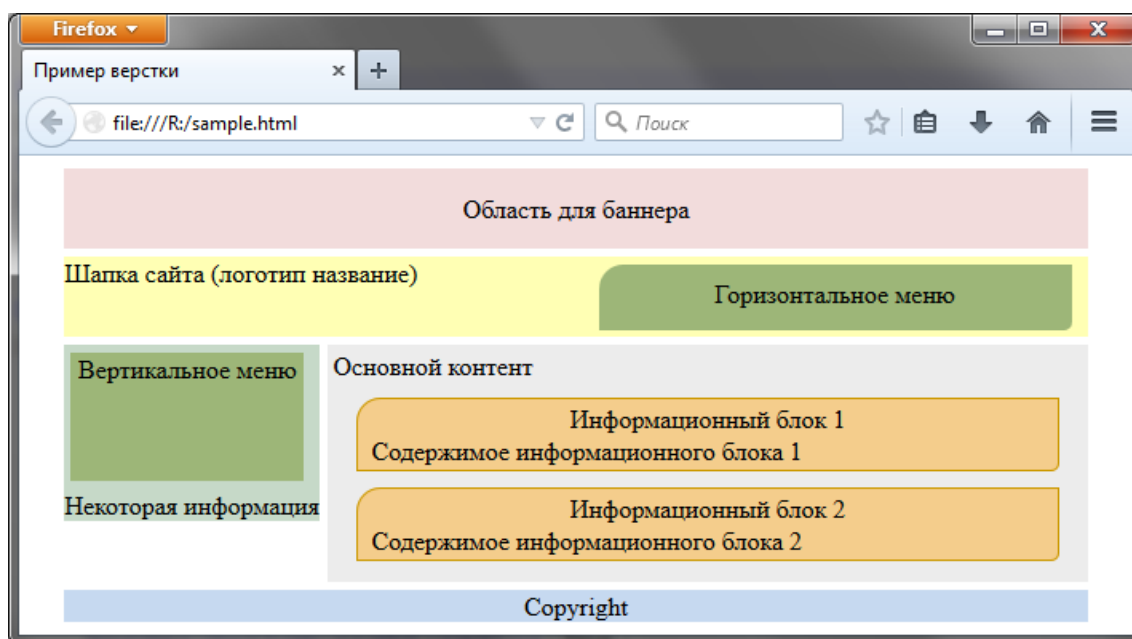


Рис. 16. Вид страницы в браузере

Когда у нас есть «скелет» страницы остается добавить меню в соответствующие области. Теперь перейдем к рассмотрению того как можно организовать меню на сайте.

ОРГАНИЗАЦИЯ МЕНЮ

Под меню понимают набор ссылок, позволяющих переходить к разным разделам или документам сайта. Меню непосредственно связано с навигацией по сайту – системой организации документов и их взаимодействия между собой. Другими словами, навигация дает пользователю представление о структуре сайта и возможность перемещаться к нужной странице. Термин *навигация* давно уже стал широким понятием и включает в себя не только способ перехода от страницы к странице, но также вид и представлениессылок. По этой причине к навигации относят элементы страницы, которые имеют к ней косвенное отношение, например меню.

Правильно организованное меню предоставляет пользователю возможность быстрого доступа к нужным ему разделам, показывает, где он находится в данный момент в структуре сайта и что на нем еще можно посмотреть.

Условно все типы меню можно отнести к следующим категориям.

- *Вертикальное меню.* Пункты меню располагаются друг под другом, и число их может быть достаточно велико. В силу своей универсальности вертикальное меню встречается на сайтах наиболее часто.

- *Горизонтальное меню.* В этом случае пункты помещаются по горизонтали, но чтобы это не привело к появлению горизонтальной полосы прокрутки, их число ограничивают или располагают в два-три ряда.

- *Ниспадающее меню.* Обычно выглядит как горизонтальное меню, но когда курсор мыши наводится на пункты, открывается дополнительное подменю.

- *Всплывающее меню.* При наведении на ссылку курсора мыши такое меню появляется в виде панели с набором вариантов перехода. Как только курсор уводится прочь со ссылки или с меню, оно пропадает.

К разновидностям меню также можно отнести различные списки, в том числе раскрывающиеся, и вкладки.

ПРОСТОЕ МЕНЮ

Технически создание простого меню из одного уровня сводится всего к трем основным способам – с помощью ссылок, таблиц и с применением слоев (блоков `div` или `nav`).

Использование ссылок размещенных друг за другом в едином тэге (`div`, `p` или ином контейнере) не дает какого либо визуального выделения, поскольку тег для ссылок не является блочным. Такой способ построения меню можно использовать для организации меню очень простой структуры, например, одноуровневое горизонтальное меню.

Использование таблицы дает преимущества по сравнению с обычными ссылками и слоями. В частности, таблицы удобно использовать, менять параметры их отображения и выравнивать по любому краю содержимое ячеек. Также для меню важно то, заданное форматирование таблицы остается неизменным т.е. ячейки при уменьшении ширины окна браузера не смещаются с горизонтальной линии, а лишь уменьшаются в размерах. Самый простой вариант создания меню – когда в каждой ячейке располагается один пункт.

Меню, созданное с помощью слоев, по виду мало отличается от своего табличного варианта. И в том и другом случае можно активно применять стили для изменения оформления, а скорость загрузки фактически одинакова.

Горизонтальное меню

Горизонтальное меню является одним из распространенных и популярных элементов навигации, используемых на сайтах. Как следует из названия, пункты меню располагаются по горизонтали, как правило, в верхней части страницы. Перечислим следующие особенности, присущие горизонтальному меню:

- ширина веб-страницы ограничена разрешением монитора, его размерами, настройками браузера и операционной системы. По этой причине большое количество пунктов меню делать не рекомендуется. Иначе это может привести к появлению горизонтальной полосы прокрутки, что не позволяет удобно пользоваться сайтом, или стать причиной изменения вида и форматирования меню;
- горизонтальное меню располагают в верхней части веб-страницы, чтобы его можно было видеть без прокрутки содержимого. Иногда горизонтальное меню для удобства пользователей дублируют внизу страницы.

Рассмотрим создание меню на основе ссылок внутри тега `div`, для этого пропишем соответствующий HTML код на нашей странице:

```
<div id="menu">
  <a href="#">Главная</a>
  <a href="#">О нас</a>
  <a href="#">Контакты</a>
</div>
```

и зададим стиль CSS для блока `div` с идентификатором "menu" и ссылок расположенных внутри блока:

```
#menu{ width:200px; background:yellow; margin:auto;}
#menu a{ text-decoration:none; }
#menu a:hover{ color:white; background:blue; }
```

Получившееся меню будет выглядеть следующим образом (рис. 17)

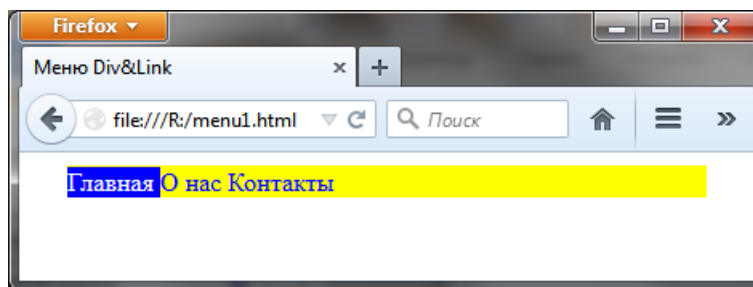


Рис. 17. Отображение меню

При выборе слоев для проектирования меню необходимо решить ряд задач – состыковка слоев друг с другом по горизонтали, выравнивание меню, изменение цвета фона при наведении курсора на ссылку и т. д. Рассмотрим выравнивание горизонтального меню, основанного на слоях. В отличие от тега TABLE, у которого есть удобный параметр align, выравнивание набора слоев происходит через стили. Первым шагом создания меню будет создание маркированного списка, при этом элемент списка будет содержать по одной ссылке. Поскольку необходимо будет иметь возможность идентифицировать список, то добавим к нему атрибут id с идентификатором "menu". Фрагмент HTML кода меню будет выглядеть следующим образом:

```
<div id="menu_back">
  <ul id="menu">
    <li><a href="#">Главная</a></li>
    <li><a href="#">О нас </a> </li>
    <li><a href="#">Контакты</a></li>
  </ul>
</div>
```

Рассмотрев пример можно обнаружить блок div с идентификатором "menu_back". Этот блок предназначен для генерации полосы, в которой будет расположено меню. Зададим настройку стилей следующим образом:

```
#menu_back {width: 90%; background:yellow; margin:auto;}
#menu {padding:0; list-style: none; width: 360px; }
#menu li { display: inline; text-align: center; }
#menu a {text-decoration: none; display: inline-block;
          width: 100px; padding: 2px;}
#menu a:hover {color:white; background:blue;}
```

В результате у нас получится меню как показано на рис. 18.

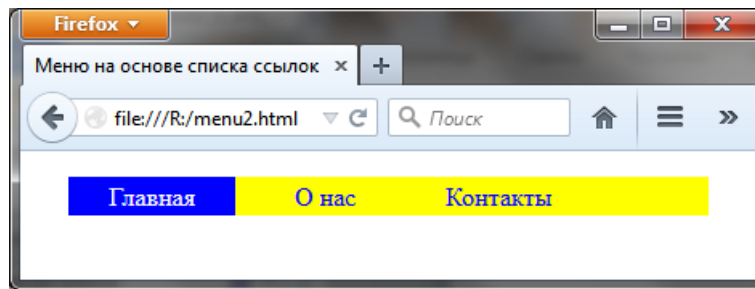


Рис. 18. Отображение меню

Как можно заметить для получения однотипно выглядящих меню приходится в зависимости от используемого подхода делать настройки стилей. При этом размер ширины для обычной ссылки, а также выравнивания для нее задать не получится, поскольку она не является блочным элементом.

Вертикальное меню

Для формирования вертикального меню на основе маркированного списка, содержащего ссылки, достаточно только изменить CSS стили соответствующим образом:

```
#menu_wrap {width: 104px; background:yellow; margin:0,auto;}
#menu {padding:0; list-style: none; }
#menu li { text-align: center; }
#menu a {text-decoration: none; display: block; width: 100px;
padding: 2px;}
#menu a:hover {color:white; background:blue;}
```

Результат применения стилей показан на рис. 19.

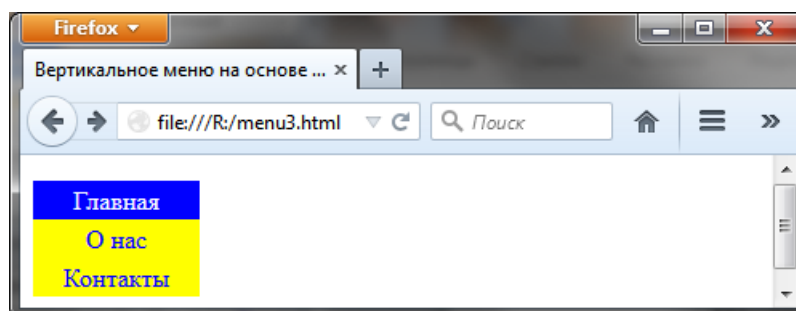


Рис. 19. Отображение вертикального меню

Если же будет принято решение о генерации меню на основе ссылок внутри тега div, то необходимо задать следующий CSS стиль для элементов:

```
#menu{ width:100px; background:yellow; text-align: center;
margin:0,auto;}
#menu a{ text-decoration:none; display:block; padding: 2px;}
#menu a:hover{ color:white; background:blue; }
```

С использованием таких стилей мы получим следующий результат (рис. 20).

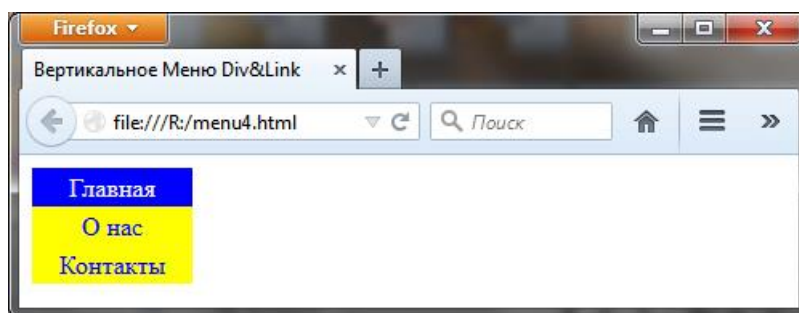


Рис. 20. Отображение вертикального меню

Сравнив два этих способа создания вертикального меню можно обнаружить, что результаты отображения в браузере идентичны. Теперь рассмотрим как создавать сложное меню, т.е. меню содержащее элементы подменю.

МЕНЮ СОДЕРЖАЩЕЕ ЭЛЕМЕНТЫ ПОДМЕНЮ

Рассмотрим пример формирования меню содержащее элементы подменю. Визуально меню будет иметь основные навигационные ссылки, расположенные в горизонтальной панели навигации, и подпункты, которые будут отображаться только после наведения курсора мыши на тот пункт меню, к которому эти подпункты относятся. Сначала необходимо создать HTML-структуру нашего меню. Основные навигационные ссылки мы поместим в маркированный список. Подпункты мы разместим в отдельном списке, вложив его в элемент `` содержащий родительскую ссылку относительно подпунктов. Теперь мы имеем четкую структуру нашей будущей панели навигации и ее HTML код будет выглядеть следующим образом:

```
<ul id="menu">
  <li><a href="#">Главная</a></li>
  <li><a href="#">О нас</a>
  <ul>
    <li><a href="#">Наши достижения</a></li>
    <li><a href="#">Фотогалерея</a></li>
  </ul></li>
  <li><a href="#">Контакты</a>
  <ul>
    <li><a href="#">Как к нам проехать</a></li>
    <li><a href="#">Обратная связь</a></li>
  </ul></li>
</ul>
</div>
```


Теперь приступим к написанию CSS кода. На первом этапе необходимо скрыть список с подпунктами, чтобы они не отображались на веб-странице все время. Для этих целей воспользуемся свойством **display** которому установим значение **none**. Чтобы отобразить подпункты при наведении на элемент `` необходимо чтобы список снова был преобразован в блочный элемент. Также убираем у обоих списков установленные по умолчанию отступы и маркеры:

```
#menu ul { display: none; }
#menu li:hover ul { display: block; }
#menu, #menu ul { margin: 0; padding: 0;
list-style-type: none; }
```

Элементы списка с навигационными ссылками делаем плавающими, формируя горизонтальное меню. При этом задаем атрибуту `float` значение `none`, для того чтобы элементы списка содержащие подпункты меню отображались друг под другом.

```
#menu li { float: left; }
#menu ul li { float: none; }
```

Необходимо сделать так, чтобы выпадающее подменю не смещало контент, расположенный под панелью навигации, вниз. Для этих целей зададим пунктам списка `position: relative;`, а списку, содержащему подпункты `position: absolute;` и добавим свойство `top` со значением `100%`, чтобы абсолютно позиционированное подменю отображалось точно под ссылкой.

```
#menu ul { display: none; position: absolute; top: 100%; }
#menu li { float: left; position: relative; }
#menu { height: 30px; }
```

Высота для родительского списка была добавлена в связи с тем, что плавающий контент в браузерах не учитывают в качестве содержимого элемента. Соответственно без добавления высоты список будет проигнорирован браузером и, следовательно, контент, следующий за списком, будет обтекать меню. Применяв только выше описанные стили сформированное меню будет выглядеть следующим образом (рис. 21).

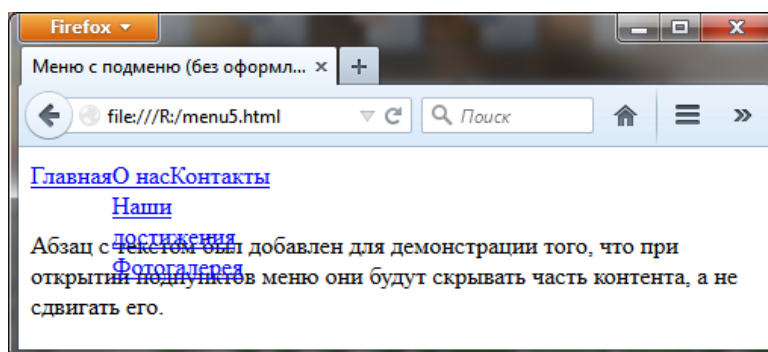


Рис. 21. Отображение многоуровневого меню

На изображении видно раскрытие пункта «О нас» при наведении мыши, а также расположенные поверх остального контента элементы подменю. Теперь остается настроить визуальное оформление списков меню. Итоговый код настройки стиля меню будет выглядеть следующим образом:

```
#menu ul { display: none; position: absolute;
           top: 100%; background-color: #f90; }
#menu li:hover ul { display: block; }
#menu, #menu ul { margin: 0; padding: 0;
                 list-style-type: none; }
#menu { height: 30px; background-color: #666;
        padding-left: 25px; min-width: 400px; }
#menu li { float: left; position: relative; height: 100%; }
#menu li a { display: block; padding: 6px; width: 130px;
            color: #fff; text-decoration: none;
            text-align: center; }
#menu ul li { float: none; }
#menu li:hover { background-color: #f90; }
#menu ul li:hover { background-color: green; }
```

В результате применения этих стилей формируется следующее меню (рис. 22).

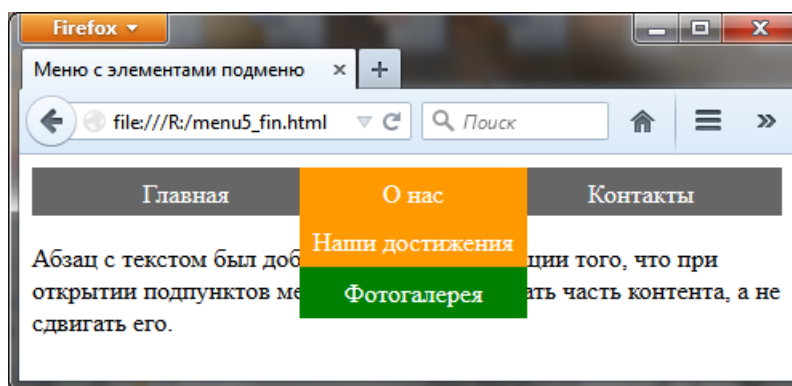


Рис. 22. Конечный вариант меню

JS

При генерации страниц в Web возникает дилемма, связанная с архитектурой "клиент-сервер". Страницы можно генерировать как на стороне клиента, так и на стороне сервера. В 1995 году специалисты компании Netscape создали механизм управления страницами на клиентской стороне, разработав язык программирования JavaScript.

JavaScript – это язык управления сценариями просмотра гипертекстовых страниц Web на стороне клиента. Его основная идея JavaScript заключается в возможности изменения значений атрибутов HTML-контейнеров и свойств среды отображения во время просмотра HTML-

документа пользователем. При этом перезагрузки страницы не происходит. Так, например, существует возможность изменить цвет фона страницы или интегрированную в документ картинку, открыть новое окно или выдать предупреждение. В языке javascript заложены следующие основные возможности:

- изменение страницы, путем добавления текста, добавления и удаления тегов, изменения стилей элементов.
- реакция на события, т.е. скрипт может ждать, когда что-нибудь случится (клик мыши, окончание загрузки страницы) и отреагировать на это выполнением функции.
- исполнение запросов к серверу и загрузка данных без перезагрузки страницы.
- установка и считывание cookie, а также валидация данных, вывод сообщений и многое другое.

Рассмотрим каким же образом происходит подключение и использование скриптов javascript.

СПОСОБЫ ПОДКЛЮЧЕНИЯ И ИСПОЛЬЗОВАНИЯ JS

Можно выделить три наиболее распространенных подхода к использованию скриптов: использование URL-схемы "JavaScript:", обработчики событий и вставка через контейнер <SCRIPT>. Разберем их основные особенности.

Первый подход который рассмотрим это использование URL-схемы "JavaScript:". Схема URL (Uniform Resource Locator) – это один из основных элементов Web-технологии. Так URL указывают в атрибуте **HREF** контейнера **A**, в атрибуте **src** контейнера **IMG** помимо этого URL также указывается в атрибуте **action** контейнера **FORM** и т.п. JavaScript позволяет поменять стандартную подгрузку URL на программу пользователя. Для этих целей разработчики языка ввели новую схему URL – "JavaScript:". Согласно схемы URL, обобщенный вызов JavaScript программы будет выглядеть следующим образом:

```
<A href="JavaScript:код_программы">...</A>  
<FORM action="JavaScript:код_программы" ...> ... </FORM>
```

В данном случае под текстом «код_программы» понимается некоторая программа-обработчик на языке JavaScript, которая будет вызвана при выборе гипертекстовой ссылки в первом случае и при отправке данных формы (нажатии кнопки Submit) – во втором.

Второй подход заключается в использовании обработчиков событий. При этом обработчики событий, указываются в атрибутах контейнеров, с которыми эти события связаны. Например, при нажатии на кнопку происходит событие Click и соответственно вызывается обработчик этого события onClick:

```
<FORM><INPUT TYPE=button VALUE="Кнопка"
onClick="alert('Вы нажали кнопку');"></FORM>
```

Помимо этого в момент завершения полной загрузки документа (он связан с контейнером `<BODY>`) происходит событие `Load` и, соответственно, будет вызван обработчик этого события `onLoad`:

```
<BODY onLoad="alert('Приветствуем!');">
...
</BODY>
```

В таблице 2 приведен перечень события возникающих при действиях пользователя с использованием мыши и клавиатуры.

Табл. 2. События Javascript

Событие	Описание
onClick	Происходит при щелчке одной из кнопок мыши в области документа
onKeyDown	Происходит при нажатии одной из клавиш клавиатуры
onKeyPressed	Происходит при нажатии и удерживании одной из клавиш клавиатуры
onKeyUp	Происходит, когда пользователь отпускает клавишу клавиатуры
onMouseDown	Происходит при нажатии одной из кнопок мыши
onMouseMove	Происходит при перемещении курсора мыши в области документа
onMouseOut	Происходит при выходе курсора мыши за область документа
onMouseOver	Происходит при входе курсора мыши в область документа
onMouseUp	Происходит, когда пользователь отпускает ранее нажатую кнопку мыши

Так событие `onMouseDown` возникает при нажатии левой кнопкой мыши. Рассмотрим пример, в котором при нажатии кнопки мыши на тексте меняется стиль оформления текста (текст выделяется жирным шрифтом). Для этого в теге **P** опишем событие:

```
<p onMouseDown="this.style.fontWeight='bold'"> Пример
использования события onMouseDown.</p>
```

Третий подход подразумевает использование вставки контейнера `<SCRIPT>`. При разборе документа HTML-парсер передает управление JavaScript-интерпретатору после того, как встретит тег начала контейнера `<SCRIPT>`. Интерпретатор получает на исполнение весь фрагмент кода внутри контейнера `SCRIPT` и возвращает управление HTML-парсеру для обработки текста страницы после тега конца контейнера `</SCRIPT>`.

Помещать JavaScript-код на HTML-странице с помощью контейнера `<SCRIPT>` можно двумя способами. Первый состоит в написании текста кода непосредственно внутри этого контейнера:

```
<SCRIPT> код_программы </SCRIPT>
```

Второй способ заключается в том, что код JavaScript выносится в отдельный файл, например, `myscript.js`, и затем включается в HTML-страницу следующим образом:

```
<SCRIPT src="myscript.js"></SCRIPT>
```

Этот способ удобен, когда один и тот же скрипт планируется использовать на разных HTML-страницах. Обратите внимание, что при наличии атрибута **src** содержимое контейнера `<SCRIPT>` пусто. Это сделано согласно спецификации HTML, в которой указано, что если скрипт подключается из внешнего файла, то скрипт, написанный между тэгами `<SCRIPT>` и `</SCRIPT>`, если таковой имеется, будет проигнорирован браузером.

ОБРАЩЕНИЕ К ЭЛЕМЕНТАМ DOM

Стандарт объектной модели документа (DOM) предусматривает несколько средств поиска элемента. Это методы `getElementById`, `getElementsByTagName` и `getElementsByName`.

Самый удобный способ найти элемент в DOM – это получить его по значению идентификатора (`id`). Для этого используется вызов метода **`document.getElementById(id)`**. Данный вызов возвращает элемент с заданным `id`, а если такой элемент не найден, то будет возвращено значение `null`. Рассмотрим простой следующий код, который изменит цвет текста на голубой в элементе `div` с `id="dataKeeper"`:

```
document.getElementById('dataKeeper').style.color = 'blue'
```

Следующий способ – это получение всех элементов с определенным тегом, и среди них выбора нужного. Для этих целей служит метод **`document.getElementsByTagName(tag)`**, который возвращает массив из элементов, имеющих указанный тег. Например, можно получить второй элементс тегом `li`:

```
document.getElementsByTagName('LI')[1]
```

Интересной особенностью является то, что **`getElementsByTagName`** можно вызывать не только для `document`, но и вообще для любого элемента, у которого есть тег (не текстового). При этом будут найдены только те объекты, которые находятся под этим элементом. Например, следующий вызов получает список элементов `LI`, находящихся внутри первого тега `div`:

```
document.getElementsByTagName('DIV')[0].  
  getElementsByTagName('LI')
```

При этом можно получить все элементы, передав звездочку `'*'` вместо тега:

```
var allElems = document.getElementsByTagName('*');
```

Также можно осуществлять поиск элемента по его имени. В таком случае используется метод **document.getElementsByName(name)**. Этот метод работает только с теми элементами, для которых в спецификации явно предусмотрен атрибут **name**: это form, input, a, select, textarea, iframe и ряд других, более редких. Метод **document.getElementsByName** не будет работать с остальными элементами типа div, p и т.п.

РАБОТА С COOKIE НА ПРИМЕРЕ

Cookie (Кúки) – это небольшой фрагмент данных, отправленный веб-сервером и хранимый на компьютере пользователя. В техническом плане куки представляют собой фрагменты данных, изначально отправляемые веб-сервером браузеру. При каждом последующем посещении сайта браузер пересылает их обратно серверу в составе HTTP-запроса. Cookie применяется для сохранения данных на стороне пользователя, на практике обычно используется для:

- аутентификации пользователя;
- хранения персональных предпочтений и настроек пользователя;
- отслеживания состояния сеанса доступа пользователя;
- ведения статистики о пользователях.

Для чтения и записи cookie используется свойство **document.cookie**. Однако, оно представляет собой не объект, а строку в специальном формате, для удобной манипуляций с которой нужны дополнительные функции. В **document.cookie** можно писать различные данные, при этом запись не перезаписывает существующие cookie, а дополняет к ним. Например воспользовавшись свойством name, такая строка поставит cookie с именем userName и значением Vasya:

```
document.cookie = "userName=Vasya";
```

Однако следует учитывать, что у cookie есть ряд важных настроек, которые очень желательно указать, так как значения по умолчанию у них не всегда могут устраивать. Эти настройки указываются после пары ключ=значение, каждое – после точки с запятой:

```
document.cookie = "name=значение; expires=дата; path=путь;  
domain=домен; secure";
```

Так одним из значимых параметров является **path**, определяющий путь, внутри которого будет доступ к cookie. В случае если его не указать, то имеется в виду текущий путь и все пути ниже него. Как правило, используется **path=/**, то есть cookie доступно со всех страниц сайта.

Свойство **expires** устанавливает дату истечения срока хранения куки. Дата должна быть представлена в формате, который возвращает метод `toGMTString()` объекта `Date`. Если значение `expires` не задано, куки будет удалено при закрытии браузера.

Опция **domain** устанавливает домен, в рамках которого действует куки. Получить значение куки могут только сайты из указанного домена. Обычно данное свойство оставляют пустым, что означает, что только домен установивший куки может получить доступ к нему.

Свойство `secure` указывает браузеру, что для пересылки куки на сервер следует использовать SSL. Очень редко используется.

Рассмотрим как можно использовать cookie для смены таблиц стилей у HTML страницы:

```
<head>
...
<link id="dyncss" rel="stylesheet" style="text/css"
      href="style1.css">
<script type="text/javascript">
function setDynCSS(url) {
    if (!arguments.length)
    {
        url = (url = document.cookie.match(/\\bdyncss=([^;]*)/))
            && url[1];
        if (!url) return '';
    }
    document.getElementById('dyncss').href = url;
    var d = new Date();
    d.setFullYear(d.getFullYear() + 1);
    document.cookie = ['dyncss=', url,
        ';expires=', d.toGMTString(), ';path=;'].join('');
    return url;
}
setDynCSS();
</script>
</head>
```

Пример использования:

```
<div>
    <span onclick=setDynCSS('style1.css')> Стил ь 1 </span>
    <span onclick=setDynCSS('style2.css')> Стил ь 2 </span>
</div>
```

Основы серверных технологий (PHP, CGI), модель MVC.

Язык PHP

PHP – это интерпретируемый язык программирования общего назначения с открытым исходным кодом. Он был сконструирован специально для ведения Web-разработок, основным преимуществом PHP-сценариев является возможность без проблем интегрироваться в обычные html-документы, при этом сохраняя всю функциональность, при условии конечно же наличия на web-сервере интерпретатора языка PHP. В настоящее время язык PHP поддерживается подавляющим большинством хостинг-провайдеров и является одним из лидеров среди языков программирования, применяющихся для создания динамических веб-сайтов.

Синтаксис PHP подобен синтаксису языка Си. Некоторые элементы, такие как ассоциативные массивы и цикл `foreach`, заимствованы из Perl. Для исполнения программы не требуется описывать какие-либо переменные, используемые модули и т. п. Любая программа может начинаться непосредственно с оператора PHP. В общем виде код интеграция PHP кода в HTML документ выглядит следующим образом:

```
<?php
    Код PHP программы
?>
```

PHP: ПОДКЛЮЧЕНИЕ ФАЙЛОВ

Одна из полезных возможностей `php` – подключение другого файла. Например, на сайте из 10 различных страниц и при этом есть элементы, присутствующие на этих страницах, так это может быть заголовочная часть, меню или иные блоки. В какой-то момент в меню потребовалось внести изменения. Если бы использовался только **HTML**-код то пришлось бы вручную в каждом отдельном файле вносить изменения. PHP позволяет существенно упростить работу с сайтом. Необходимые элементы размещаются в отдельном файле, а на каждой из 10-ти страниц происходит подключение нужных файлов. Таким образом, все изменения нужно будет вносить только в файл с меню, а на 10-ти других оно будет отображаться уже с изменениями.

Помимо этого, при реализации внутреннего функционала, который не видим пользователю, чтобы сделать программный код более удобочитаемым, зачастую приходится помещать определения функций и/или классов в отдельные файлы исходя из их функционального назначения.

Возможность подключения файлов в PHP обеспечивают четыре языковые инструкции:

- `include`
- `require`

- include_once
- require_once

Эти инструкции принимают в качестве параметра имя подключаемого php файла. Рассмотрим эти инструкции более подробней. Инструкция **include** позволяет подключать и присоединять к PHP-сценарию другие сценарии. Так, при запуске программы интерпретатор просто заменит инструкцию на содержимое подключаемого файла. Рассмотрим, как это работает на небольшом примере, для этого создадим файл с именем vars.php и пропишем внутри следующий код:

```
<?php
    $var1 = 'Первая переменная';
    $var2 = 'Значение второй переменной';
?>
```

Теперь создадим другой файл и назовем его test.php, в нем будет выполняться подключение файла vars.php. При формировании кода предполагается, что файл vars.php находится в той же директории что и файл test.php, поэтому при подключении будет использоваться только имя файла, а не полный путь:

```
<?php
    include 'vars.php';
    echo "\$var1 = $var1<br> \$var2 = $var2"
?>
```

Как видно из примера инструкция **include** присоединяет содержимое подключаемого файла, благодаря чему программа получает доступ к переменным, функциям, классам и иным объектам, находящимся в подключаемом файле.

Важно помнить, что если подключение файла происходит внутри тела функции, то тогда весь код, содержащийся во включаемом файле, будет вести себя так, как будто он был определен внутри этой функции. Фактически этот код будет иметь локальную область видимости.

Теперь рассмотрим инструкцию **include_once**. Она работает абсолютно также как и **include**, с той лишь разницей, что если код из файла уже был один раз подключен, он не будет подключен и выполнен повторно. Это помогает избежать проблемы с переопределением функций, переменных и т.д.

Инструкции **require** и **require_once** работают идентично **include** и **include_once** за исключением лишь одной особенности – ответной реакцией на невозможность получения запрошенного ресурса. Так если подключаемый файл не будет найден, выполнение скрипта будет остановлено, в то время как **include** и **include_once** выведут предупреждение и продолжат выполнение скрипта.

PHP: РАБОТА POST и GET

При разработке любого проекта, одной из самых основных вещей является общение с пользователем. Можно что-то у него спрашивать, и давать ему право на ответ (опрос), также возможно предоставлять пользователям право написать свое мнение, о товаре или услуге.

Во всех случаях у пользователя должно быть право что-то написать и на что-то нажать. Для этого и существуют формы.

Передача переменных при помощи метода GET

Данный метод передачи переменных применяется в PHP для передачи переменных в файл при помощи адресной строки. То есть переменные передаются сразу через адресную строку браузера. Примером может быть, например, ссылка на статью, которая имеет примерно следующий вид:

```
http://www.mysite.ru/sample.php?p=315
```

В данном случае передается переменная \$p со значением 315. Теперь более подробно на примере рассмотрим работу метод GET. Пускай нам нужно передать три переменных \$a, \$b и \$c **методом GET** и вывести их сумму на экран. Для этого можно использовать следующий код:

```
$a = $_GET['a'];  
$b = $_GET['b'];  
$c = $_GET['c'];  
$summa = $a + $b + $c;  
echo "Сумма $a + $b + $c = $summa";
```

Поскольку все переменные перед передачей будут помещаться в глобальный массив GET, то первым делом присваиваем нашим переменным значения соответствующих элементов массива GET. Выполняется это в самом начале во избежание разнообразных ошибок при передаче переменных. Далее для демонстрации работы прописываем произвольную формулу и выводим результат на экран.

Для проверки работы метода GET достаточно просто добавить к ссылке на файл знак вопроса «?» и через амперсанд «&» перечислить переменные с их значениями:

```
http://www.mysite.ru/sample.php?a=1&b=2&c=3
```

Как видно из примера, сначала добавляем знак вопроса сразу после названия файла. Далее прописываем переменную и через знак равно указываем ее значение. После этого через амперсанд аналогичным образом перечисляем другие переменные. Теперь при переходе по этой ссылке на экран будет выведена сумма переменных \$a, \$b и \$c.

Данный способ очень простой и не требует создания дополнительных файлов. Все необходимые данные поступают прямо через адресную строку браузера.

Теперь перейдем ко второму способу передачи переменных в PHP – методу POST.

Передача переменных в PHP при помощи метода POST

Данный способ позволяет скрыто передавать переменные с одного файла в другой. Рассмотрим простой пример – имеется два файла, в первом находится форма для ввода исходных данных, а во втором исполняемый код обработки, который принимает переменные.

Код первого файла с формой для отправки данных. Дадим ему название sample_post-1.php:

```
<!--Форма -->
<form action="post_processor.php" method="post" name="form1"
      target="_blank">
<!--Текстовые поля -->
  <p><input name="name" type="text" size="20"></p>
  <p><input name="lastname" type="text" size="20"></p>
<!--Кнопка -->
  <p><input name="submit" type="submit" value="Передать"></p>
</form>
```

Код второго файла, который будет служить приемником переменных (назовем его post_processor.php):

```
$name = $_POST[name];
$lastname = $_POST[lastname];
echo "Значения переменных, переданных методом POST - $name и
$lastname";
```

Как и с методом **GET**, первым делом присваиваем переменным значения соответствующих элементов глобального массива POST. Далее для наглядности выводим эти переменные на экран при помощи оператора вывода echo.

Теперь при загрузке первого файла в браузере будет отображена форма. После ввода данных и нажатия по кнопке «Отправить» будет открыта страница со вторым файлом, на которой выведутся значения, прописанные в форме на предыдущей странице. То есть второму файлу будут переданы значения переменных с первого файла.

Сравнение методов GET и POST

При использовании метода GET данные передаются путем добавления к URL-адресу. Таким образом, они будут видны пользователю, что с точки зрения безопасности не всегда хорошо. Также максимальный объем передаваемых данных будет зависеть от браузера – от максимально-допустимого количества символов адресной строке браузера.

При использовании метода POST данные не будут видны пользователю (не отображаются в адресной строке браузера). И поэтому они более защищены, а, следовательно, и программа обрабатывающая эти данные более защищена в плане безопасности. Также объем передаваемых данных практически ни чем не ограничен.

Выбирая способ передачи данных нужно учитывать приведенные особенности и останавливаться на наиболее приемлемом методе.

PHP: РАБОТА С БАЗАМИ ДАННЫХ

Чтобы получить доступ к базе данных из Web, используя PHP, надо выполнить следующие основные шаги:

1. Подключение к серверу с СУБД.
2. Выбор базы данных.
3. Выполнение запроса к базе данных: выборка; добавление; изменение; удаление.
4. Получение результата запроса.
5. Отсоединение от базы данных.

Рассмотрим каким образом можно работать с базой данных на примере подключения к СУБД MySQL. Для соединения с базой данных MySQL имеется три разных API:

- mysql
- mysqli – MySQL Improved (улучшенная)
- pdo – PHP Data Objects (объекты данных PHP)

MySQL

Традиционный API mysql хорошо выполняет свою работу и получил широкое распространение благодаря тому, что делает процесс получения записей из базы данных очень простым.

Так для организации подключения к серверу базы данных используется функция `mysql_connect`. В общем виде ее синтаксис выглядит следующим образом:

```
mysql_connect(HostName, UserName, Password));
```

В качестве аргументов здесь выступают имя компьютера (сервера), имя пользователя и пароль. В качестве имени может выступать IP адрес

компьютера. Функция **mysql_connect** возвращают идентификатор подключения, если все прошло успешно. Например:

```
$link = mysql_connect(HostName, UserName, Password);  
if ( !$link ) die ("Невозможно подключение к MySQL");
```

Функция **mysql_connect** устанавливает обычное соединение с сервером MySQL. Обычное соединение закрывается, когда сценарий завершает работу или когда вызывается функция **mysql_close**:

После того, как соединение с сервером MySQL установлено, следующим этапом нужно выбрать базу данных. Для этого используется функция **mysql_select_db**. Ее аргумент: имя базы данных. Функция возвращает **true**, если указанная база данных существует и доступ к ней возможен. Например:

```
$db = "sample";  
mysql_select_db($db) or die ("Невозможно открыть $db");
```

Для добавления, удаления, изменения и выбора данных нужно сконструировать и выполнить запрос SQL.

Выполнить запрос к базе данных позволяет функция **mysql_query**, которой в качестве аргумента передается строка с запросом.

```
int mysql_query(string $query [,int $link_identifier])
```

Эта функция в своем роде универсальна: она посылает MySQL-серверу запрос \$query и возвращает идентификатор ответа, или результата. Поскольку в языке SQL есть команды, возвращают только признак, успешно они выполнились или нет (например, это команды UPDATE, INSERT и т. д.). В таком случае этот признак и будет возвращен функцией. В случае выборки данных (например, запрос SELECT) возвращается как раз идентификатор вывода, нулевое значение которого свидетельствует о том, что произошла ошибка. Параметр \$query представляет собой строку, составленную по правилам языка SQL. Используется установленное ранее соединение \$link_identifier, а в случае его отсутствия – последнее открытое соединение.

Для того чтобы записи возвращались в нужной кодировке, следует после выбора базы данных указать один из запросов:

```
mysql_query('SET NAMES cp1251'); // Для кодировки Windows-1251  
mysql_query('SET NAMES utf8'); // Для кодировки UTF-8
```

Поскольку сама функция возвращает не что иное, как идентификатор (id) результирующего набора данных. То есть все, что было выбрано посредством запроса – она оставляет у себя, а передает только id этих данных. Непосредственно для обращения к самим данным используются уже

другие функции. Например, чтобы выбрать первую строку результата из таблицы mytable можно сделать так:

```
<?
$result = mysql_query("select * from mytable");
$row = mysql_fetch_row($result);
?>
```

После этого в \$row будет сорежаться строка в виде массива. Есть несколько функций для выборки строки из результата: **mysql_fetch_row**, **mysql_fetch_array**, **mysql_fetch_assoc**. Единственное чем они отличаются — это видом массива, который они возвращают. Например **mysql_fetch_array** может вернуть как обычный, так и ассоциативный массив.

Следует отметить, что после каждой выборки строки внутренний указатель переводится на следующую строку.

MySQLi

Библиотека php_mysqli.dll предоставляет более современные методы доступа к базе данных MySQL и позволяет получить доступ к функциональности, которая имеется в MySQL версии 4.1 и выше. Библиотека предоставляет как процедурный стиль доступа, так и объектный. Параллельно будем рассматривать два стиля доступа. Так для организации подключения к серверу базы данных можно воспользоваться двумя способами:

```
$db = mysqli_connect(<HostName, UserName, Password, DataBase);
$db = new mysqli(HostName, UserName, Password, DataBase);
```

Все параметры являются необязательными. Процедурный стиль возвращает идентификатор соединения, а в случае неудачи возвращает false.

Проверить соединение можно следующим образом:

```
if (@$db = mysqli_connect("localhost", "root", "123", "test"))
{
    // Выполняем работу с базой данных
}
else {
    echo "Не удалось установить подключение к базе данных";
}
```

При объектном стиле такой способ проверки не подходит. Для проверки отсутствие ошибок при подключении применяется функция **mysqli_connect_errno**. Рассмотрим код проверки соединения с базой данных:

```
@$db = new mysqli("localhost", "root", "123", "test");
if (!mysqli_connect_errno()) {
    // Выполняем работу с базой данных
}
else {
    echo "Не удалось установить подключение к базе данных";
}
```

Для закрытия соединения при процедурном стиле используется функция **mysqli_close()**:

```
mysqli_close(<Идентификатор>);
```

В случае объектного стиля используется метод **close()**:

```
<Экземпляр класса>->close();
```

Выбрать базу данных можно при подключении в функции **mysqli_connect()** или в конструкторе класса. Для выбора базы данных уже после подключения при процедурном стиле используется функция **mysqli_select_db()**. Функция имеет следующий формат:

```
mysqli_select_db(<Идентификатор>, <Имя базы данных>);
```

При объектном стиле используется метод **select_db()**. Метод имеет формат:

```
<Экземпляр класса>->select_db(<Имя базы данных>);
```

Выполнить запрос к базе данных при процедурном стиле позволяет функция **mysqli_query()**. Данная функция возвращает идентификатор результата. Она имеет следующий формат:

```
mysqli_query(<Идентификатор>, <SQL-запрос>);
```

Для удаления идентификатора результата и освобождения используемых ресурсов применяется функция **mysqli_free_result()**, которая имеет следующий формат записи:

```
mysqli_free_result(<Идентификатор результата>);
```

Выполнить же запрос к базе данных при объектном стиле позволяет метод **query()**, обладающий следующим форматом записи:

```
<Экземпляр класса>->query(<SQL-запрос>);
```

Данный метод возвращает экземпляр результата, для удаления которого применяется метод **close()**. Метод имеет формат:

```
<Экземпляр результата>->close();
```

Для того чтобы записи возвращались в нужной кодировке, следует после подключения выполнить запрос

```
mysqli_query($db, 'SET NAMES cp1251'); // windows-1251  
mysqli_query($db, 'SET NAMES utf8'); // UTF-8
```

при процедурном стиле или

```
$db->query('SET NAMES cp1251'); // windows-1251  
$db->query('SET NAMES utf8'); // UTF-8
```

при объектном стиле.

СУБД MySQL поддерживает подготавливаемые запросы. Подготавливаемые (или параметризованные) запросы используются для повышения эффективности, когда один запрос выполняется многократно. Так под подготовленными запросами понимается некий вид скомпилированного шаблона SQL запроса, который будет запускаться приложением и настраиваться с помощью входных параметров. Подготовленные запросы могут способствовать повышению безопасности путём отделения логики SQL запроса, от данных, подставляемых в него. Такое разделение логики и данных может помочь предотвратить внедрение SQL-инъекций.

Выполнение подготавливаемого запроса проводится в два этапа: подготовка и исполнение. На этапе подготовки на сервер посылается шаблон запроса. Сервер выполняет синтаксическую проверку этого шаблона, строит план выполнения запроса и выделяет под него ресурсы. MySQL сервер поддерживает неименованные, или позиционные, псевдопеременные (?). Для подготовки запроса используется процедура **mysqli_prepare()** и метод **prepare()** при использовании объектно-ориентированный стиля. В результате выполнения процедура **mysqli_prepare()** возвращает указатель на выражение, которое может использоваться для дальнейших операций с ним. При этом запрос должен состоять из одного SQL выражения.

После подготовки запроса ему необходимо передать значения, для этого используется привязка параметров методом **bind_param()** или процедурой **mysqli_stmt_bind_param()**. Привязанным параметрам задаются желаемые значения.

Рассмотрим использование подготовленного запроса на примере отображения сведений о сотрудниках фирм в зависимости от того в каком городе находится фирма и какой деятельностью она занимается. В процедурном стиле код будет выглядеть следующим образом:

```
if ($stmt = mysqli_prepare($link, "SELECT Name, Department,  
    FirmName, FROM Workers WHERE CityABR=? And FirmType=?")) {  
    mysqli_stmt_bind_param($stmt, "ss", $city, $ftype);  
    $city='KRS'  
    $ftype ='Media'  
    mysqli_stmt_execute($stmt);
```



```

        mysqli_stmt_bind_result($col1, $col2, $col3);
        while (mysqli_stmt_fetch($stmt)) {
            printf("%s %s %s\n", $col1, $col2, $col3);
        }
        mysqli_stmt_close($stmt);
    }
}

```

Теперь рассмотрим код для объектно-ориентированного:

```

$stmt = $mysqli->prepare("SELECT Name, Department,
    FirmName, FROM Workers WHERE CityABR=? And FirmType=?");
$stmt->bind_param('ss', $city, $ftype);
$city='KRS'
$ftype = 'Media'
$stmt->execute();
$stmt->bind_result($col1, $col2, $col3);
while ($stmt->fetch()) {
    printf("%s %s %s\n", $col1, $col2, $col3);
}
$stmt->close();

```

PDO

PHP Data Objects (PDO) – расширение для PHP, предоставляющее разработчику простой и универсальный интерфейс для доступа к различным базам данных. Каждый драйвер базы данных, в котором реализован этот интерфейс, может представить специфичный для базы данных функционал в виде стандартных функций расширения.

Следует отметить, что само по себе расширение PDO не позволяет манипулировать доступом к базе данных. Чтобы воспользоваться возможностями PDO, необходимо использовать соответствующий конкретной базе данных PDO драйвер. Увидеть список доступных драйверов можно используя php код:

```
print_r(PDO::getAvailableDrivers());
```

Соединения с базой данных автоматически устанавливаются при создании объекта PDO от его базового класса. При этом не имеет значения, какой драйвер будет использован. Конструктор класса принимает аргументы для задания источника данных (DSN), а также необязательные имя пользователя и пароль (если есть). Для СУБД MySQL подключение будет выглядеть следующим образом:

```

$dbh = new PDO('mysql:host=localhost;dbname=test', $user,
    $pass);

```

В случае ошибки при подключении будет выброшено исключение **PDOException**. Его можно перехватить и обработать используя следующую конструкцию:

```
try {
    $dbh = new PDO('mysql:host=localhost;dbname=test', $user,
        $pass);
    // Выполняем работу с базой данных
} catch (PDOException $e) {
    print "Ошибка! : " . $e->getMessage() . "<br/>";
    die();
}
```

При успешном подключении к базе данных в скрипт будет возвращен созданный PDO объект. Соединение будет оставаться активным на протяжении всего времени жизни объекта. Для закрытия соединения необходимо уничтожить объект путем удаления всех ссылок на него. Это можно сделать, присвоив значение NULL всем переменным, указывающим на объект. Если не сделать этого явно, PHP автоматически закроет соединение по окончании работы скрипта.

Иногда может оказаться полезным использование постоянных соединений к базам данных. Постоянные соединения не закрываются при завершении работы скрипта, они кэшируются и используются повторно, когда другой скрипт запрашивает соединение с теми же учетными данными. Постоянные соединения позволяют избежать создания новых подключений каждый раз, когда требуется обмен данными с базой, что в результате дает прирост скорости работы таких приложений. Пример создания постоянного соединения:

```
$dbh = new PDO('mysql:host=localhost;dbname=test', $user,
    $pass, array( PDO::ATTR_PERSISTENT => true ));
```

Для выполнения запросов можно пользоваться двумя методами. Если в запрос не передаются никакие переменные, то можно воспользоваться функцией **query()**. Она выполнит запрос и вернёт объект PDO statement. Получить данные из этого объекта можно как традиционным образом, через **while**, так и через **foreach()**, например:

```
$stmt = $pdo->query('SELECT name FROM users');
while ($row = $stmt->fetch())
{
    echo $row['name'] . "\n";
}
```

Если же в запрос передаётся хотя бы одна переменная, то этот запрос рекомендуется выполнять через подготовленные выражения.

PDO поддерживает позиционные псевдопеременные (?), для которых важен порядок передаваемых переменных, и именованные (:name), для которых порядок не важен. Примеры:

```
$sql = 'SELECT name FROM users WHERE email = ?';  
$sql = 'SELECT name FROM users WHERE email = :email';
```

Чтобы выполнить такой запрос, сначала его надо подготовить с помощью функции **prepare()**, которая возвращает PDO statement, но ещё без данных. Чтобы их получить, надо исполнить этот запрос, предварительно передав в него переменные. Передать можно двумя способами:

Чаще всего можно просто выполнить метод **execute()**, передав ему массив с переменными:

```
$stmt = prepare('SELECT name FROM users WHERE email = ?')  
$stmt->execute(array($email));  
  
$stmt = prepare('SELECT name FROM users WHERE email =  
:email');  
$stmt->execute(array('email' => $email));
```

Как видно, в случае именованных псевдопеременных в **execute()** должен передаваться массив, у которого ключи должны совпадать с именами псевдопеременных.

Иногда, может потребоваться второй способ, когда переменные сначала привязывают к запросу по одной, с помощью **bindValue()** / **bindParam()**, а потом только исполняют. В этом случае в **execute()** ничего не передается. Сначала рассмотрим выполнение подготовленного запроса с именованными псевдопеременными:

```
$calories = 150;  
$colour = 'red';  
$sth = $dbh->prepare('SELECT name, colour, calories  
FROM fruit  
WHERE calories < :calories AND colour = :colour');  
$sth->bindValue(':calories', $calories, PDO::PARAM_INT);  
$sth->bindValue(':colour', $colour, PDO::PARAM_STR);  
$sth->execute();
```

В случае выполнения того же самого подготовленного запроса, но с неименованными псевдопеременными код будет выглядеть следующим образом:

```
$calories = 150;  
$colour = 'red';  
$sth = $dbh->prepare('SELECT name, colour, calories  
FROM fruit
```

```
WHERE calories < ? AND colour = ?');
$sth->bindValue(1, $calories, PDO::PARAM_INT);
$sth->bindValue(2, $colour, PDO::PARAM_STR);
$sth->execute();
```

Сравнение(особенности) MySQLi и PDO

Как PDO так MySQLi предлагают объектно-ориентированное API. Но в тоже время MySQLi имеет процедурную часть API, которая для новичков может показаться проще для освоения. При наличии модулей работающих с собственным драйвером PHP MySQL, мигрирование на процедурную часть MySQLi будет достаточно простой задачей.

Ключевым преимуществом PDO перед MySQLi является поддержка различных баз данных, в то время как MySQLi поддерживает только **MySQL**. На момент учебного пособия **PDO** может использовать 12 драйверов.

Если рассматривать библиотеки по предоставляемым средствам защиты от SQL инъекций, которые доступны разработчикам то они схожи. Так, допустим, злодей пытается встроить вредный запрос SQL через параметр 'username' HTTP запроса (GET):

```
$_GET['username'] = "'; DELETE FROM users; /*"
```

Поскольку MySQLi и PDO поддерживают множественные запросы то если пропустить такое выражение, то оно будет включено в запрос в том виде как есть. Соответственно выполняя запрос:

```
query("SELECT * FROM users WHERE username = $username");
```

будут удалены все строки из таблицы **users**. Для обеспечения защиты MySQLi и PDO предлагают методы для «ручной» зачистки параметра. Посмотрим на примере как это происходит для MySQLi:

```
$username = mysqli_real_escape_string($_GET['username']);
$query = "SELECT * FROM users WHERE username = '$username'";
```

Функция **mysqli_real_escape_string** только отбрасывает лишние символы в строке. В случае использования PDO используется метод **PDO::quote**, который не только отбрасывает лишние символы в строке, но и заключает ее в кавычки. Пример для PDO:

```
$username = PDO::quote($_GET['username']);
$query = "SELECT * FROM users WHERE username = $username";
```

Другими способами защиты являются использование подготовленных запросов и хранимых процедур.

Если проводить сравнение по скорости выполнения запросов то MySQLi имеет небольшое преимущество по результатам тестов на начало 2014 года. Так это преимущество составляет ~2.5% для обычных запросов и ~6.5% для подготовленных выражений. Так что стоит принять информацию во внимание, если будет важно даже минимальное различие в характеристиках при выборе способов работы с базами на СУБД MySQL.

MVC

Шаблон MVC (Model-View-Controller) описывает способ построения структуры приложения, целью которого является отделение бизнес-логики от пользовательского интерфейса. Это дает ряд преимуществ – приложение легче масштабируется, тестируется, сопровождается и реализуется. Рассмотрим концептуальную схему шаблона MVC (рис. 22)

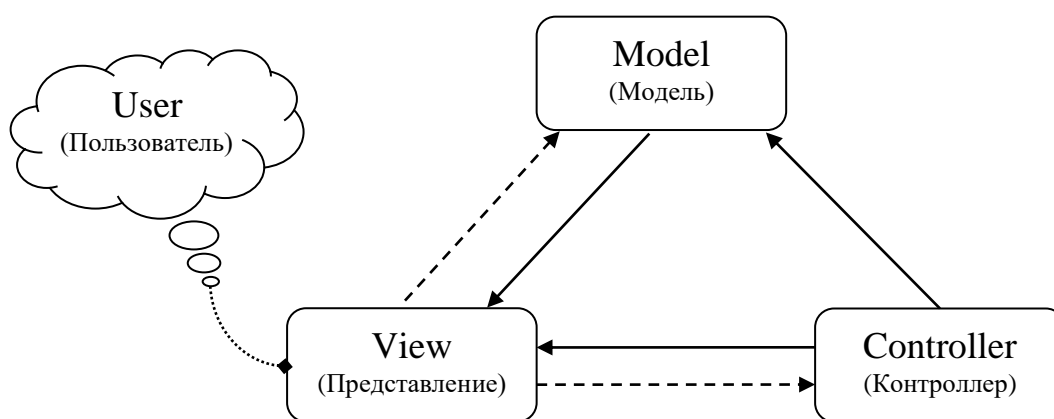


Рис. 22. Модель MVC

В рамках разработки веб приложений в архитектуре MVC модель предоставляет данные и правила бизнес-логики, представление отвечает за пользовательский интерфейс, а контроллер обеспечивает взаимодействие между моделью и представлением. Рассмотрим эти компоненты более подробно.

Модель (Model) – содержит бизнес-логику приложения и включает методы выборки (это могут быть методы ORM), обработки (например, правила валидации) и предоставления конкретных данных. При этом сама модель не взаимодействует с пользователем напрямую. Все переменные, относящиеся к запросу пользователя обрабатываются в контроллере. Также модель не должна генерировать HTML или другой код отображения, который может изменяться в зависимости от нужд пользователя. Такой код должен обрабатываться в представлениях (view).

При реализации одна и та же модель, например: модель авторизации пользователей может использоваться как в пользовательской, так и в административной части приложения. В таком случае целесообразно выне-

сение общего кода в отдельный класс и для подприложений использовать производные классы содержащие специфичные методы.

Представления (view) или по другому «вид» – используются для задания внешнего отображения данных, полученных из контроллера и модели. Представления обычно содержат HTML-разметку и небольшие вставки PHP-кода для обхода, форматирования и отображения данных. При этом представления не должны напрямую обращаться к базе данных. Этим должны заниматься модели. Также представления не должны работать с данными, полученными из запроса пользователя. Эту задачу должен выполнять контроллер. Однако представления могут напрямую обращаться к свойствам и методам контроллера или моделей, для получения готовых к выводу данных.

Представления обычно разделяют на общий шаблон, и части шаблона, которые используют для отображения данных выводимых из модели или отображения форм ввода данных. В общем шаблоне содержится разметка которая будет характерна для всех страниц (например, шапку и подвал и т.п.)

Контроллер (Controller) является связующим звеном, которое соединяет модели, представления и иные компоненты в единое приложение. Контроллер отвечает за обработку запросов пользователя. При этом контроллер не должен содержать SQL-запросов, HTML-кода и другой разметки. В хорошо спроектированном MVC-приложении контроллеры обычно очень тонкие и содержат только несколько десятков строк кода в отличие от моделей, которые содержат большую часть кода, связанную с обработкой данных. Логика контроллера довольно типична и поэтому большая ее часть выносится в базовые классы.

Типичную последовательность работы MVC-приложения при выполнении отображения данных согласно запросу на выборку можно описать следующим образом. При первом заходе на веб-ресурс скрипт инициализации создает экземпляр приложения и запускает его на выполнение. При этом отображается представление (view), например, главной страницы сайта. После того как пользователь указывает в определенной форме критерии отбора данных и нажимает на кнопку поиска. Браузер формирует запрос приложению, при получении которого приложение определяют запрошенные контроллер и действие (например, выбор товаров определенного типа). После этого приложение создает экземпляр контроллера и запускает необходимый метод, в котором, к примеру, содержатся вызовы модели, считывающие информацию из базы данных. После этого формируется представление с данными, полученными из модели и выводит результат пользователю.

Основы форматов передачи данных (XML, CML, JSON)

XML и его производные

XML (Extensible Markup Language) представляет собой расширяемый язык разметки. Язык называется расширяемым, поскольку он не фиксирует разметку, используемую в документах. Соответственно разработчику предоставляется возможность создавать разметку в соответствии с потребностями к конкретной области, будучи ограниченным лишь синтаксическими правилами языка. Спецификация XML описывает XML-документы и частично описывает поведение XML-процессоров.

XML-файлы и файлы других расширений, основанные на языке XML, получили очень широкое распространение. В XML-файлах хранятся самые различные данные – от настроек приложений до баз данных. Файлы на основе XML используются для обмена информацией в Интернете и между программами. Так, например, начиная с выпуска версии 2007, в системах Microsoft Office используются форматы файлов на основе XML, такие как DOCX, XLSX и PPTX. В сфере коммерции большинством Российских компаний используется единый стандарт обмена коммерческой информацией в формате XML – CommerceML. Так, этим стандартом предусматривается использование схем XML, в частности, для обмена:

- каталогами товаров в системах управления каталогом;
- коммерческими предложениями (заказами);
- документами;

СТРУКТУРА ФАЙЛОВ ФОРМАТА XML

Поскольку файлы формата XML содержат текстовые данные, их можно легко отредактировать в любом текстовом редакторе. Большая часть текста помещается в элементы, в которых текст окружен тегами. Структура XML-документа представляет собой дерево элементов.

Простейший XML-документ состоит из двух частей: пролога и элемента Документ (его также называют корневым элементом).

Пролог XML

Пролог содержит объявление XML, указывающее на то, что это XML-документ, и содержит номер версии XML. Пролог может также содержать следующие необязательные компоненты:

- объявление типа документа, определяющее тип и структуру документа, которое должно следовать после XML-объявления;
- одну или несколько инструкций по обработке, содержащих информацию о порядке проходов при обработке приложения XML-процессором.

Рассмотрим на примере, как выглядит пролог:

```
<?xml version="1.0"?>
```

```
<!-- File Name: products.xml -->
```

В данном примере первая строка представляет собой объявление XML, указывающее на то, что это XML-документ и содержащее номер версии. Объявление XML не является обязательным, хотя раздел 2.8 рекомендации W3C по XML 1.0 требует его включения. Обобщенный синтаксис объявления XML выглядит следующим образом:

```
<?xml version [encoding [standalone]]?>
```

Рассмотрим параметры присутствующие в объявлении:

version – указывает номер версии XML, так согласно стандарта XML 1.1, объявление версии документа является обязательным, при отсутствии объявления версии XML-документ будет восприниматься как документом XML 1.0.

encoding – задает кодировку. Основной кодировкой XML является Unicode. Если кодировка не задана, обработчик XML подразумевает, что используется кодировка UTF-8 или UTF-16 в зависимости от наличия или отсутствия в самом начале обрабатываемого файла особой последовательности байт (называемой Byte Order Mark или BOM). Для использования в XML-документе символов кириллицы (как в символьных данных, так и в именах элементов), возможно указать кодировку windows-1251 в объявление XML:

```
<?xml version="1.0" encoding="windows-1251"?>
```

Такая форма (с указанием версии XML и кодировки) является минимальным объявлением XML, настоятельно рекомендованным к использованию во всех XML-файлах.

standalone – автономность XML. Этот параметр объявления указывает, содержит ли внешнее подмножество DTD какие-либо объявления, которые могут повлиять на текущее содержимое документа. Если в объявлении автономности задано значение "yes", например:

```
<?xml version="1.0" standalone="yes"?>
```

и в документе есть ссылки на внешнее определение DTD или внешние сущности, то средство синтаксического анализа сообщит об ошибке. В случае если параметр автономности опущен, то тогда будет считаться что ему присвоено значение "no".

Вторая строка пролога из рассматриваемого выше примера состоит из пробела. С целью улучшения внешнего вида документа возможно

вставлять любое количество пустых строк между элементами пролога. При обработке они будут игнорироваться.

Третья строка пролога представляет собой комментарий. Включение комментариев в XML-документ не является обязательным, но позволяет сделать его более понятным. Комментарий начинается с символов `<!--` и заканчивается символами `-->`. Между этими двумя группами символов возможно размещать любой текст, за исключением двойного тире (`--`).

Тело документа XML

Тело документа может состоять из следующих конструкций:

- элементов, которые могут включать атрибуты и пространства имен;
- комментариев;
- инструкций обработки;
- секций CDATA.

Элемент – это основная часть XML документа. Границы элемента задаются двумя способами:

- начальным и конечным тегом (текст между тегами называется содержанием элемента), например:

```
<goods>Телефон Samsung Galaxy S3</goods>
```

- тегом пустого элемента (пустой элемент не имеет содержания), например:

```
<description />
```

При этом вместо пустого элемента можно использовать эквивалентную ему запись начального и конечного тега:

```
<description></description>
```

Элемент в XML документе может иметь любое количество атрибутов (или не иметь вообще). Атрибуты элемента предназначены для уточнения каких-либо характеристик элемента. Они могут использоваться либо в начальном теге, либо в теге пустого элемента, например:

```
<goods id="art582" year="2013">  
  <title>Sony Xperia ZR</title>  
</goods>
```

В представленном примере для элемента `goods` определены два атрибута: `id` – артикул прайс листа и год поступления в продажу.

Инструкция по обработке имеет следующую форму записи:

<? кому инструкция ?>

Здесь "кому" указывает имя приложения, которому адресована инструкция, а "инструкция" – информация, передаваемая приложению. Инструкция может состоять из любой последовательности символов, кроме пары ?>. В случае использования в качестве XML-процессора Internet Explorer, существуют две возможности применения инструкций по обработке:

– стандартные инструкции для сообщения браузеру, как отображать документ с использованием таблицы стилей, например:

```
<?xml-stylesheet type="text/css" href="Sample.css"?>
```

– Web-сценарий для отображения XML-документа. В таком случае возможно размещение в документе любые незарезервированные инструкции по обработке, которые будут использоваться вашим сценарием. Например, можно вставить в документ инструкцию по обработке, сообщающую сценарию уровень детализации при отображении:

```
<?MyScript detail="2"?>
```

При этом возможно разместить инструкцию по обработке в любое место XML-документа вне других элементов разметки: в пролог документа, после корневого элемента, либо внутри содержимого элемента.

Внутри символьных данных в содержимом элемента нельзя помещать символы угловых скобок (например, <) или знак амперсанда &. Одним из способов преодолеть это ограничение является использование ссылки на символ (< или &). Другим способом является использование ссылки на предопределённый общий примитив (< или &). Однако, в случае, если требуется многократно вставлять символы < и & (если, например, это какой-либо исходный код или разметка HTML), использование ссылок неудобно и затрудняет восприятие данных. В этом случае проще поместить текст, содержащий такие символы, в раздел CDATA.

Раздел CDATA начинается с символов <![CDATA[и заканчивается символами]]>. Между этими двумя ограничителями можно размещать любые символы, кроме]]>. В разделе CDATA не нужно использовать ссылки на символы или предопределённые общие примитивы, т.к. синтаксический анализатор не будет замещать такую ссылку соответствующим символом. Разделы CDATA не могут быть вложенными. Раздел CDATA можете поместить в любое место, занимаемое символьными данными – т.е. внутри содержимого элемента, но не внутри XML-разметки:

```
<goods> ... <![CDATA[ Text: <& ]]> ... </goods>
```

Разделы CDATA отображаются браузером как содержимое элемента.

БАЗОВЫЕ ПРАВИЛА XML

Ниже приведено несколько основных правил создания форматированного XML-документа. Форматированный документ соответствует минимальному набору правил, обеспечивающих возможность обработки документа браузером или другой программой. Рассмотрим эти правила:

- Документ должен иметь только один элемент верхнего уровня (элемент Документ или корневой элемент), при этом все другие элементы должны быть вложены в элемент верхнего уровня.

- Элементы должны быть вложены упорядоченным образом. Это означает что если элемент начинается внутри другого элемента, то он должен и заканчиваться внутри этого элемента.

- Каждый элемент должен иметь начальный и конечный тег. В отличие от HTML, в XML не разрешается опускать конечный тег – даже в том случае, когда браузер в состоянии определить, где заканчивается элемент. Однако возможно использование сокращенной нотации для пустых элементов (т.е. элементов, не имеющих содержимого).

- Имя типа элемента в начальном теге должно в точности соответствовать имени в соответствующем конечном теге.

- Имена типов элементов чувствительны к регистру, в котором они набраны.

Таким образом, можно сказать, что XML файл – это документ, в котором использованы теги для определения объектов и их атрибутов. При этом форматирование данных напоминает язык разметки HTML-документов. Но в отличие от HTML, в XML используются теги, которые задаются пользователями.

JSON

JSON (JavaScript Object Notation) – простой формат обмена данными, удобный для чтения и написания как человеком, так и компьютером. Он основан на подмножестве языка программирования JavaScript, определенного в стандарте ECMA-262 3rd Edition в декабре 1999 года.

По своей сути JSON это текстовый формат, полностью независимый от языка реализации, но он использует соглашения, знакомые программистам С-подобных языков, таких как С, С++, С#, Java, JavaScript, Perl, Python и многих других. Эти свойства делают JSON идеальным языком обмена данными.

Есть несколько основных правил для создания строки JSON. Строка JSON содержит либо массив значений, либо объект (ассоциативный массив пар имя/значение). Массив заключается в квадратные скобки ([и]) и содержит разделенный запятой список значений. Объект заключается в фигурные скобки ({ и }) и содержит разделенный запятой список пар имя/значение. Пара имя/значение состоит из имени поля, заключенного в

двойные кавычки, за которым следует двоеточие (:) и значение поля. При этом значение в массиве или объекте может быть:

- числом (целым или с плавающей точкой);
- строкой (записывается в двойных кавычках);
- логическим значением (true или false);
- массивом (заключенным в квадратные скобки)
- объект (заклученный в фигурные скобки)
- значением null

Строка в нотации JSON коллекция нуля или больше символов Unicode, заключенная в двойные кавычки, используя \ (обратную косую черту) в качестве символа экранирования. Таким образом, если понадобится включить двойные кавычки в строку, нужно использовать обратную косую черту – \". Так же, как и во многих языках программирования, можно помещать управляющие символы и шестнадцатеричные коды в строку, с использованием символа экранирования. Символ в JSON представляется как односимвольная строка.

Число представляется так же, как в C или Java, кроме того, что используется только десятичная система счисления.

На примере оформления заказа посмотрим, как будет выглядеть запись в формате JSON:

```
{
  "orderID": 12345,
  "clientName": "Ваня Иванов",
  "clientEmail": "ivanov@sample.ru",
  "contents": [
    {
      "productID": 345,
      "productName": "Зарядное устройство Power Energy PX40",
      "quantity": 1
    },
    {
      "productID": 765,
      "productName": "Аккумулятор Powercell 1800 мАч ",
      "quantity": 2
    }
  ],
  "orderCompleted": true
}
```

Рассмотрим эту строку более подробно. В начале создётся объект с помощью фигурных скобок ({ и }). В объекте указываются несколько пар имя/значение (табл. 3).

Табл. 3. Описание записи в формате JSON

Пара JSON имя/значение	Описание
"orderId": 12345	свойство с именем "orderId" и целочисленным значением 12345
"clientName": "Ваня Иванов"	свойство с именем "clientName" и строковым значением "Ваня Иванов"
"clientEmail": "ivanov@sample.ru"	свойство с именем "clientEmail" и строковым значением "ivanov@example.com"
"contents": [...]	свойство с именем "contents", значение которого является массивом
"orderCompleted": true	свойство с именем "orderCompleted" и логическим значением true

В массиве "contents" также есть 2 объекта, представляющие отдельные позиции в заказе. Каждый объект содержит 3 свойства: productID, productName, и quantity.

Во многих отношениях можно рассматривать JSON как альтернативу XML, по крайней мере, в сфере веб приложений. Концепция AJAX оригинально основывалась на использовании XML для передачи данных между сервером и браузером. Но в последние годы JSON становится все более популярным для переноса данных AJAX.

Хотя XML является проверенной технологией, которая используется в достаточном количестве приложений, преимуществами JSON являются более компактный и простой для распознавания формат данных.

Коммерческие CMS

СТРУКТУРА CMS БИТРИКС

Одной из коммерческих систем управления содержимым является Bitrix Framework – созданная на основе PHP платформа для разработки веб-приложений. На основе этой платформы создан продукт для создания всесторонних веб-проектов любой сложности – 1С-Битрикс: Управление сайтом. В базовой поставке идёт большой набор компонентов, и именно он обеспечивает быстрое развёртывание и внедрение проектов. Продукты на базе Bitrix Framework выходят в нескольких редакциях. На момент написания пособия наиболее популярной является редакция «Малый бизнес», которая применяется для Интернет-магазинов.

Битрикс обладает продуманной и удобной структурой и построена в соответствии с идеологией модель-представление-контроллер. В Битриксе моделью является API, контроллер – это тот или иной компонент, например, каталог товаров, а шаблоны компонентов или страниц являются представлением.

Несколько десятков модулей системы содержат набор функций, необходимых для реализации какой-то глобальной, большой задачи: веб-формы, работа интернет-магазина, организации социальной сети и другие. Модули также содержат инструментарий для администратора сайта для управления этими функциями.

Компоненты входят в состав модулей, но решают более узкую, частную задачу — например, выводят список новостей или товаров. Вносить свои изменения в код продукта рекомендуется на уровне компонентов. Программист может модифицировать их как угодно, использовать свои наработки и использовать неограниченное число шаблонов на каждый из компонентов. На одной странице сайта может располагаться несколько компонентов, кроме того, их можно включать в шаблон сайта. Таким образом, программист имеет возможность собрать сайт как конструктор, после чего доработать необходимые компоненты для получения желаемого результата как в функциональном, так и в визуальном плане.

Модуль – это набор каких-либо сущностей. Компонент – это то, что этими сущностями управляет.

Посмотрим на примере модуля Инфоблоки. Этот модуль представляет собой совокупность таблиц в базе данных и php-классов, которые могут проводить какие-либо операции с данными из таблиц (например, CIBlockElement::GetList() или CIBlockElement::GetByID ()). Компонентом является уже, например, Новость детально, который имеет собственные настройки (показывать дату, картинку и т.д. и т.п.) и работает с методами php-классов модуля.

Страница сайта представляет из себя PHP файл, состоящий из пролога, тела страницы (основной рабочей области) и эпилога. Формирование

страницы сайта производится динамически на основе используемого шаблона страницы, данных выводимых компонентами и статической информации, размещенной на странице.

Битрикс реализован на файлах, что дает больше свободы разработчику сайта. Поскольку файл в системе – это просто исполняемый файл, то и исполнять он может что угодно: хоть собственный PHP-код программиста, хоть стандартные компоненты – в любом порядке.

Файлы можно править как по FTP, так и в SSH, не прибегая к дополнительным инструментам СУБД. Их легко копировать, перемещать, делать резервные копии и т.п. Строго говоря, вы можете весь контент хранить в БД. Но для простых статичных сайтов это будет явное усложнение и замедление.

Реализация на файлах кажется проблематичной в том плане, что от такой системы ожидается десятки тысяч файлов на диске. Обычно это не так. Динамическая информация (новости, каталог товаров, статьи) сохраняются в БД модулем Информационные блоки. Тогда для вывода, например, десятка тысяч товаров в интернет-магазине используется одна единственная физическая страница (файл). В этом файле вызывается компонент инфоблоков, который в свою очередь выбирает и выводит товары из базы данных.

Например, для каталога товаров действительно нужно создать папку на диске, но только одну, например, /catalog, поместить туда комплексный компонент и далее страницы товаров могут иметь вид, например: http://***.ru/catalog/1029.html Естественно, что эти адреса будут «мнимыми» и обрабатываться системой. Файлов под них в папке /catalog не создается.

Однако, для каждого товара будет создан файл в кэше, чтобы, при следующем обращении покупателя сервер не напрягался с запросами к БД. Это и позволяет запускать магазины уровня Эльдorado.

При должном умении публичная часть может состоять из десятка физических файлов. Весь контент может быть в инфоблоках, включая меню. Но обычно статические страницы (например, О компании) удобнее редактировать как файл, а не как запись БД. Но если таких статических страниц становится неограниченно много, то это повод, чтобы структурировать их и разместить не на диске, а в инфоблоках.

Таким образом, в качестве инструмента хранения структуры сайта выбрана именно файловая система, а не база данных в силу того что:

- Файл дает больше свободы разработчику сайта, поскольку файл в системе – это просто исполняемый файл.

- Так понятнее для управления. В корне такого представления - структура статических страниц HTML, разложенных по папкам. Путем некоторого совершенствования (внедряя небольшое количество PHP-кода), мы из такого сайта сразу получаем работающий на **Bitrix Framework** проект.

– В какой-то мере это – традиция, которая имела большое значение на заре становления CMS.

– Такое представление соответствует опыту контент-менеджеров, которые работают с локальными файловыми системами (папки и файлы).

– Структура сайта также может быть и в БД (инфоблоки), но управлять иерархией в реляционной БД не очень-то удобно.

Рассмотрим использование файлов в Bitrix Framework на примерах:

1. Файловая система и меню. Меню в файлах позволяет не подключать БД там, где это реально не нужно. То же самое относится к свойствам страниц и разделов, а также правам доступа к файлам. Теоретически можно собрать информационный сайт, где вообще не будет ни одного обращения к БД. Будет работать быстрее, особенно на разделяемом хостинге. Есть и бонусы: при копировании раздела сразу естественным образом копируются меню, права доступа, свойства раздела.

2. Файловая система и пользователи. Пользователям из административного раздела открыт доступ к файлам ядра и другим программным файлам. Но пользователи бывают разные. Например, техподдержка 1С-Битрикса. Если веб-разработчик не уверен в своих пользователях, то он всегда может запретить им как редактирование кода PHP, так и целых разделов (ядра). По современной концепции **Битрикс** в публичной части не должно быть кода PHP – все должно быть инкапсулировано в компоненты. Тогда пользователь редактирует или текстово-графическое содержимое, или настраивает компонент.

3. Файловая система и языковые версии. Было бы трудно сопровождать языковую информацию в БД. Информация в языковых файлах меняется крайне редко – проще раз в год отредактировать строчку в языковом файле, чем хранить эти статические фразы в базе. И повторимся: база данных – это медленно и избыточно.

Файловая структура Битрикса организована таким образом, что программные компоненты ядра продукта были отделены от пользовательских файлов, а также файлов, определяющих внешнее представление сайта. Данная особенность позволяет:

– избежать нежелательной модификации ядра продукта при работе с файлами системы;

– исключить возможность изменения публичной части сайта при загрузке обновлений продукта.

– настроить внешний вид сайта практически под любую вашу задачу
Вся система целиком лежит в каталоге /bitrix/, в него входят ряд подкаталогов и файлов.

/admin/ - административные скрипты;

/cache/ - файлы кэша;

/activities/ - папки действий для бизнес-процесов;

/components/ - папка для системных и пользовательских компонентов;

/gadgets/ - папки гаджетов;

/js/ - файлы javascript модулей;

/stack_cache/ - файлы кеша "с вытеснением";
 /themes/ - темы административного раздела;
 /wizards/ - папки мастеров;
 /images/ - изображения используемые как системой в целом, так и отдельными модулями;
 /managed_cache/ - управляемый кеш;
 /modules/ - каталог с модулями системы, каждый подкаталог которого имеет свою строго определённую структуру;
 /php_interface/ - вспомогательный служебный каталог, в него входят следующие каталоги и файлы:
 dbconn.php - параметры соединения с базой;
 init.php - дополнительные параметры портала;
 after_connect.php - подключается сразу же после создания соединения с базой;
 dbconn_error.php - подключается при ошибке в момент создания соединения с базой;
 dbquery_error.php - подключается при ошибке в момент выполнения SQL запроса;
 /ID сайта/init.php - дополнительные параметры сайта; файл подключается сразу же после определения специальной константы с идентификатором сайта - **SITE_ID**;
 /templates/ - каталог с шаблонами сайтов и компонентов , в него входят следующие подкаталоги:
 /.default/ - подкаталог с общими файлами, используемыми тем или иным шаблоном по умолчанию, структура данного каталога аналогична нижеописанной структуре каталога содержащего конкретный шаблон;
 /ID шаблона сайта/ - подкаталог с шаблоном сайта, в него входят следующие подкаталоги и файлы:
 /components/ - каталог с кастомизированными шаблонами компонентов;
 /lang/ - языковые файлы принадлежащие как данному шаблону в целом, так и отдельным компонентам;
 /images/ - каталог с изображениями данного шаблона;
 /page_templates/ - каталог с шаблонами страниц и их описанием хранящимся в файле **.content.php**. Когда пользователь создает новую страницу, он может выбрать, по какому шаблону из представленных в этом каталоге это будет сделано;
 header.php - пролог данного шаблона;
 footer.php - эпилог данного шаблона;
 styles.css - CSS стили шаблона;
 /tools/ - при инсталляции в этот каталог копируются дополнительные страницы, которые могут быть непосредственно использованы на любых страницах сайта: помощь, календарь, показ изображения и т.п.;
 /updates/ - каталог, автоматически создаваемый системой обновлений;

`header.php` - стандартный файл, подключающий в свою очередь конкретный пролог текущего шаблона сайта; данный файл должен использоваться на всех страницах публичной части;

footer.php - стандартный файл, подключающий в свою очередь конкретный эпилог текущего шаблона сайта; данный файл должен использоваться на всех страницах публичной части;

license_key.php - файл с лицензионным ключом;

spread.php - файл используемый главным модулем для переноса куков посетителя на дополнительные домены различных сайтов;

redirect.php - файл используемый модулем **Статистика** для фиксации событий перехода по ссылке;

rk.php - файл по умолчанию используемый модулем **Реклама** для фиксации событий клика по баннеру;

stop_redirect.php - файл используемый модулем **Статистика** для выдачи какого либо сообщения посетителю, попавшему в стоп-лист;

activity_limit.php - файл используемый модулем **Статистика** для выдачи сообщения роботу при превышении им лимита активности;

и другие служебные файлы и папки.

В зависимости от используемой редакции некоторые каталоги и файлы могут отсутствовать.

РЕАЛИЗАЦИЯ ПРАВ ДОСТУПА К ИНФОРМАЦИИ

В системе Битрикс поддерживается два уровня разграничения прав доступа:

- Доступ на файлы и каталоги.
- Права в рамках логики модуля.

Доступ на файлы и каталоги

Этот уровень прав проверяется в прологе, задается с помощью специального файла `.access.php`, содержащего PHP массив следующего формата:

```
$PERM[файл/каталог][ID группы пользователей] = "ID права доступа";
```

Где:

– файл/каталог – имя файла или каталога для которых назначаются права доступа;

– ID группы пользователей – ID группы пользователей на которую распространяется данное право (допустимо также использование символа - *, что означает – для всех групп);

– ID права доступа – на сегодняшний день поддерживаются следующие значения (в порядке возрастания):

– D – запрещён (при обращении к файлу доступ будет всегда запрещён);

– R – чтение (при обращении к файлу доступ будет разрешен);

- U – документооборот (файл может быть отредактирован в режиме документооборота);
- W – запись (файл может быть отредактирован непосредственно);
- X – полный доступ (подразумевает право на "запись" и модификацию прав доступа).

В административной части сайта права доступа на файлы и каталоги можно назначать с помощью Менеджера файлов.

Если пользователь принадлежит нескольким группам, то берется максимальное право из всех прав доступа заданных для этих групп.

Если для текущего файла или каталога явно не задан уровень прав, то берется уровень прав заданный для вышележащих каталогов.

Пример 1

Файл /dir/.access.php

```
<?
    $PERM["index.php"]["2"] = "R";
    $PERM["index.php"]["3"] = "D";
?>
```

При попытке открытия страницы /dir/index.php пользователь, принадлежащий группе ID=3, будет иметь право доступа D (запрещено), пользователь из группы ID=2 будет иметь право R (чтение). Пользователь, принадлежащий обоим группам, будет иметь максимальный уровень доступа - R (чтение).

Пример 2

Файл /.access.php

```
<?
    $PERM["admin"]["*"] = "D";
    $PERM["admin"]["1"] = "R";
    $PERM["/"]["*"] = "R";
    $PERM["/"]["1"] = "W";
?>
```

Файл /admin/.access.php

```
<?
    $PERM["index.php"]["3"] = "R";
?>
```

При доступе к странице /admin/index.php пользователь, принадлежащий группе ID=3, будет иметь доступ, а пользователю, принадлежащему группе ID=2, будет в доступе отказано. При доступе к странице /index.php все посетители будут иметь доступ.

Права в рамках логики модуля

Если речь идет об обычных статичных публичных страницах, то к ним применяется только уровень 1 доступа на файлы и каталоги.

Если пользователь имеет на файл как минимум право R (чтение) и если данный файл является функциональной частью того или иного модуля, то проверяется 2-ой уровень прав, задаваемый в настройках соответствующего модуля. Например: При заходе на страницу Список обращений в техподдержке администратор видит все обращения, сотрудник техподдержки - только те за которые он ответственен, а обычный пользователь - только свои обращения. Так работает право доступа в рамках логики модуля Техподдержка.

Используются две методологии разграничения прав доступа 2-го уровня (уровень прав в рамках логики модуля):

- права;
- роли.

Отличие их заключается в том, что если пользователь обладает несколькими правами, то выбирается максимальное. Если же пользователь обладает несколькими ролями, то он соответственно будет обладать суммарными возможностями этих ролей.

Модули, в которых поддерживаются роли, можно увидеть в фильтре Модуль на странице Настройки > Пользователи > Уровни доступа в Административном разделе. Во всех остальных модулях и во всех остальных настройках системы используются права.

Пример:

Права. Если вы принадлежите группам для которых в модуле Статистика заданы права Полный административный доступ и например, Просмотр статистики без финансовых показателей, то вы будете обладать максимальным правом – Полный административный доступ.

Роли. Если вы принадлежите к группам для которых в модуле Техподдержка заданы роли Клиент техподдержки и Демо-доступ, то вы одновременно будете обладать возможностями этих двух ролей. Т.е. вы сможете видеть все обращения в режиме демо-доступа и одновременно с этим можете создавать свои обращения как клиент техподдержки.

СОЗДАНИЕ САЙТА

В Битрикс имеется возможность на базе одного экземпляра продукта создавать и поддерживать неограниченное количество сайтов. Особенности системы многосайтовости являются:

- единые права на управление модулями сайта;
- единый набор пользователей на все сайты;
- единая система ведения статистики на все сайты.

Сайт – это совокупность:

- Учетной записи в базе данных, которая создаётся в административном меню на странице Список сайтов (Настройки > Настройки продукта > Сайты > Список сайтов) и включает в себя следующие основные параметры:

- идентификатор - набор символов, идентифицирующий сайт;

- доменное имя - одно или более доменное имя сайта;
- папка сайта - путь к каталогу, в котором будет храниться публичная часть сайта;
- язык сайта;
- формат даты;
- формат времени;
- URL – протокол и доменное имя по умолчанию (например, <http://www.site.ru>)
- DocumentRoot – корень сайта;
- условия подключения шаблонов: каждый сайт может иметь один или более шаблонов для отображения страниц публичной части, каждый такой шаблон может быть подключен по тому или иному условию.
- Публичной части – совокупности скриптов (страниц) лежащих в папке сайта и принадлежащих этому сайту.
- Настроек. Каждый модуль может иметь ряд настроек связанных с сайтом, например, у модуля Информационные блоки эти настройки представляют из себя привязку информационного блока к тому или иному сайту, у модуля Техподдержка - привязку статуса, категории и т.п. к сайту.

СТРУКТУРА САЙТА

Битрикс при создании сайта предполагает следующую структуру:

- Шаблон – определяет представление сайта пользователям. Существуют шаблоны компонентов и шаблоны сайта.
- Компоненты – задают вывод данных.
- Страница – элемент структуры сайта.

Страница представляет из себя PHP файл, состоящий из пролога, тела страницы (основной рабочей области) и эпилога.

Формирование страницы сайта производится динамически, на основе используемого шаблона страницы, данных, выводимых компонентами, и статической информации, размещенной на странице. Создание шаблонов сайта и размещение на них компонентов осуществляется разработчиками сайтов.

В общем случае все страницы сайта имеют следующую структуру, изображенную на рис. 24.

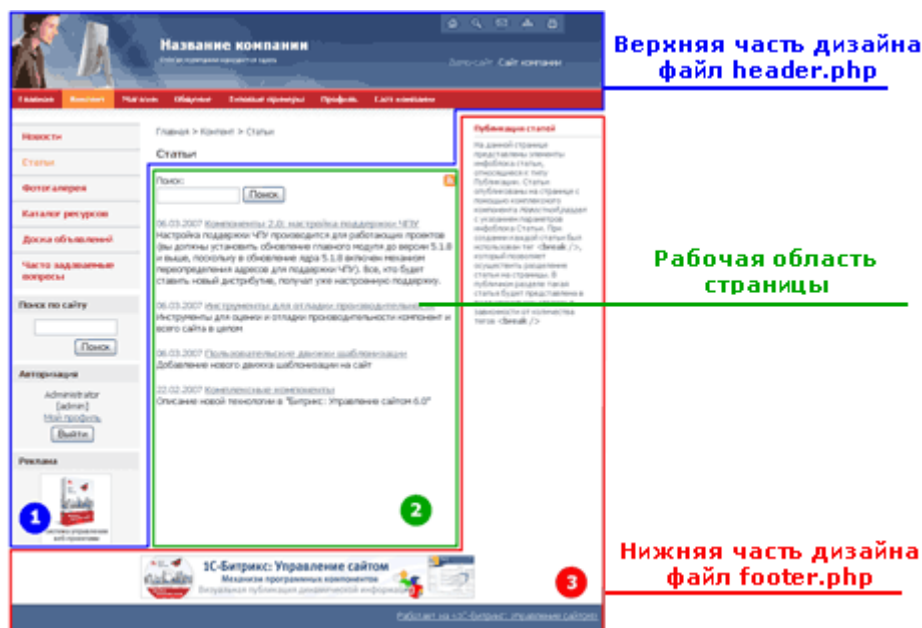


Рис. 24. Структура страницы сайта

Верхняя - header. Включает в себя, как правило, верхнюю и левую часть дизайна со статической информацией (логотипом, лозунгом и так далее), верхним горизонтальным меню и левым меню (если они есть в дизайне). Может включать в себя информационные динамические материалы.

Основная рабочая область – work area. Рабочая область страницы, в которой размещаются собственно информационные материалы сайта. В качестве Основной рабочей области может подключаться как физический файл, так и создаваемый системой на основе комплексных компонентов, динамический код.

Если в качестве основной рабочей области подключается физический файл, то такая страница называется статической. Если подключается динамический код, то такая страница называется динамической.

Нижняя – footer. Включает в себя, как правило, статическую информацию (контактная информация, сведения об авторе и владельце сайта и так далее), нижнее горизонтальное меню и правое меню (если они есть в дизайне). Может включать в себя информационные материалы.

Верхняя и нижняя части дизайна формируются на основе шаблона дизайна сайта. Т.е. информация, отображаемая в данных областях, определяется параметрами шаблона сайта.

В общем случае структура страницы выглядит как показано во фрагменте кода:

```
<?
// подключение пролога
require($_SERVER["DOCUMENT_ROOT"]."/bitrix/header.php");
?>
тело страницы
<?
```

```
// подключение эпилога
require($_SERVER["DOCUMENT_ROOT"]."/bitrix/footer.php");
?>
```

Благодаря технологии отложенных функций часть визуальных элементов выводимых в прологе может быть задана в теле страницы, это такие элементы как:

- заголовок страницы (выводится функцией CMain::ShowTitle);
- навигационная цепочка (выводится функцией CMain::ShowNavChain);
- CSS стили (выводятся функцией CMain::ShowCSS);
- мета-теги (выводятся функцией CMain::ShowMeta);
- другое.

Принципиальной особенностью данной технологии является то, что она позволяет отложить исполнение некоторых функций, выполняя их в эпилоге, а результаты их выполнения подставляя в вышележащий код.

Ряд задач не могут быть решены с помощью технологии отложенных функций. Например, когда необходимо производить какие-либо действия в Прологе над значениями, которые в предыдущем примере задавались бы в теле страницы (например, свойства страницы). В этом случае возникает необходимость разбить пролог на служебную и визуальную части и эти значения задавать между ними.

Достигается это следующим образом:

```
<?
// подключение служебной части пролога
re-
quire($_SERVER["DOCUMENT_ROOT"]."/bitrix/modules/main/include/prolo
g_before.php");

// здесь можно задать например, свойство страницы
// с помощью функции $APPLICATION->SetPageProperty
// и обработать затем его в визуальной части эпилога

// подключение визуальной части пролога
re-
quire($_SERVER["DOCUMENT_ROOT"]."/bitrix/modules/main/include/prolo
g_after.php");
?>
Содержимое страницы
<?
// подключение эпилога
require($_SERVER["DOCUMENT_ROOT"]."/bitrix/footer.php");
?>
```

В служебной части пролога происходит:

- подключение к базе данных MySQL или другой;
- обработка агентов;
- инициализация служебных констант;
- проверка прав доступа к файлам и каталогам;
- подключение необходимых модулей
- исполнение обработчиков событий OnPageStart, OnBeforeProlog;
- ряд других необходимых действий.

Особенностью служебной части пролога является то, что она не выводит никаких данных (не посылает header браузеру).

В визуальной части пролога происходит подключение файла /bitrix/templates/ID шаблона сайта/header.php, где ID шаблона сайта - идентификатор текущего шаблона сайта. Данный файл хранит левую верхнюю часть текущего шаблона сайта.

Эпилог тоже может быть разбит на визуальную и служебную части:

```
<?
// подключение служебной части пролога
re-
quire($_SERVER["DOCUMENT_ROOT"]."/bitrix/modules/main/include/prolo
g_before.php");

// подключение визуальной части пролога
re-
quire($_SERVER["DOCUMENT_ROOT"]."/bitrix/modules/main/include/prolo
g_after.php");

?>
Содержимое страницы
<?

// подключение визуальной части эпилога
re-
quire($_SERVER["DOCUMENT_ROOT"]."/bitrix/modules/main/include/epilo
g_before.php");

// подключение служебной части эпилога
re-
quire($_SERVER["DOCUMENT_ROOT"]."/bitrix/modules/main/include/epilo
g_after.php");
?>
```

В визуальной части эпилога происходит подключение файла /bitrix/templates/ID шаблона сайта/footer.php, где ID шаблона сайта - идентификатор текущего шаблона сайта. Данный файл хранит правую нижнюю часть текущего шаблона сайта. Помимо этого - выводится ряд невидимых IFRAME'ов, используемых технологией переноса посетителей.

В служебной части эпилога происходит:

- отправка почтовых сообщений;
- исполнение обработчиков события;
- отключение от базы;
- ряд других служебных действий.

Довольно часто возникают задачи, когда нет необходимости в подключении визуальной частей пролога и эпилога. В этом случае достаточно подключить служебные части пролога и эпилога:

```
<?
// подключение служебной части пролога
re-
quire($_SERVER["DOCUMENT_ROOT"]."/bitrix/modules/main/include/prolo
g_before.php");
?>
тело страницы
<?
// подключение служебной части эпилога
re-
quire($_SERVER["DOCUMENT_ROOT"]."/bitrix/modules/main/include/epilo
g_after.php");
?>
```

Для корректной работы системы, обязательным являются подключения служебных частей пролога и эпилога.

СОЗДАНИЕ ШАБЛОНА СТРАНИЦЫ

Шаблон страницы – это PHP файл, содержимое которого соответствует правилам формирования структуры страницы. Шаблоны могут использоваться при создании новой страницы.

Шаблоны страниц хранятся в каталогах:

- /bitrix/templates/.default/page_templates/;
- /bitrix/templates/ID шаблона сайта/page_templates/.

В каждом таком каталоге могут находиться непосредственно сами файлы шаблонов страниц, а также служебный файл .content.php, основная задача которого - хранить описания и порядок сортировки шаблонов страниц. Эта информация хранится в массиве \$TEMPLATE структура которого представлена ниже:

```
Array
(
    [имя файла] => Array
        (
            [name] => заголовок шаблона страницы
```

```

        [sort] => индекс сортировки
    )
)

```

При формировании списка шаблонов страниц используется следующий алгоритм:

- получаем ID шаблона сайта для текущего сайта который подключается без PHP условия;
- последовательно подключаем файлы:
 - /bitrix/templates/.default/page_templates/.content.php
 - /bitrix/templates/ID шаблона сайта/page_templates/.content.php

В каждом из этих файлов будет описан свой массив \$TEMPLATE. После подключения этих файлов, мы будем иметь объединенный массив \$TEMPLATE. Далее для каждого элемента этого массива, представляющего из себя описание одного шаблона страницы, выполняем следующее:

- проверяем физическое существование шаблона страницы
- если существует, то добавляем его в список шаблонов
- сортируем полученный список шаблонов по индексу сортировки.

Свойства раздела хранятся в файле .section.php соответствующего каталога (раздела сайта). Свойства страницы задаются, как правило, либо в теле страницы, либо между служебной частью и визуальной частью пролога.

Свойства раздела автоматически наследуются всеми подразделами и страницами данного раздела. При необходимости вы можете отредактировать свойства любой отдельно взятой страницы раздела, подправив ее параметры под конкретную ситуацию.

В работе со свойствами используются следующие функции:

CMain::SetPageProperty – устанавливает свойство страницы.

CMain::SetDirProperty - устанавливает свойство раздела.

```

<?
    $APPLICATION->SetPageProperty("keywords", "веб, разработ-
ка, программирование");
    $APPLICATION->SetDirProperty("keywords", "дизайн, веб,
сайт");
?>

```

CMain::ShowProperty – выводит свойство страницы или свойство раздела с использованием технологии отложенных функций.

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title><?$APPLICATION->ShowProperty("page_title")?></title>

```

```

</head>
<body    link="#525252"    alink="#F1555A"    vlink="#939393"
text="#000000">
...

```

CMain::GetProperty – возвращает свойство страницы или свойство раздела.

```

<?
    $keywords = $APPLICATION->GetProperty("keywords");
    if (strlen($keywords)>0) echo $keywords;
?>

```

CMain::GetPageProperty - возвращает свойство страницы.

```

<?
    $keywords = $APPLICATION->GetPageProperty("keywords");
    if (strlen($keywords)>0) echo $keywords;
?>

```

CMain::GetPagePropertyList - возвращает массив всех свойств страницы.

```

<?
    $arProp = $APPLICATION->GetPagePropertyList();
    foreach($arProp as $key=>$value)
        echo '';
?>

```

CMain::GetDirProperty - возвращает свойство раздела.

```

<?
    $keywords = $APPLICATION->GetDirProperty("keywords");
    if (strlen($keywords)>0) echo $keywords;
?>

```

CMain::GetDirPropertyList - возвращает массив свойств раздела, собранных рекурсивно до корня сайта.

```

<?
    $arProp = $APPLICATION->GetDirPropertyList();
    foreach($arProp as $key=>$value)
        echo '';
?>

```

Для работы с мета-тегами используются свойства страницы и раздела. Для работы с ними используются следующие функции:

CMain::ShowMeta – выводит свойство страницы или свойство раздела, обрамляя их HTML-тегом <meta> с использованием технологии отложенных функций.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<?$APPLICATION->ShowMeta("keywords_prop", "keywords")?>
<?$APPLICATION->ShowMeta("description_prop", "description")?>
</head>
<body link="#525252" alink="#F1555A" vlink="#939393"
text="#000000">
...
```

CMain::GetMeta - возвращает свойство страницы или свойство раздела, обрамляя их HTML-тегом <meta>.

```
<?
    $meta_keywords = $APPLICATION->GetMeta("keywords_prop",
"keywords");
    if (strlen($meta_keywords)>0) echo $meta_keywords;
?>
```

Параметры страницы предназначены для передачи параметров в функции модулей для изменения их стандартного поведения. Например, при необходимости отключить запоминание последней страницы в сессии (при использовании постраничной навигации) или изменить стандартный формат вывода даты в функциях модуля Информационные блоки.

Параметры страницы доступны только в пределах страницы. Они не сохраняются ни в базе, ни в сессии.

Для работы с параметрами страницы предназначен класс CPageOption.

Примеры использования:

```
<?
CPageOption::SetOptionString("main", "nav_page_in_session",
"N");
CPageOption::SetOptionString("iblock", "FORMAT_ACTIVE_DATES",
"FULL");
?>
```

Общий порядок выполнения страницы следующий:

- Служебная часть пролога;
- Визуальная часть пролога;
- Тело страницы;
- Визуальная часть эпилога;

– Служебная часть эпилога.

Параметры компонента и шаблона доступны из программных модулей компоненты и шаблона как массив `$arParams`. Результатом работы программного модуля компоненты является массив `$arResult`, подаваемый на вход шаблона компонента. Результирующий HTML-код выводится обычным оператором `echo` на поток (при этом он встраивается в нужное место страницы).

Во время работы компонента и шаблона имеется возможность использовать функционал модулей Bitrix Framework, которые, в свою очередь, могут обращаться к базе данных продукта.

Отображение страниц в публичном разделе сайта выполняется на основе шаблонов дизайна сайта.

ШАБЛОН ДИЗАЙНА

Шаблон дизайна – это внешний вид сайта, в котором определяется расположение различных элементов на сайте, художественный стиль и способ отображения страниц. Включает в себя программный html-код, графические элементы, таблицы стилей, дополнительные файлы для отображения контента. Может также включать в себя шаблоны компонентов, шаблоны готовых страниц и сниппеты.

В общем случае шаблон сайта задает «обрамление» страницы, а за вывод динамической информации отвечают визуальные компоненты.

Количество используемых на сайте шаблонов дизайна не ограничено. Для каждого шаблона определяется условие, при котором данный шаблон будет применяться к страницам сайта (рис. 25)

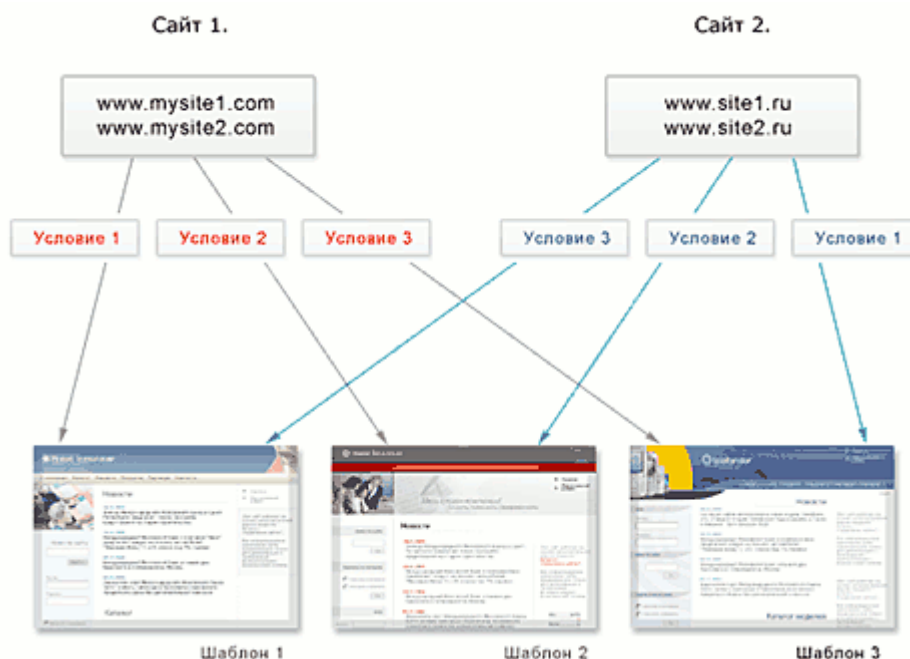


Рис. 25. Применение шаблонов к страницам сайта

Настройка условий применения того или иного шаблона определяется отдельно для каждого сайта в форме создания и редактирования сайта Настройки > Настройки продукта > Сайты > Список сайтов (рис. 26):

Шаблон	Сорт.	Тип условия	Условие
Настольное приложение	1	Для папки или файла	/desktop_app/
Корпоративный сайт	150	[без условия]	<без условия>
Версия для печати	150	Параметр в URL	print = Y
Прохождение курса обучения 9.1.0	150	Для папки или файла	/communication/learning/co
PHP	151	Для папки или файла	/php/
(нет)	152	Для групп пользователей	<ul style="list-style-type: none"> Администраторы Пользователи имеющие право голоса Зарегистрированные пользователи Пользователи имеющие право голоса Пользователи панели управления

Рис. 26. Настройка условий применения шаблонов сайта

В шаблон сайта входят:

Набор файлов в каталоге /bitrix/templates/ID шаблона сайта/, где ID шаблона сайта - поле ID в форме редактирования шаблона сайта. Ниже представлена структура данного каталога:

- файл header.php - пролог данного шаблона;
- файл footer.php - эпилог данного шаблона;
- файл styles.css - CSS стили шаблона;
- /components/ - каталог с шаблонами компонентов, принадлежащими тому или иному модулю;
- /lang/ - языковые файлы, принадлежащие как данному шаблону в целом, так и отдельным компонентам;
- /images/ - каталог с изображениями данного шаблона сайта;
- /page_templates/ - каталог с шаблонами страниц и их описанием, хранящимся в файле .content.php. Когда пользователь создает новую страницу, он может выбрать, по какому шаблону он это сделает.
- /include_areas/ - каталог с файлами - содержимым включаемых областей. Произвольная (т.е. название не регламентировано) папка для файлов включаемых областей компонента (main.include). Обычно вызов компонента делают из шаблона сайта. Файлы включаемых областей группируют в эту папку для удобства, т.к. они могут иметь оформление, специфичное для конкретного шаблона сайта. Там могут быть название, лого, контактная информация - чтобы пользователь редактировал только эти данные, не затрагивая шаблон сайта.
- ряд других вспомогательных произвольных файлов и папок, входящих в данный шаблон.

Стили шаблона сайта подключаются последними - это крайняя возможность переопределить, например, стили стандартных компонентов. Иначе потребуются масштабная кастомизация шаблонов компонентов. Если есть необходимость подключить свои стили последними, то это можно сделать с помощью функции `CMain::AddHeadString`.

ЯЗЫК САЙТА

Язык – это учетная запись в базе данных, доступная для редактирования в административном меню на странице Настройки > Настройки продукта > Языковые параметры > Языки интерфейса, со следующими основными полями:

- идентификатор;
- название;
- формат даты;
- формат времени;
- кодировка.

Как в публичной, так и в административной частях, язык используется в первую очередь для выбора того или иного языкового файла.

В административной части язык определяет формат времени и даты, кодировку страниц. В публичной части - данные параметры определяются настройками сайта.

Языковой файл - PHP скрипт, хранящий переводы языковых фраз на тот или иной язык.

Данный скрипт состоит из массива `$MESS`, ключи которого - идентификаторы языковых фраз, а значения - переводы на соответствующий язык.

Пример языкового файла для русского языка:

```
<?
$MESS [ 'SUP_SAVE' ] = "Сохранить";
$MESS [ 'SUP_APPLY' ] = "Применить";
$MESS [ 'SUP_RESET' ] = "Сбросить";
$MESS [ 'SUP_EDIT' ] = "Изменить";
$MESS [ 'SUP_DELETE' ] = "Удалить";
?>
```

Пример языкового файла для английского языка:

```
<?
$MESS [ 'SUP_SAVE' ] = "Save";
$MESS [ 'SUP_APPLY' ] = "Apply";
$MESS [ 'SUP_RESET' ] = "Reset";
$MESS [ 'SUP_EDIT' ] = "Change";
$MESS [ 'SUP_DELETE' ] = "Delete";
?>
```

Для каждого языка существует свой набор языковых файлов, хранящихся в подкаталогах /lang/ структуры файлов системы или модуля.

Языковые файлы как правило используются в административных скриптах модулей или в компонентах и в зависимости от этого подключаются одной из следующих функций:

- IncludeModuleLangFile - в административных скриптах
- IncludeTemplateLangFile - в компонентах

Для удобства поиска и дальнейшей модификации языковых фраз можно пользоваться параметром страницы show_lang_files=Y, позволяющим быстро найти и исправить ту или иную языковую фразу.

АДМИНИСТРАТИВНАЯ ПАНЕЛЬ

Административная панель управления – HTML-код, который может быть выведен авторизованному пользователю при наличии у него достаточных прав на операции, представленные на панели управления. HTML-код представляет из себя область с кнопками, в самом верху страницы, каждая из которых предназначена для той или иной операции (рис. 27).



Рис. 27. Административная панель сайта

Подключение панели осуществляется с помощью функции CMain::ShowPanel. Данная функция использует технологию отложенных функций, что позволяет добавлять кнопки в панель непосредственно в теле страницы.

Добавить кнопку в панель можно с помощью функции CMain::AddPanelButton. Эту же функцию можно использовать в скрипте /bitrix/php_interface/include/add_top_panel.php, который автоматически будет подключен при выводе панели.

Панель управления имеет два основных режима:

- Сайт - режим для работы над содержимым сайта.
- Администрирование - административный раздел для полнофункционального управления всем интернет-проектом.

СОЗДАНИЕ ШАБЛОНА ДИЗАЙНА САЙТА

Bitrix Framework имеет возможность создания шаблона с помощью встроенного визуального редактора. Однако этот редактор разработан для работы с контентом. Поэтому этой возможностью пользоваться не рекомендуется. Редактирование шаблона дизайна сайта в визуальном режиме будет происходить некорректно, если:

- в атрибутах HTML-тегов содержится php-код;
- если строки и ячейки таблицы прерываются php-кодом при формировании таблицы.

Если в коде шаблона дизайна сайта есть такие особенности, то редактировать его следует только в режиме кода. Также не рекомендуется редактировать шаблон в визуальном редакторе при наличии сложной верстки.

Перед началом работы необходимо определить, сколько различных шаблонов сайта понадобится. Обычно при разработке сайта прорисовываются все различные страницы или основные элементы сайта.

Bitrix Framework позволяет использовать неограниченное число шаблонов и назначать их по разным условиям. Рассмотрим простейший вариант, что на всех этих страницах простого сайта фактически меняется только контентная часть, а дизайн – не изменяется. Исключение составляет главная страница, у которой контентная область устроена по-другому (не содержит заголовка страницы) и разделена на две части. Это можно реализовать как дополнительными условиями в шаблоне сайта, так и созданием двух шаблонов. Рекомендуется использовать дополнительные условия, в этом случае потребуется всего один шаблон сайта.

Перейти к редактированию шаблона можно любым из способов:

- создав (открыв для редактирования) нужный шаблон (Настройки > Настройки продукта > Сайты > Шаблоны Сайтов);
- выбрав Шаблон в меню Пуск (Настройки > Настройки продукта > Сайты > Шаблоны Сайтов);
- с помощью кнопки Шаблон сайта на Панели управления (Шаблон сайта > В Панели управления > Редактировать шаблон);
- прямое редактирование файлов header.php и footer.php в папке шаблона.

Проанализируйте шаблон и определите, какая часть кода должна относиться к прологу (файл header.php), какая к эпилогу (файл footer.php), а какая часть – к рабочей области страницы. Выбранные части кода должны быть размещены в соответствующих файлах, а рабочая область должна быть отмечена тегом #WORK_AREA# в шаблоне сайта.

Добавьте соответствующий этим частям код в указанные файлы. Либо, если редактирование происходит в редакторе, разделите их тегом #WORK_AREA#, удалив из шаблона контентную часть.

Необходимо заменить некоторые части верстки на служебные директивы Bitrix Framework для создания шаблона:

- Заменить подключение стилей и, возможно, javascript файлов на директиву `<?$APPLICATION->ShowHead()?>`.
- Заменить прописанный явно заголовок страницы на `<title><?$APPLICATION->ShowTitle()?></title>`.
- Сразу после тэга `<body>` добавить `<?$APPLICATION->ShowPanel();?>`. Если этого не сделать, панель управления не появится.

– Перед всеми картинками добавить путь к ним `/bitrix/templates/<?echo SITE_TEMPLATE_ID;?>/images/`.

– Заменить контент на специальный разделитель - `#WORK_AREA#`.

Для создания полноценного сайта необходимо интегрировать в шаблон компоненты. Для этого необходимо выделить в дизайне сайта блоки, которые содержат динамическую информацию (вместо них будет размещен вызов компонентов) и блоки, информация в которых должна изменяться пользователем без изменения шаблона (это будут включаемые области).

Пример выделения функциональных областей и выбора компонентов для реализации функций показан на рис. 28.



Рис. 28. Функциональные области сайта

Если шаблон редактируется без визуального редактора системы (а рекомендуется именно такой вариант), то возможны два варианта вставки кода визуальных компонентов:

– из пользовательской документации. В ней описан формат вызова всех компонентов.

– вставкой кода в визуальном редакторе на какой-нибудь тестовой странице. Полученный код копируется в нужное место шаблона. Рекомендуется именно этот способ, так как документация может отставать от актуальной версии продукта, или вы можете использовать неактуальную версию продукта.

Компоненты версии 2.0 подключаются в коде страницы с помощью функции `IncludeComponent()`. В качестве параметров функции используется:

– название компонента в форме `<пространство_имен>:<название_компонента>`. Причем название компонента реко-

мендуется строить иерархически, начиная с общего понятия и заканчивая конкретным назначением компонента. Например, catalog, catalog.element, catalog.section.list и т.п;

- название шаблона. Шаблон компонента "по умолчанию" можно задавать пустой строкой, также можно явно определять .default;

- массив параметров компонента Array(...).

Код для вставки компонента «Включаемая область (bitrix:main.include)», с настройкой на вывод из файла, выглядит так:

```
<?APPLICATION->IncludeComponent(
    "bitrix:main.include",
    "",
    Array(
        "AREA_FILE_SHOW" => "file",
        "PATH" => $APPLICATION-
>GetTemplatePath("include_areas/company_name.php"),
        "EDIT_TEMPLATE" => ""
    )
);?>
```

В этом примере include_areas/company_name.php - файл с включаемой областью:

```
<?if(!defined("B_PROLOG_INCLUDED"))||B_PROLOG_INCLUDED!==true)
die();?>
World Book
Все книги мира
```

В файле включаемой области не требуется подключать пролог и эпилог (файлы header.php и footer.php). Необходимо лишь проверить, что файл включаемой области подключен из системы, а не вызван напрямую. Это делает строка:

```
<?if(!defined("B_PROLOG_INCLUDED"))||B_PROLOG_INCLUDED!==true)
die();?>
```

Для подключения меню необходимо вставить в нужное место шаблона вызов компонента:

```
<?APPLICATION->IncludeComponent(
    "bitrix:menu",
    "horizontal_multilevel",
    Array(
        "ROOT_MENU_TYPE" => "top",
        "MAX_LEVEL" => "3",
        "CHILD_MENU_TYPE" => "left",
        "USE_EXT" => "Y",
```

```

        "MENU_CACHE_TYPE" => "A",
        "MENU_CACHE_TIME" => "3600",
        "MENU_CACHE_USE_GROUPS" => "Y",
        "MENU_CACHE_GET_VARS" => Array()
    )
};?>

```

Это можно сделать как через визуальный редактор, так и через PHP-код шаблона.

По умолчанию шаблоны компонентов имеют некоторый стандартный вид. Для приведения формы вывода данных в соответствие с дизайном сайта, а также для организации вывода данных в нестандартном виде применяется кастомизация шаблонов компонентов.

На простых сайтах, как правило, шаблоны требуют кастомизации только при решении первой задачи. Для изменения системного шаблона компонента под конкретный сайт, его необходимо сначала целиком скопировать в папку шаблона сайта. После этого можно перейти к редактированию скопированного шаблона.

Приведем пример кастомизации шаблона компонента меню.

Выделите в HTML-верстке код, отвечающий за показ верхнего меню. Например:

```

<div class="topmenu">
    <ul class="topmenu">
        <li><a href="#" class="first">На главную</a></li>
        <li><a href="#">Новости</a></li>
        <li><a href="#" class="selected">Магазины</a></li>
        <li><a href="#">Книги</a></li>
        <li><a href="#">Форум</a></li>
        <li><a href="#">О компании</a></li>
        <li><a href="#">О Контакты</a></li>
    </ul>
</div>

```

В этом коде пункты меню представлены в виде списка, который обладает следующими нюансами:

- у первого пункта меню должен быть указан стиль `first`;
- у выделенного пункта меню должен быть указан стиль `selected`;
- меню является одноуровневым.

Модифицировать будем код шаблона `gray-tabs-menu`. Скопируйте шаблон в собственное пространство имен и откройте его для редактирования.

Редактирование шаблона можно проводить как в форме Bitrix Framework, так и копированием кода и правкой его в другом редакторе. Использовать другой редактор удобнее в случае объемных текстов, так как

форма редактирования в Bitrix Framework не поддерживает цветное выделение тегов. Для примера используйте Notepad++.

Код меню для этого шаблона выглядит так:

```

1<?if          (!defined("B_PROLOG_INCLUDED"))          ||
B_PROLOG_INCLUDED!==true)die();?>
2
3 <?if (!empty($arResult)):?>
4 <div class="gray-tabs-menu">
5     <ul>
6 <?foreach($arResult as $arItem):?>
7
8     <?if ($arItem["PERMISSION"] > "D"):?>
9         <li><a
href="<?=$arItem["LINK"]?>"><nobr><?=$arItem["TEXT"]?></nobr></a>
10         <?endif?>
11
12 <?endforeach?>
13
14     </ul>
15</div>
16<div class="menu-clear-left"></div>
17<?endif?>

```

Рассмотрим каждую строчку этого шаблона в табл. 4.

Табл. 4. Описание шаблона реализации меню

1.	Проверка, вызван ли этот файл напрямую или нет. Если напрямую – прекратить работу.
3.	Если массив с пунктами меню \$arResult не пуст, то выполнять дальнейшие действия
4,5.	Внешний блок и начало списка пунктов меню
6.	Цикл по массиву с пунктами меню. В \$arItem – текущий элемент цикла.
8-10.	Если текущий пользователь обладает правами на просмотр данной страницы, вывести элемент списка с ссылкой на эту страницу. В полях LINK и TEXT содержится адрес страницы и название пункта меню, соответственно.
12.	Конец цикла по массиву с пунктами меню.
14,15.	Конец списка пунктов меню и конец блока
16.	Специальный HTML-тэг, специфичный для использованной верстки
17.	Конец условия на наличие пунктов меню (см. строку 3)

Таким образом, шаблон меню содержит:
– область пролога шаблона меню;

- область тела шаблона меню (вывод повторяющихся элементов);
- область эпилога шаблона меню.

После адаптации шаблон примет вид (зеленым цветом выделены изменения):

```

1      <?if      (!defined("B_PROLOG_INCLUDED"))      ||
B_PROLOG_INCLUDED!==true)die();?>
2
3 <?if (!empty($arResult)):?>
4 <div class="topmenu">
5     <ul class="topmenu">
5a <? $cnt=0; ?>
6 <?foreach($arResult as $arItem):?>
7
8     <?if ($arItem["PERMISSION"] > "D"):?>
8a     <?if ($arItem["SELECTED"]): ?>
8b     <li><a                                href="<?=$arItem["LINK"]?>"
class="selected"><?=$arItem["TEXT"]?></a></li>
8c     <?else:?>
8d         <?if ($cnt==0):?>
8e     <li><a                                href="<?=$arItem["LINK"]?>"
class="first"><?=$arItem["TEXT"]?></a></li>
8f         <?else:?>
9     <li><a
href="<?=$arItem["LINK"]?>"><?=$arItem["TEXT"]?></a></li>
10a     <?endif?>
10b     <?endif?>
10c <?$cnt++; ?>
10 <?endif?>
11
12 <?endforeach?>
13
14     </ul>
15</div>
16
17<?endif?>

```

Что сделано в этом примере кода:

- В строках 4,5 заменили стили у блока и у списка.
- В строке 5а ввели переменную \$cnt с единственной целью – отследить первый элемент списка – в верстке он задается другим стилем. Эта переменная используется в строке 8d в строке 10с.
- В строках 8-10 добавили условие проверки активного пункта меню и первого пункта меню.
- И, наконец, удалили специфику верстки из 16 строки.

– Кроме того, у нас нет необходимости в специализированных стилях и картинках для этого шаблона. Поэтому нужно удалить из каталога /bitrix/templates/.default/components/menu/top_menu/ файл style.css и папку /images/.

СОЗДАНИЕ СТРАНИЦ И РАЗДЕЛОВ САЙТА

Перед тем как создать новую страницу или раздел нужно определить к какому типу информации они будут относиться.

Для статической информации в рамках файловой структуры сайта страница - это файл, а раздел - это папка. Соответственно, создавая страницу вы создаете файл, создавая раздел - папку.

Для динамической информации необходимо создать только одну физическую страницу и разместить на ней комплексный компонент, который и создаст раздел, состоящий из динамически (программно) создаваемых страниц. Как правило эти действия уже выполнены разработчиком сайта или его администратором и контент-менеджеру остается только создавать отдельные страницы с динамическим содержанием. Примерно таким способом.

Создавать раздел или страницу со статической информацией можно как с публичной части, так и с административного раздела.

Создание страниц осуществляется с помощью кнопки на рис. 29.

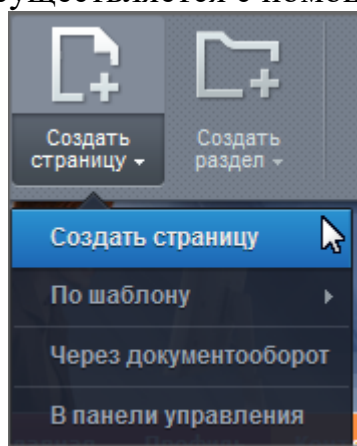


Рис. 29. Создание страницы

С помощью верхней части кнопки можно в один клик запустить мастер создания страницы. С помощью нижней части кнопки можно выбрать вариант создания страницы: обычным мастером, из административной части или через документооборот.

Мастер создает страницу в несколько шагов. Число шагов зависит от активных опций, выбранных в первом шаге. Если все опции будут неактивными, то завершение создания страницы возможно уже после первого шага. Завершение работы мастера производится по нажатию на кнопку Готово.

Первый шагю

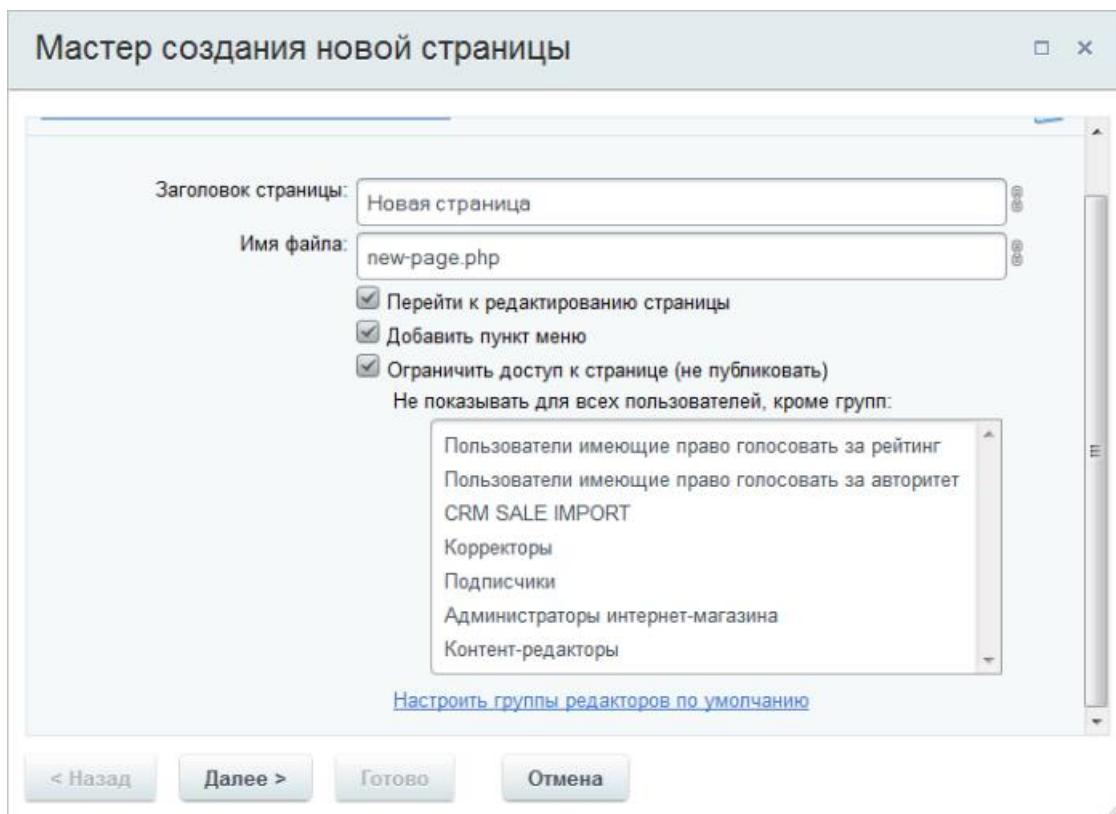



Рис. 29. Создание страницы

Поле Заголовок страницы - это название страницы.

Поле Имя файла - это название файла в рамках файловой структуры сайта. По умолчанию формируется автоматически из поля Заголовок страницы. Если кликнуть по иконке , то автоматическое формирование файла отключится и название необходимо будет ввести вручную.

В названии файла недопустимо использование следующих спецсимволов: \ / ? * < > " ' .

Снятие флажка Перейти к редактированию страницы исключит из мастера последний шаг и система создаст пустую страницу.

Снятие флажка Добавить пункт меню исключит из мастера второй шаг и система создаст страницу, не отображаемую в меню.

Установка флажка Ограничить доступ к странице позволит создать страницу, наполнить ее содержанием и добавить в меню, но ограничить доступ для определенных групп пользователей, например, всех пользователей, кроме разработчиков сайта. Функция удобна, если создаваемая страница должна наполняться содержанием несколькими людьми или с течением времени перед ее публикацией на сайте.

Ссылка Настроить группы редакторов по умолчанию позволяет быстро перейти в Административный раздел для редактирования прав доступа конкретных групп.

Второй шаг (рис. 30).

Задание пункта меню для страницы

Мастер создания новой страницы

Создание новой страницы в разделе /
[Создать новую страницу в Панели управления](#)

Имя нового пункта:

Тип меню:

Вставить перед пунктом:

< Назад Далее > Готово Отмена

Рис. 30. Создание страницы

Имя нового пункта - название, под которым страница будет отображаться в меню.

Тип меню - меню в которое будет добавлена страница

Вставить перед пунктом - в выпадающем списке существующих списков меню выберите нужный пункт.

Третий шаг (рис. 31).

Задание свойств страницы. Начальные значения свойств страницы наследуются из свойств раздела, в котором она создана. При необходимости можно поменять эти значения.

Мастер создания новой страницы

Создание новой страницы в разделе /
[Создать новую страницу в Панели управления](#)

Свойства страницы

Описание страницы:

Ключевые слова:

Заголовок окна браузера:

Продвигаемые слова:

ROBOTS:

Теги страницы

Теги:

< Назад Далее > Готово Отмена

Рис. 31. Создание страницы

Поле Описание - описание страницы.

Поле Ключевые слова - ключевые слова страницы, необходимы для поисковых машин.

Поле Теги - теги для поиска страницы. Функция активна при установленном флажке правее поля.

После этого шага нужно нажать на кнопку Готово, мастер перейдет к редактированию страницы.

Четвертый шаг

Наполнение страницы содержанием.

Добавьте на страницу нужную статическую информацию или компоненты (рис. 32).

Создание страницы по шаблону удобно если стоит задача создания многих в целом однотипных страниц. Но требуется чтобы администратор создал такие шаблоны заранее. В этом случае создание страницы производится по команде Создать страницу > По шаблону > нужный шаблон. В этом случае проходятся все шаги мастера создания страницы, только на этапе редактирования страницы будет отображён текст, добавленный администратором в шаблон.

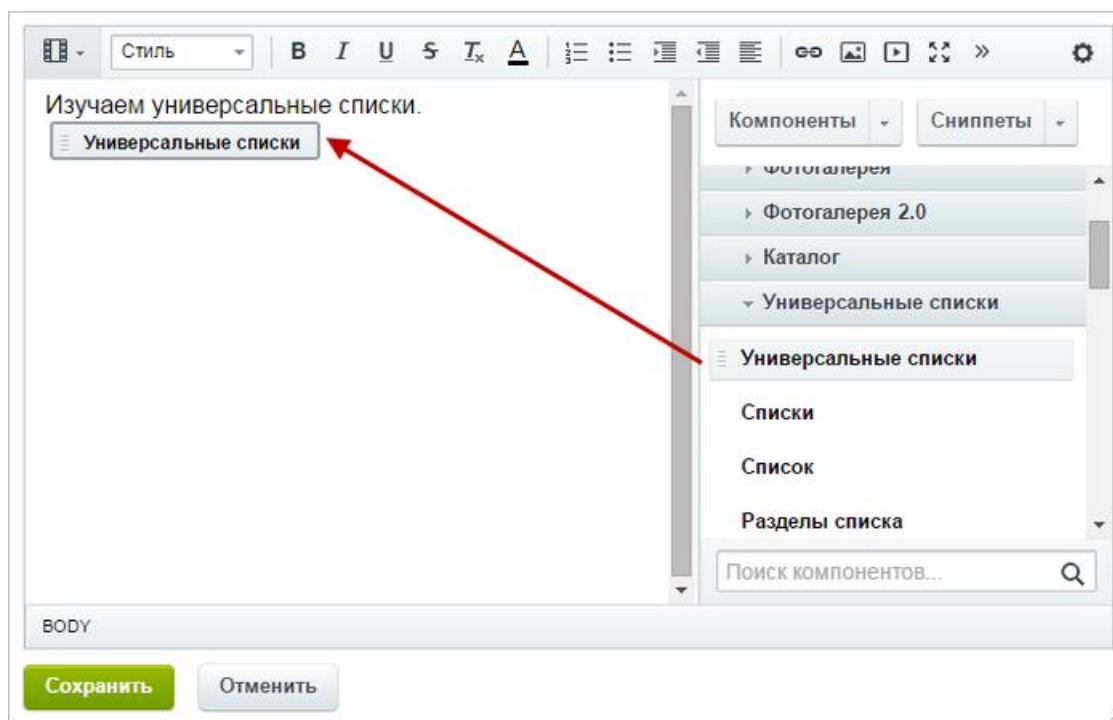


Рис. 32. Создание страницы в визуальном редакторе

Создание страниц через документооборот доступно в "1С-Битрикс: Управление сайтом" начиная с редакции "Бизнес". Использование этой функции имеет ограничения: нельзя сразу назначить странице пункт меню, нельзя задать ключевые слова и теги, недоступно использование автоматической транслитерации.

Для создания новой страницы через Документооборот воспользуйтесь командой Через документооборот кнопки Создать страницу в Пуб-

личной части (либо кнопкой **Добавить** на контекстной панели страницы **Контент > Документооборот > Документы** в Административной части). Откроется форма создания страницы (в Административном отделе системы) как показано на рис. 33.

Поле **Статус** - выбирается текущий статус страницы. Для всеобщего доступа документ должен иметь статус **Опубликован**.

Поле **Полное имя файла** - имя файла и путь до него в структуре файловой системы сайта. Возможно изменение и имени и пути.

Поле **Заголовок** - заголовок страницы.

Поле для ввода содержания страницы. Можно использовать визуальный редактор или вводить текст в текстовом формате или тегах HTML.

Рис. 33. Создание страницы через документооборот

Удаление страницы производится с помощью кнопки **Изменить страницу** (рис. 34):

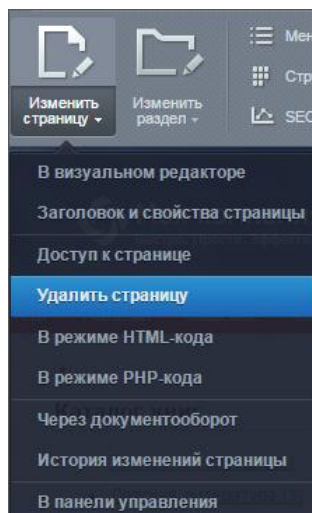


Рис. 34. Удаление страницы

После того, как вы кликните пункт Удалить страницу, появится окно подтверждения (рис. 35):

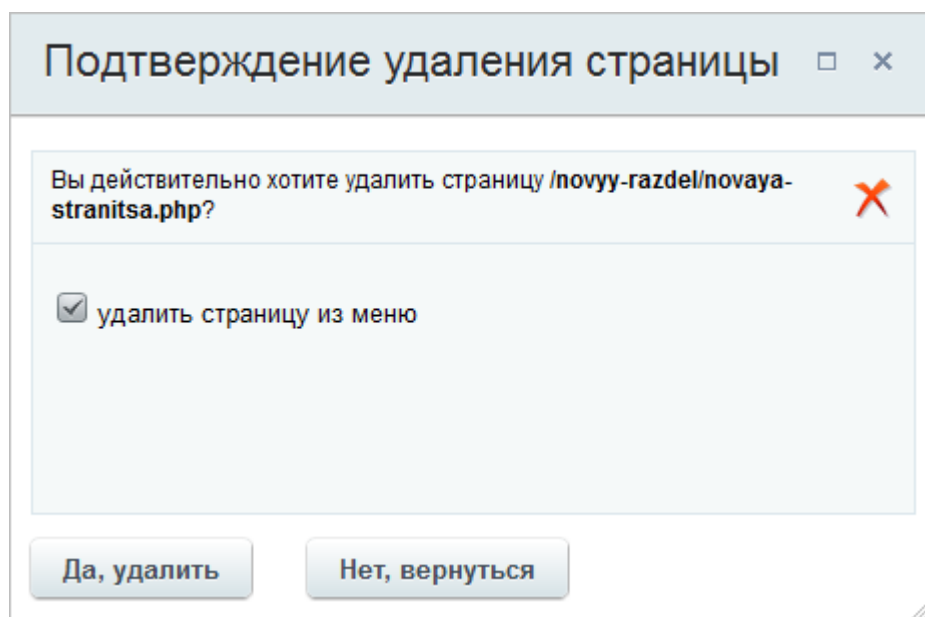


Рис. 35. Подтверждение удаления страницы

Если вы снимите галочку удалить страницу из меню, то будет удален только файл .php страницы из структуры сайта, однако пункт в меню сайта останется и при переходе по нему будет показываться 404 ошибка, пока не будет создан новый файл для данной страницы.

Удалить страницу можно и из административного раздела с помощью контекстного меню.

Удаление индексной (главной) страницы может привести к недоступности всего раздела.

Создание раздела производится с помощью кнопки (рис. 36).

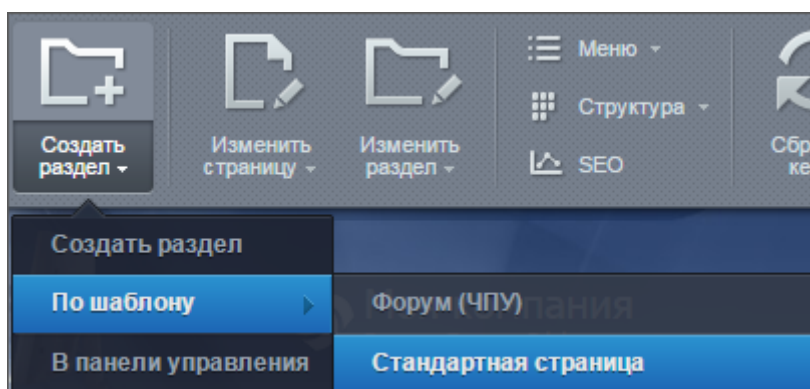


Рис. 36. Создание раздела

Верхняя часть кнопки сразу запускает мастер создания раздела. Нижняя часть позволяет выбрать создание раздела через мастер, либо в панели управления. Контент-менеджерам рекомендуется использовать мастер.

Первые два и последний шаги мастера создания раздела аналогичны мастеру создания страницы.

Третий шаг

Задание свойств раздела. Значения свойств страницы наследуются из раздела, в котором она создана (рис. 37).

Рис. 37. Форма настроек раздела

- Поле Описание страницы - дается описание раздела.
- Поле Ключевые слова - введите ключевые слова, которые будут использоваться поисковыми машинами.
- Поле Заголовок окна браузера - укажите текст, который будет выводиться в качестве заголовка окна браузера на страницах раздела.
- Поле Продвигаемые слова - слова, по которым будет анализироваться раздел инструментами модуля SEO.
- Поле ROBOTS - указания на индексацию и следование по ссылкам для поисковых машин.

При создании раздела автоматически создается страница index.php, которая и отображается на сайте как главная страница раздела. Удаление этой страницы приведет к тому, что раздел станет недоступен для просмотра.

УПРАВЛЕНИЕ ДИНАМИЧЕСКОЙ ИНФОРМАЦИЕЙ

Вывод динамичной информации из базы данных в Bitrix Framework осуществляется в основном с помощью информационных блоков, т.е. не-

которого отображения таблиц базы данных в информационной модели сайта. Создавая сайт необходимо продумать структуру информационных блоков. Рассмотрим пример простого использования информационного блока на примере каталога.

Для создания каталога товаров нам потребуется 2 инфоблок: инфоблок товаров и инфоблок торговых предложений. Их можно хранить как в одном типе, так и в разных. Мы будем хранить в разных, т.е. создадим тип инфоблоков для каталога основных товаров (Каталоги) и отдельный тип для торговых предложений (Торговые предложения).

Примечание: при создании реального проекта желательно удалить все демо-данные: типы инфоблоков и сами инфоблоки (мнение опытного разработчика).

Перейдите в административную часть на страницу Контент > Инфоблоки > Типы инфоблоков. Откроется список типов информационных блоков, имеющихся в системе (рис. 38):

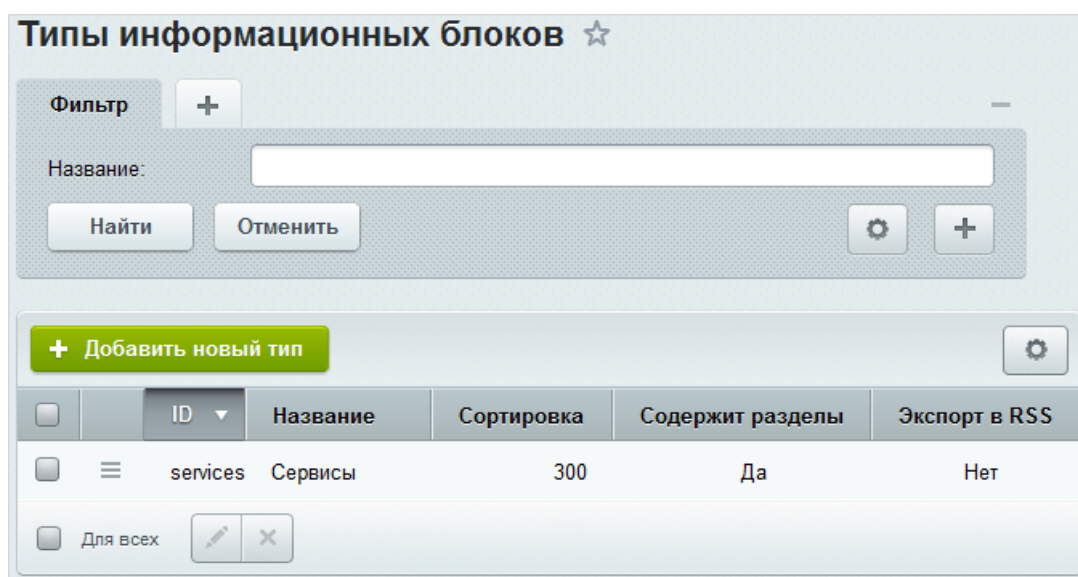


Рис. 38. Список типов инфоблоков в системе

Сперва создадим тип инфоблока основных товаров – Каталоги. Для этого нажмите кнопку Добавить новый тип, расположенную на контекстной панели. Откроется форма создания нового типа инфоблока (рис. 39):

Основное Дополнительно

Настройки типа

Идентификатор (ID):

Использовать древовидный классификатор элементов по разделам: ☒

Языкозависимые названия и заголовки объектов:

Язык	Название	Разделы	Элементы
Russian:	<input type="text" value="Каталоги"/>	<input type="text" value="Разделы"/>	<input type="text" value="Товары"/>
English:	<input type="text" value="Catalog"/>	<input type="text" value="Sections"/>	<input type="text" value="Products"/>

Сохранить Применить Отменить

Рис. 39. Создание нового типа инфоблока

На закладке Основное укажите символьный код типа инфоблоков в поле Идентификатор (ID). Код - это произвольный набор латинских букв и символов, понятный администратору сайта. В нашем примере это будет catalog.

Использование транслита в наименованиях может порождать огромное количество проблем при командной разработке, так как программисты транслитерируют термины по-разному. Лучше использовать английские названия - так любой разработчик может посмотреть в словаре значение этого термина, если он его не знает.

Флажок у поля Использовать древовидный классификатор элементов по разделам определяет, будет ли структура создаваемого типа инфоблока древовидной. Отказ от древовидной схемы требуется в крайне редких случаях.

Введите Название типа информационных блоков для русского языка и англоязычной версии.

Даже если вы не планируете использование других языковых версий вашего сайта, вам все равно необходимо заполнить колонку Название для всех имеющихся языковых интерфейсов сайта.

Колонки Разделы и Элементы необязательны для заполнения. Но ввод в эти поля терминов позволяет сделать работу с каталогом более удобной, так как можно задать более соответствующие и понятные названия для будущих редакторов сайта.

Сохраните внесенные изменения. Форма закроется, система перейдет к списку имеющихся типов инфоблоков, среди которых отразится и вновь созданный тип инфоблока.

Аналогично создается тип инфоблоков Торговые предложения с идентификатором, например, offers.

Для создания каталога товаров нам потребуется два инфоблока: инфоблок товаров и инфоблок торговых предложений. Их можно хранить как в одном типе, так и в разных. Мы будем хранить в разных, т.е. создадим тип инфоблоков для каталога основных товаров (Каталоги) и отдельный тип для торговых предложений (Торговые предложения).

Перейдите в административную часть на страницу Контент > Инфоблоки > Типы инфоблоков. Откроется список типов информационных блоков, имеющихся в системе, как показано на рисунке 38.

Сперва создадим тип инфоблока основных товаров – Каталоги. Для этого нажмите кнопку Добавить новый тип, расположенную на контекстной панели. Откроется форма создания нового типа инфоблока (рис. 39).

На закладке Основное укажите символьный код типа инфоблоков в поле Идентификатор (ID). Код - это произвольный набор латинских букв и символов, понятный администратору сайта. В нашем примере это будет catalog.

Использование транслита в наименованиях может порождать огромное количество проблем при командной разработке, так как программисты транслитерируют термины по-разному. Лучше использовать английские названия - так любой разработчик может посмотреть в словаре значение этого термина, если он его не знает.

Флажок у поля Использовать древовидный классификатор элементов по разделам определяет, будет ли структура создаваемого типа инфоблока древовидной. Отказ от древовидной схемы требуется в крайне редких случаях.

Введите Название типа информационных блоков для русского языка и англоязычной версии.

Даже если вы не планируете использование других языковых версий вашего сайта, все равно необходимо заполнить колонку Название для всех имеющихся языковых интерфейсов сайта.

Колонки Разделы и Элементы необязательны для заполнения. Но ввод в эти поля терминов позволяет сделать работу с каталогом более удобной, так как можно задать более соответствующие и понятные названия для будущих редакторов сайта.

Сохраните внесенные изменения. Форма закроется, система перейдет к списку имеющихся типов инфоблоков, среди которых отразится и вновь созданный тип инфоблока.

Аналогично создайте тип инфоблоков Торговые предложения с идентификатором, например, offers.

Перейдем на закладку Поля разделов и для разделов сделаем обязательным к заполнению поле Символьный код (рис. 40):

Рис. 40. Настройка разделов инфоблока

На закладке Торговый каталог отметим, что созданный инфоблок является торговым каталогом и имеет торговые предложения (рис.41):

ID	Название	Тип	Акт.	Множ.	Обяз.	Сорт.	Код	Изм.	Удал.
		Строка	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	500			...
		Строка	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	500			...
		Строка	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	500			...

Рис. 41. Настройка торгового каталога

В качестве инфоблока торговых предложений указываем новый инфоблок типа Торговые предложения с названием Напольные покрытия (предложения).

На закладке Доступ мы указываем полный доступ для администраторов, право на чтение для зарегистрированных пользователей и обычных посетителей, а администраторам интернет-магазина даем право на внесение изменений (рис. 42):

Доступ по умолчанию	
Для всех пользователей [2]:	Чтение

Доступ для групп пользователей	
Администраторы [1]:	Полный доступ (изменение прав доступа)
Зарегистрированные пользователи [5]:	Чтение
Пользователи, имеющие право голосовать за рейтинг [3]:	По умолчанию
Пользователи имеющие право голосовать за авторитет [4]:	По умолчанию
Администраторы интернет-магазина [6]:	Изменение
Контент-редакторы [7]:	По умолчанию

Рис. 42. Настройка прав доступа к инфоблоку

Сохраняем внесенные данные. В результате у нас создадутся два инфоблока

Откроем форму редактирования инфоблока Напольные покрытия (предложения) и на закладке Доступ укажем такие же права доступа, как и для инфоблока товаров - Напольные покрытия.

На странице Магазин > Каталог товаров > Напольные покрытия > Свойства товаров производится создание и настройка свойств элементов каталога. Именно здесь вы задаете параметры, описанные в ТЗ на элемент товара. Чтобы понять, какие свойства нам задавать в этой странице, вспомним само ТЗ и в скобках каждого пункта объясним, какими средствами системы мы будем его решать.

Таким образом, нам необходимо создать следующие свойства:

- артикул;
- новинка;
- производитель;
- толщина;
- размер доски.

Список свойств элементов будет выглядеть так, как показано на рисунке.

Вы можете отредактировать порядок сортировки свойств необходимым для вас образом, чтобы изменить положение свойств между собой. На

странице товара свойства будут выводиться в соответствии с заданной сортировкой (рис. 43).

Напольные покрытия: Свойства ☆

Фильтр +

Название:

Найти Отменить ⚙️ +

+ Добавить свойство

<input type="checkbox"/>		ID	Название	Символьный код	Тип	Сортировка ▲	Активность	Обязательное
<input type="checkbox"/>	≡	45	Артикул	ARTNUMBER	Строка	100	Да	Нет
<input type="checkbox"/>	≡	46	Новинка	NEWPRODUCT	Список	200	Да	Нет
<input type="checkbox"/>	≡	47	Производитель	MANUFACTURER	Строка	300	Да	Нет
<input type="checkbox"/>	≡	48	Толщина	DEPTH	Строка	400	Да	Нет
<input type="checkbox"/>	≡	49	Размер доски	SIZE	Строка	500	Да	Нет

☐ Для всех - действия -

Рис. 43. Список свойств инфоблока

Для добавления свойства нажмите кнопку Добавить свойство, расположенную на контекстной панели. В открывшейся форме укажите параметры свойства (рис. 44):

Свойство

Настройки свойства

ID: Новое

Тип: Строка

Свойство активно: ☒

Порядок сортировки: 100

Название: Артикул

Символьный код: ARTNUMBER

Множественное: ☐

Обязательное: ☐

Значения свойства участвуют в поиске: ☒

Выводить на странице списка элементов поле для фильтрации по этому свойству: ☒

Выводить поле для описания значения: ☐

Количество полей для ввода новых множественных значений: 5

Подсказка:

Показывать на странице редактирования элемента: ☒

Показывать в умном фильтре: ☐

Вид в умном фильтре: Флажки

Показать развёрнутым: ☐

Размер поля для ввода значения (Строк x Столбцов): 1 x 30

Значение по умолчанию:

Сохранить Применить Отменить

Рис. 44. Настройка параметров свойства инфоблока

В поле Тип выберите Строка.

В поле Название введите название свойства – Артикул.

В поле Символьный код введите ARTNUMBER.

Символьный код - мнемонический код свойства, используемый системой при работе с ним. Может состоять из латинских символов и цифр, но не должен начинаться с цифры.

Отметьте опцию Значения свойства участвуют в поиске, чтобы можно было выполнять поиск товаров по значениям данного свойства.

Отметьте опцию Выводить на странице списка элементов поле для фильтрации по этому свойству, чтобы можно было фильтровать список товаров по этому свойству.

Проверьте, чтобы опция Показывать на странице редактирования элемента была отмечена (если нет, то отметьте ее), поскольку свойство общее для всех типов товаров.

Остальные параметры оставьте без изменений.

Сохраните свойство.

Данное свойство будет использоваться для пометки новых товаров в магазине. В форме создания свойства укажите его параметры следующим образом:

В поле Тип выберите Список.

В поле Название введите Новинка.

В поле Символьный код введите NEWPRODUCT.

Аналогично, как и для свойства Артикул, отметьте опции:

Значения свойства участвуют в поиске

Выводить на странице списка элементов поле для фильтрации по этому свойству

Показывать на странице редактирования элемента

В секции Значения списка добавьте значение да с XML_ID=Y (рис. 45):

Значения списка:				
ID	XML_ID	Значение	Сорт.	Умолч.
		(нет значения по умолчанию)		<input checked="" type="radio"/>
27	<input type="text" value="Y"/>	<input type="text" value="да"/>	<input type="text" value="500"/>	<input type="radio"/>
	<input type="text"/>	<input type="text"/>	<input type="text" value="500"/>	<input type="radio"/>

Рис. 45. Настройка свойства типа «Список»

В настройках свойства значение по умолчанию не задаем, чтобы по умолчанию товар не считался новинкой.

Сохраните свойство.

Свойство Производитель (код MANUFACTURER) создается аналогично свойству Артикул, но дополнительно в настройках необходимо отметить опцию Показывать в умном фильтре и задать внешний вид свойства в фильтре с помощью параметров Вид в умном фильтре и Показать развернутым. В нашем случае сделаем отображение свойства свернутым при входе на страницу, а при разворачивании значения будут показаны с помощью радиокнопок (рис. 46):

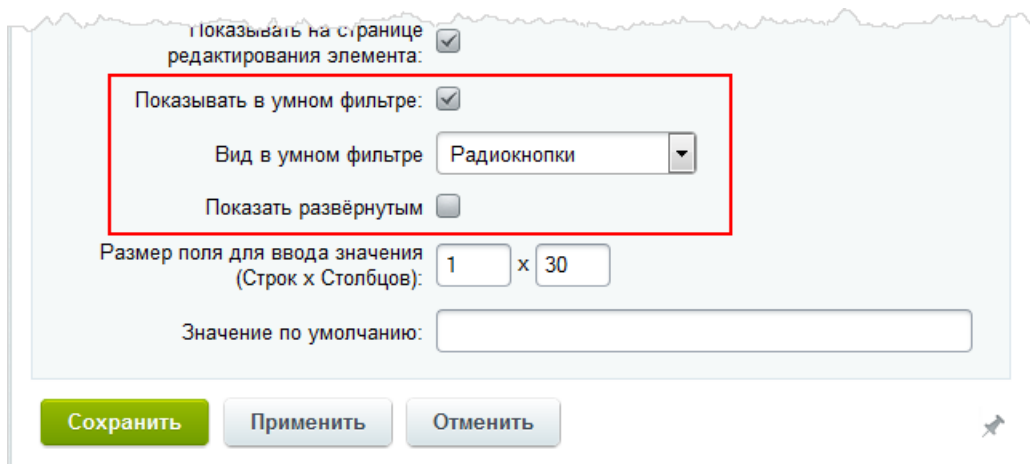


Рис. 46. Настройка отображения свойства инфоблока

Свойство Толщина (код DEPTH) - это свойство не для всех разделов, а только для разделов Линолеум и Подложка, поэтому в форме создания свойства опция Показывать на странице редактирования элемента должна быть не отмечена (рис. 47):

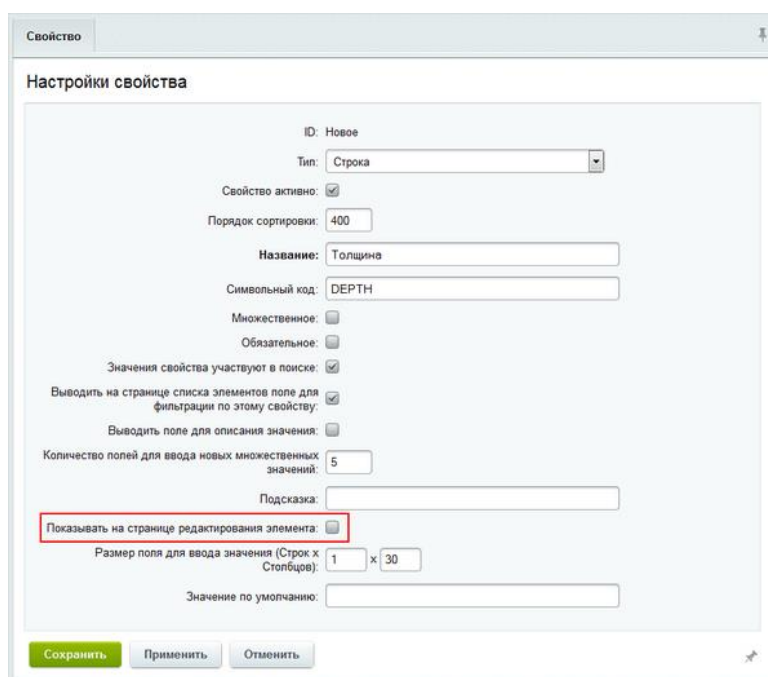


Рис. 47. Настройка отображения свойства при редактировании

Все остальные настройки выполняем аналогично свойству Артикул.

Свойство Размер доски (код SIZE) - это свойство не для всех разделов, а только для разделов Ламинат и Паркет, поэтому его создаем аналогично свойству Толщина (рис. 48):

Рис. 48. Настройка отображения свойства при редактировании

Для отображения каталога будем использовать комплексный компонент Каталог (catalog.catalog). Допустим, что у нас имеется ряд требований к отображению каталога на сайте. Перечислим их, указав в скобках какими средствами системы они решаются:

- выводить названия для страниц каталога (параметр компонента Устанавливать заголовок страницы);
- устанавливать ключевые слова, мета-описание для страниц каталога (в компоненте не должна быть настроена их установка из свойств как для списка товаров, так и для детального просмотра, чтобы данные брались из настроек SEO);
- должен быть доступен фильтр для отбора интересующих товаров (параметры в секции компонента Настройки фильтра);

– для оптовых покупателей должна показываться оптовая цена, а не розничная (параметры в секции компонента Цены);

– свойство Артикул (товаров, а не предложений) должно показываться только на странице со списком товаров (свойство должно быть выбрано в поле Свойства для списка товаров, но не выбрано в настройках детального просмотра);

– выбор необходимого торгового предложения должен быть доступен сразу при просмотре списка товаров (включить расширенную схему в параметрах компонента);

– на странице товара должна быть возможность оставить отзыв о товаре (параметр компонента Включить отзывы о товаре);

Как видим, все задачи решаются штатными настройками компонента Каталог. Рассмотрим реализацию более детально. Для этого откроем параметры компонента Каталог и выполним их настройку:

Секция Основные параметры

Укажите тип инфоблоков Каталоги и инфоблок Напольные покрытия.

Секция Дополнительные настройки

Установите флажок в поле Устанавливать заголовок страницы. Теперь заголовок страницы будет браться из поля Название соответствующего странице элемента.

Установите флажок в поле Устанавливать статус 404 если не найдены элемент или раздел. В этом случае товарный каталог покажет не сообщение на пустой странице (элемент или раздел не найден), а перебросит пользователя на 404 страницу.

Разработчик должен взять за правило: на проекте должны быть вменяемые тексты на страницах 404, 500 ошибок, а также сообщение о технических работах.

Секция Управление адресами страниц

Включите поддержку ЧПУ и настройте адреса страниц. Предварительно нужно указать адреса этих страниц в настройках инфоблока (рис. 49):

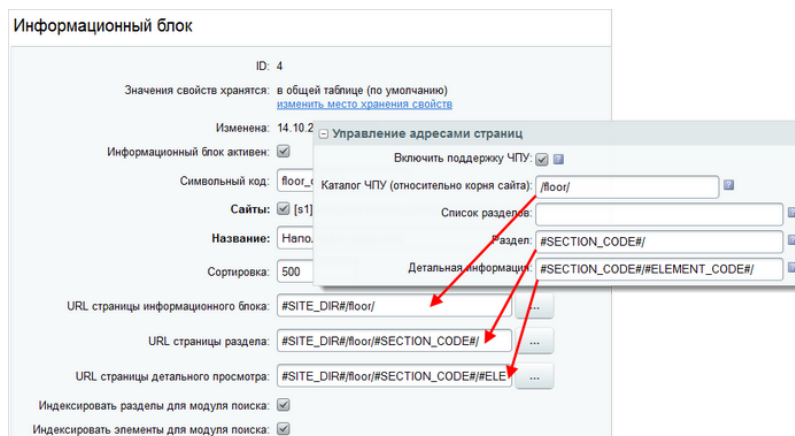


Рис. 49. Настройка адресов страниц инфоблока

Секция Настройки фильтра

Отметьте опцию Показывать фильтр и в поле Тип цены выберите розничную цену, чтобы покупатели могли отбирать товары по цене. Остальные параметры группы оставьте по умолчанию. В фильтре будут отображаться те свойства товаров (и простых, и с предложениями), в настройках которых был задан показ в умном фильтре.

Секция Цены

Обязательно укажите типы цен: розничную и оптовую.

Секция Добавление в корзину

Правильно укажите адрес корзины и отметьте свойства, значения которых должны отображаться в корзине.

Секция Настройки списка

В поле Свойства выберите Артикул, Производитель, Толщина и Размер доски, а в поле Свойства предложений - Цвет, Ширина и Количество полос.

Поля Установить ключевые слова страницы из свойства раздела, Установить описание страницы из свойства раздела и Установить заголовок окна браузера из свойства раздела должны быть незаполненными, поскольку метаданные формируются в соответствии с SEO настройками.

Секция Настройки детального просмотра

В поле Свойства выберите Производитель, Толщина и Размер доски, а в поле Свойства предложений - Артикул, Цвет, Ширина и Количество полос.

Аналогично, как и для списка, поля Установить ключевые слова страницы из свойства раздела, Установить описание страницы из свойства раздела и Установить заголовок окна браузера из свойства раздела должны быть незаполненными.

Секция Дополнительно

В поле Схема отображения выберите расширенный.

В поле Свойства для отбора предложений отметьте Цвет, Ширина и Количество полос.

Отметьте опцию Включить отзывы о товаре. Станут доступными настройки отзывов: отметьте опцию Использовать комментарии и задайте название для блога латинскими буквами.

Настройка шаблона каталога выполняется аналогично шаблону меню, рассмотренному выше.

Примерный код шаблона каталога:

```
<?if(!defined("B_PROLOG_INCLUDED"))||B_PROLOG_INCLUDED!==true)die();  
$this->setFrameMode(true);?>  
<?if($arParams["DISPLAY_TOP_PAGER"])?><?=$arResult["NAV_STRING"]?>  
<br /><?endif;?>  
<table cellpadding="0" cellspacing="0" border="0">  
<?foreach($arResult["ITEMS"] as $cell=>$arResult):?>  
<?<$this->AddEditAction($arResult['ID'], $arResult['EDIT_LINK'],
```



```

CIBlock::GetArrayByID($arParams["IBLOCK_ID"], "ELEMENT_EDIT"));
$this->AddDeleteAction($arResult['ID'], $arResult['DELETE_LINK'],
CIBlock::GetArrayByID($arParams["IBLOCK_ID"], "ELEMENT_DELETE"),
array("CONFIRM" => GetMessage('CT_BCT_ELEMENT_DELETE_CONFIRM')));
$arResult["DETAIL_PAGE_URL"] =
$arResult["DISPLAY_PROPERTIES"]["URL"]["VALUE"];?><?if($cell%$arParams["LINE_ELEMENT_COUNT"] == 0):?><tr><?endif;?><td valign="top"
width="<?=>round(100/$arParams["LINE_ELEMENT_COUNT"])?>%"
id="<?=$this->GetEditAreaId($arResult['ID']);?>"
<table cellpadding="0" cellspacing="2" border="0">
<tr><?if(is_array($arResult["PREVIEW_PICTURE"])):?>
<td valign="top">
<a href="<?=$arResult["DETAIL_PAGE_URL"]?>"
target="_blank">
"
width="<?=$arResult["PREVIEW_PICTURE"]["WIDTH"]?>"
height="<?=$arResult["PREVIEW_PICTURE"]["HEIGHT"]?>" alt="<?=$
arResult["PREVIEW_PICTURE"]["ALT"]?>"
title="<?=$arResult["PREVIEW_PICTURE"]["TITLE"]?>"
/></a><br /></td>
<?elseif(is_array($arResult["DETAIL_PICTURE"])):?>
<td valign="top">
<a href="<?=$arResult["DETAIL_PAGE_URL"]?>" target="_blank">
"
width="<?=$arResult["DETAIL_PICTURE"]["WIDTH"]?>"
height="<?=$arResult["DETAIL_PICTURE"]["HEIGHT"]?>"
alt="<?=$arResult["DETAIL_PICTURE"]["ALT"]?>"
title="<?=$arResult["DETAIL_PICTURE"]["TITLE"]?>"
/></a><br /></td><?endif?>
<td valign="top"><a href="<?=$arResult["DETAIL_PAGE_URL"]?>" tar-
get="_blank"><?=$arResult["NAME"]?></a><br /><br />
<?=$arResult["PREVIEW_TEXT"]?></td></tr></table>&nbsp;</td>
<?<math>\$cell++</math>;
if($cell%$arParams["LINE_ELEMENT_COUNT"] == 0):?>
</tr>
<?endif?>
<?endforeach; // foreach($arResult["ITEMS"] as $arResult):?>
<?if($cell%$arParams["LINE_ELEMENT_COUNT"] != 0):?>
<?while(($cell++)%$arParams["LINE_ELEMENT_COUNT"] != 0):?>
<td>&nbsp;</td>
<?endwhile;?></tr><?endif?></table>
<?if($arParams["DISPLAY_BOTTOM_PAGER"]):?>
<br /><?=$arResult["NAV_STRING"]?>
<?endif;?>
</div>

```

Библиографический список

1. Гоше Х.Д. HTML5. Для профессионалов. – СПб.: Питер, 2014. – 560 с.
2. Макфарланд Д. Большая книга CSS3. – СПб.: Питер, 2016. – 608 с.
3. Фримен Э., Робсон Э. Изучаем программирование JavaScript. – СПб.: Питер, 2016. – 640 с.
4. Шапошников И. Справочник Web-мастера. XML. – М.: Книга По Требованию, 2012. – 298 с.
5. Котеров Д., Костарев А. PHP5. – СПб.: БХВ-Петербург, 2014. – 1104 с.
6. Маклафлин Б. PHP и MySQL. Исчерпывающее руководство. – СПб.: Питер, 2016. – 544 с.
7. Прохоренок Н.А. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера. – 3-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2010. – 912 с.
8. Бейли Л. Изучаем PHP и MySQL / Линн Бейли, Майкл Моррисон; [пер. с англ.]. – М. : Эксмо, 2010. – 800 с.
9. Квинт И. HTML, XHTML и CSS на 100 %. — СПб.: Питер, 2010. – 384 с.
10. Расторгуев О. Сайт на 1С-Битрикс. Создание, поддержка и продвижение. Базовое практическое руководство. – М.: Наука и техника, 2012. – 256 с.
11. 1С-Битрикс, документация. Электронный ресурс. <http://dev.1c-bitrix.ru/docs/>