Лясин Д.Н., Абрамова О.Ф.

Основы проектирования Web-приложений

Волжский

2019

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ВОЛЖСКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ (ФИЛИАЛ)
ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО БЮДЖЕТНОГО ОБРАЗОВАТЕЛЬНОГО
УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОЛГОГРАДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Д.Н. Лясин, О.Ф. Абрамова

Основы проектирования Web-приложений

Электронное учебное пособие

2019

УДК 004.45(07)
ББК 32.973я73
Л 976

Р е ц е н з е н т ы:
канд. физ.-мат. наук, доцент кафедры экономической теории, математики и информационных систем МБОУ ВО
«Волжский институт экономики, педагогики и права»
Алпатов А. В.,
канд. физ.-мат. наук, зав. кафедрой математики Волжского филиала ФГАОУ ВО ВолГУ, доцент
Полковников А.А.

Издается по решению редакционно-издательского совета
Волгоградского государственного технического университета

Лясин, Д.Н.
Основы проектирования Web-приложений [Электронный ресурс] : учебное пособие / Д.Н. Лясин, О.Ф. Абрамова ; ВПИ (филиал) ВолгГТУ, – Электрон. текстовые дан. (1 файл: 1,84 МБ). – Волжский, 2019. – Режим доступа: http://lib.volpi.ru. – Загл. с титул. экрана.
ISBN 978-5-9948-3303-2

Учебное пособие знакомит читателя с основами разработки приложения для работы в среде Web. В пособии рассматриваются основные технологии, лежащие в основе функционирования данного типа приложений, дан обзор языков представления данных и программирования, являющихся основным инструментом для разработчиков и верстальщиков. Первая часть пособия посвящена обзору базовых принципов работы приложений в глобальной сети, описываются протокол HTTP как базовый транспорт данных между составляющими Web-приложений, языки описания и представления данных HTML и CSS, технология DHTML динамического изменения состояния документов в браузере.

Материал пособия не требует от читателя предварительных знаний в Web-разработке, понятийный аппарат, инструментарий и подходы к разработке данной сферы программирования постепенно объясняются и иллюстрируются, что позволяет даже начинающему разработчику вникнуть в тему и получить знания, позволяющие создавать современные и эффективные Web-приложения.

Предназначено для студентов, обучающихся по направлениям 09.03.01 «Информатика и вычислительная техника» и 09.03.04 «Программная инженерия» в рамках курса «Основы проектирования Web-приложений».

Ил. 12, табл. 8, библиограф.: 20 назв.

ISBN 978-5-9948-3303-2

Оглавление

Введение **4**
1. Основные понятия Web-приложений **5**
1.1. Всемирная паутина World Wide Web **7**
1.2. Адресация ресурсов в World Wide Web **8**
1.3. Протокол передачи данных HTTP **11**
1.4. Особенности функционирования Web-приложений **18**
2. Технологии разработки Web-приложений **21**
2.1. Язык разметки HyperText Markup Language **21**
2.2. Каскадные таблицы стилей (Cascading Style Sheets, CSS) **40**
2.3. Динамическая обработка данных в HTML-документах **68**
Заключение **98**
Список литературы **99**

Введение

Всемирная паутина прочно вошла в жизнь современного общества и отдельного человека. Являясь источником информации, средством коммуникации, каналом получения важных сервисов, она стала востребованной и доступной, всепроникающей и зачастую надоедливой. Web пришел к нам стремительно и навсегда. В связи с этим представить сегодня информационные технологии без Web-составляющей практически невозможно, информационные системы задумываются, проектируются, разворачиваются и эксплуатируются в реалиях глобальной сети; за относительно недолгий срок существования сети Интернет появились и продолжают развиваться технологии, инструменты, методики разработки приложений для работы в Web-среде. В связи с этим знакомство с принципами функционирования глобальной сети, а также разработки приложений для Web можно назвать актуальным и необходимым, и именно поэтому было написано настоящее учебное пособие.

В первой части пособия можно познакомиться с базовыми технологиями, лежащими в основе Web: протоколом HTTP, URI-адресацией, языками HTML и CSS, DHTML и скриптами клиентской стороны. Эти сведения полезны сами по себе, поскольку позволяют, например, осуществлять верстку HTML-документов, без которой трудно представить работу Web-приложения, а также являются фундаментом для изучения технологий разработки скриптов серверной стороны приложений, проектирования архитектуры Web-приложений, вопросов обеспечения безопасности и надежности их работы, которые будут рассмотрены в следующих частях пособия.

1. Основные понятия Web-приложений

Начиная разговор о Web-приложениях, необходимо, прежде всего, определиться с идентификацией этого понятия, определить те грани, которые отделяют этот тип приложений от всех прочих. Современные приложения можно классифицировать по множеству критериев: они делятся на *прикладные* (решающие задачи некоторой конкретной предметной области) и *системные* (обеспечивающие функционирование самой вычислительной системы, ее диагностику или управление ею), *монолитные* (содержащие весь необходимый функционал в одном программном модуле) и *распределенные* (в которых общий функционал разнесен по нескольким программным модулям, которые в свою очередь могут быть размещены в различных узлах вычислительной сети), *одно-* и *многопользовательские* (в зависимости от количества пользователей, которые имеют доступ к приложению), *нативные* (написанные для конкретной программно-аппаратной платформы) и *кроссплатформенные* (способные выполняться на различных типах вычислительных устройств и версиях операционных систем). Если описывать современные Web-приложения, то их можно отнести к многопользовательским распределенным приложениям, работающим по технологии «клиент-сервер» с использованием программы-браузера в качестве основной среды коммуникации с пользователем.

Современное Web-приложение строится как симбиоз множества технологий, протоколов, методологий и инструментов, но можно выделить несколько базовых компонентов, лежащих в основе его функционирования:

- *сеть Интернет* (*Интранет*), которая является основной средой передачи данных между компонентами Web-приложения;

- *протокол HTTP* (*Hyper Text Transfer Protocol*), который формулирует основные правила обмена информацией между клиентской и серверной частями Web-приложения;

- *языки представления данных* (HTML, XML, CSS), которые позволяют описывать семантику передаваемых между частями Web-приложения данных, а также желаемый внешний вид при их отображении пользователю;

- *языки обработки данных*, обеспечивающие программное формирование запросов и ответов модулями Web-приложения для динамической обработки данных и действий пользователей. Примерами здесь могут служить такие языки, как JavaScript (обработка данных на стороне клиента) и PHP, Python, C#, Java, Ruby, тот же JavaScript (обработка данных на стороне сервера);

- *Web-сервер*, обеспечивающий точку подключения клиентов к Web-приложению, а также среду интерпретации HTTP-запросов, запуска для их обработки Web-приложения и передачи ответов клиенту;

- *Web-браузер* как наиболее популярный (но не единственно возможный) вариант реализации среды клиентской части Web-приложения, реализующей интерактивное взаимодействие с конечным пользователем.

Web-приложения относят к одному из двух типов:

- Ориентированные на представление информации. Приложения этого типа формируют на стороне сервера готовое представление данных на таких языках, как HTML, XML и CSS, задачей клиента (обычно – браузера) в этом случае является отображение этого представления на экране;
- Ориентированные на предоставление услуг (Web-сервисы или Web-службы). Подобные приложения обрабатывают запросы клиента (в качестве которой может оказаться другая Web-служба) и возвращают блоки данных в оговоренных форматах (XML, JSON), а программа-клиент самостоятельно принимает решение о способе обработки и представления полученных от сервера данных пользователю. Фактически Web-службы реализуют свой про-

токол обмена данными поверх протокола HTTP. Примерами здесь могут служить протоколы REST или SOAP.

1.1. Всемирная паутина World Wide Web

Средой функционирования Web-приложений является *Всемирная паутина* (*World Wide Web*). Она представляет собой существующую поверх всемирной сети Интернет инфраструктуру, состоящую из множества Web-серверов, хранящих огромные объемы статических и динамических страниц, связанных между собой гипертекстовыми ссылками и передающих эти документы по запросам в соответствии с протоколом HTTP. Сеть WWW является лишь одним из сервисов сети Интернет, но на сегодняшний момент данный сервис является самым востребованным и быстрорастущим в плане количества узлов и объема трафика в нем.

У Всемирной паутины в отличие от сети Интернет есть вполне конкретный автор-создатель. Основные идеи, которые легли в основу и в модернизированном, но вполне сохранившем начальные принципе виде реализованы в Web сегодня, предложил и реализовал для внутренней сети Европейского совета по ядерным исследованиям (CERN) инженер этой организации Тим Бернерс-Ли. В рамках проекта по созданию единого корпоративного хранилища документов, связанных между собой *гиперссылками* (блоками, адресующими другой, логически связанный документ хранилища) им были созданы первые Web-сервер и Web-браузер, протокол HTTP, язык разметки документов HTML, универсальные идентификаторы ресурсов в сети URI. Все эти технологии, с тех пор многократно усовершенствованные, и сейчас составляют основу WWW. Поэтому закономерно, что Тим Бернерс-Ли возглавляет *консорциум Всемирной паутины* (*WWW consortium, W3C*), организацию, которая с 1994-го года разрабатывает и внедряет все технологические обновления всемирной сети.

Основные особенности функционирования сети Интернет вообще и Всемирной паутины в частности, которые делают Web-приложения очень

удобным средством предоставления информационных сервисов для своих клиентов:

- у Сети нет владельца, она является общечеловеческим достоянием. Существуют лишь ряд институтов, осуществляющих технологическое развитие сети;

- выход из строя некоторых узлов и сегментов Сети не делает недоступными все прочие узлы;

- Сеть не знает границ и может связать компьютеры в различных точках сети;

- принципы функционирования Сети можно воспроизвести в любом масштабе, реализуя, например, закрытую корпоративную сеть на тех же принципах, которых использует вся Всемирная паутина.

1.2. Адресация ресурсов в World Wide Web

Ресурсы Web-приложений (страницы, сервисы) адресуются в сети Интернет с использованием *Uniform Resource Identifier* (URI, Унифицированный идентификатор ресурса). URI представляет собой последовательность символов, уникально идентифицирующую физический или виртуальный ресурс, доступный через сеть Интернет. Чаще всего используются две разновидности URI: *Uniform Resource Locator* (URL, Унифицированный локатор ресурса) и *Uniform Resource Name* (URN, Унифицированное имя ресурса). Разница между ними заключается в том, что первый указывает, где найти ресурс, второй – как его идентифицировать.

Обобщенно URI выглядит следующим образом:

```
Схема : Базовый_ИД / Путь [? Запрос # Фрагмент ]
```

где *Схема* – протокол обращения к ресурсу (http, https, ftp, mailto …);

Базовый_ИД – глобальный идентификатор местоположения ресурса, опционально дополняемый аутентификационными данными. Структура

Базового идентификатора включает логин и пароль доступа к ресурсу, адрес ресурса (DNS или IP), порт, на котором доступен ресурс:

```
<логин>:<пароль>@<хост>:<порт>
```

Путь – уточняет путь к ресурсу в рамках хоста;

Запрос – конкретизирует данные, необходимые для поиска ресурсов или обработки запроса клиента;

Фрагмент – уточняет положение вторичного ресурса в контексте ранее определенного первичного (якорь).

Примеры URL:

```
http://www.mysite.ru/catalog/?arg1=1&arg2=value
ftp://user:pass@some.com:2021/docs/reports/rep.pdf
https://somesite.com#anchor
```

URN-ссылки предназначены для идентификации самого ресурса (типа, содержимого) без указания пути, по которому его можно найти. Примерами могут служить magnet-ссылки, идентифицирующие ресурсы в пиринговых сетях:

magnet:?xt=urn:btih:E40CDFE227F9B9ED80A5CC2A151CD5BEE6B3C0A2

При определении URL-ссылок на страницах Web-приложений разделяют абсолютные и относительные ссылки. Абсолютная ссылка должна содержать полную информацию об адресуемом ресурсе, начиная с протокола и включая полный путь к документу или папке. Примеры абсолютных ссылок:

```
http://www.mysite.ru
https://mysite.com/docs
https://mysite.com/docs/doc1.html
```

Если ссылка адресует папку, то сервер обычно ищет индексный файл (index.php, index.html, default.aspx или иной, указанный в настройках Web-сервера), поэтому следующие адреса эквивалентны:

```
http://www.mysite.ru  ⇔  http://www.mysite.ru/index.html
```

Абсолютные ссылки просты в интерпретации того ресурса, который они адресуют, но они зачастую являются слишком громоздкими, а также требуют переопределения, если адресуют ресурсы этого же ресурса, а он меняет свое местоположение. Поэтому подобные ссылки чаще всего используются для адресации ресурсов сторонних Web-приложений.

Относительные ссылки отчасти решают проблемы использования абсолютных: они адресуют документ относительно текущего документа или корня сайта.

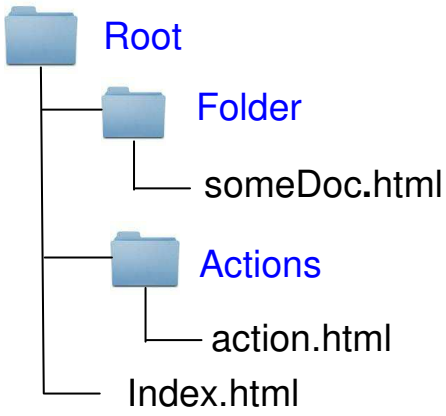


Рисунок 1.1 – Фрагмент файловой иерархии сайта

Для сайта, фрагмент файловой иерархии которого представлен на рис. 1.1, в файле index.html ссылка на документ в другой папке может выглядеть следующим образом:

```
<a href="Actions/action.html">Выполнить действие</a>
```

Здесь адрес ссылки учитывает, что целевой файл находится в папке Actions, дочерней по отношению к той, которая содержит сам файл со ссылкой. А, например, для оформления ссылки из файла someDoc.html на файл action.html необходимо указать в пути ссылку на родительскую папку «..»:

```
<a href="../Actions/action.html">Выполнить действие</a>
```

Относительные ссылки можно также формировать относительно корня сайта, а не текущего положения файла, в котором оформляется ссылка. В этом случае ссылка на файл action.html будет выглядеть одинаково во всех документах сайта:

```
<a href="/Actions/action.html">Выполнить действие</a>
```

Еще одна распространенная разновидность ссылок – так называемые ЧПУ-ссылки. Эта аббревиатура раскрывается как «Человеко-Понятный

URL». Подобная ссылка действительно более удобна для восприятия человеком, поскольку в ней используется минимум служебных символов и числовых идентификаторов, они заменяются на мнемонику текстовых строк. Рассмотрим пример. Многие Get-запросы требуют конкретизации запрашиваемых данных в области запроса:

```
http://www.mysite.ru?cat=210&type=12&id=129045
```

В данном адресе запрос URL конкретизирует характеристики отбираемых сущностей, используя некоторую внутреннюю для сайта идентификацию (параметр cat может соответствовать категории отбираемого товара, type – его конкретный тип, a id – идентификатор в базе данных). Подобный URL трудно запомнить и воспроизвести посетителю, он не может быть интерпретирован поисковыми системами, что отрицательно сказывается на продвижении сайта. Поэтому часто подобные URL заменяют ЧПУ следующего вида:

```
http://www.mysite.ru/computers/monitor/129045
```

Отсутствие служебных символов (? = &), замена идентификаторов сущностей на их строковые эквиваленты положительно сказывается на SEO-продвижении сайта и легче запоминается посетителем. Поддержка ЧПУ требует от разработчика дополнительной конфигурации ресурса, поскольку путь URL теперь не соответствует реальной иерархии каталогов файловой системы на сервере.

1.3. Протокол передачи данных HTTP

HTTP (Hypertext Transfer Protocol, Протокол передачи гипертекста) представляет собой протокол прикладного уровня семейства протоколов TCP/IP. Описание протокола HTTP можно найти в RFC-1945, RFC-2068, RFC-2069, RFC-2616, RFC-2617, RFC-7235. Протокол HTTP – самый распространенный прикладной протокол стека TCP/IP. Когда вы посещаете различные WWW-сайты с помощью браузера, браузер взаимодействует с WEB-серверами, используя именно протокол HTTP. Всякий раз при пере-

ходе по гиперсвязи от одного ресурса к другому браузер обращается к HTTP для доступа к серверу, хранящему необходимую информацию. На данный момент существует несколько версий протокола HTTP. Первой версией протокола HTTP была версия 0.9, в которой использовался простой запрос и поддерживался единственный метод GET. Таким образом, с помощью этого протокола можно было только получить документ или файл с сервера. В 1996 году была стандартизирована версия 1.1, которая в настоящий момент поддерживается большинством серверов в сети Интернет. В этих версиях используется полный запрос, и набор методов значительно расширен. В частности, можно не только получать файлы с сервера, но и передавать файлы на сервер, удалять их с сервера (естественно, при наличии определенных прав), передавать разными способами информацию специальным программам, работающим на сервере, а также управлять параметрами соединения, кэширования, вида, типа и кодировки ресурса и т.д.

Детали работы протокола HTTP относятся к техническим деталям работы стека TCP/IP и подробное знакомство с ним начинающим специалистам в области Web-разработки кажется лишней тратой времени. Однако особенности работы основного протокола обмена данными в WWW отражены в тегах HTML-верстки документов, должны учитываться при организации системы безопасности сайта, определяют доступный функционал Web-приложения, поэтому познакомимся с ним подробнее.

Взаимодействие клиента и сервера в рамках протокола HTTP описано на рис. 1.2.

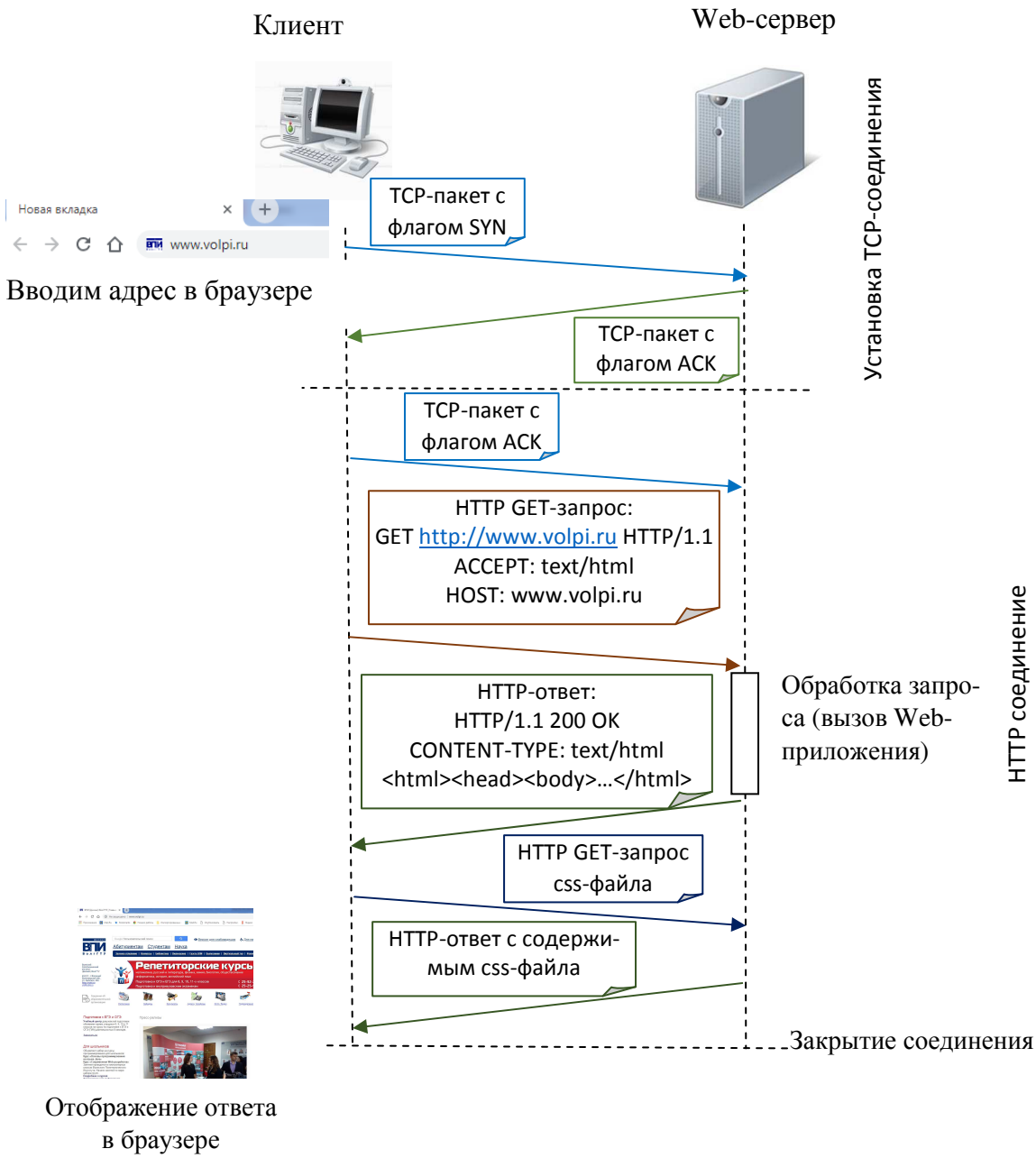


Рисунок 1.2 — Взаимодействие клиента-браузера и Web-сервера по протоколу HTTP

Начинается диалог клиента и сервера с установки типового TCP-соединения на порту web-сервера (80, 8080, …), после чего клиент посылает серверу ряд HTTP-запросов, которые сервер распознает, обрабатывает и формирует ответ клиенту в виде HTTP-ответа. При этом данные, которые вернет сервер, могут быть получены как непосредственно из содержимого запрошенного файла (типично для html-, css-, js-файлов, графических файлов), так сформированы динамически после запроса Web-приложением. В

этом случае сервер передает Web-приложению весь запрос, оно, согласно алгоритма, выполнит действия по его обработке и формированию ответа, результаты этой обработки Web-приложение направляет серверу, а тот возвращает их клиенту в виде HTTP-ответа. Взаимодействие клиента и сервера по протоколу HTTP осуществляется в режиме клиент-сервер, причем, как следует из рис. 1.2, после обмена пакетами «запрос – ответ» соединение разрывается, и следующая транзакция обмена осуществляется в рамках нового сеанса, проходящего по схеме «соединение – обмен пакетами – разрыв соединения». В связи с таким поведением протокола HTTP при Web-обмене по умолчанию не сохраняется состояние обмена, каждый новый сеанс обмен начинается с чистого листа, протокол не запоминает историю предыдущих запросов, не сохраняются ресурсы, которыми обменивались клиент и сервер в предыдущих сеансах обмена. Даже несмотря на то, что версия 1.1 протокола HTTP позволяет сохранять TCP-соединение открытым на несколько HTTP-сеансов, полагаться на то, что весь возможный цикл работы пользователя с сайтом осуществится в рамках одного соединения, не приходится. В связи с этим для Web-приложений актуальна проблема сохранения сеанса, чтобы новые запросы могли использовать данные, сформировавшиеся или передававшиеся ранее. Для этого используются такие механизмы, как сохранение данных на стороне клиента с помощью cookies или локальных хранилищ или на стороне сервера с использованием механизма сессий.

Форматы запроса и ответа HTTP-протокола строго документированы в стандартах, подробнее ознакомиться с ними можно в [1]. Все HTTP-транзакции имеют один общий формат. Каждый запрос клиента и ответ сервера состоит из трех частей: первой строки запроса (ответа), заголовка и тела. В *строке запроса* указываются: метод запроса, адрес сервера, версия протокола. Метод определяет состав и структуру заголовков запроса, вид передачи и структуру параметров запроса. Наиболее распространен-

ными методами являются «GET» и «POST». Иногда также используются методы «HEAD», «DELETE», «PUT». В спецификации HTTP1.1 добавлены некоторые новые методы, например, «OPTIONS» и «TRACE» [3].

Метод «GET» соответствует запросу некоторого ресурса от сервера. Для этого метода тело запроса пустое, параметры передаются в строке адреса. Запрашивает ресурс или сервис. Максимальный размер запроса задается параметром сервера и используемым браузером.

Метод «POST» обычно соответствует передаче данных от клиента серверу. Параметры и данные запроса передаются в теле запроса. Максимальный размер задается в настройках сервера.

Метод «PUT» используется для сохранения передаваемого ресурса на сервере.

Метод «DELETE» используется для удаления ресурса с сервера.

Метод «HEAD» отправляет лишь заголовок запроса, чтобы проверить состояние соединения.

В первой строке ответа сервера указываются версия протокола, код результата обработки запроса сервером и пояснение к нему. Первая цифра кода статуса HTTP определяет класс кода ответа, а каждый из пяти классов содержит группу значений статуса кода. Существует пять возможных значений для первой цифры (от 1 до 5):

1xx: информационные коды	запрос получен, продолжается обработка
2xx: успешные коды	данные были успешно приняты, обработаны и использованы
3xx: коды перенаправления	для выполнения запроса требуются дополнительные действия
4xx: коды ошибок клиента	запрос содержит синтаксические ошибки, либо запрошенные ресурсы не доступны
5xx: коды ошибок сервера	сервер не смог обработать или выполнить запрос

Наиболее распространенные коды статуса обработки HTTP-запросов со строками пояснения:

100 - продолжать, Continue

200 - OK

201 - создан, Created

202 - принято, Accepted

204 - нет содержимого, No Content

206 - частичное содержимое, Partial Content

300 - множественный выбор, Multiple Choices

301 - постоянно перенесен, Moved Permanently

302 - временно перемещен, Moved Temporarily

304 - не модифицирован, Not Modified

305 - необходимо использовать прокси-сервер, Use Proxy

400 - некорректный Запрос, Bad Request

401 - несанкционированно, Unauthorized

403 - запрещено, Forbidden

404 - не найден, Not Found

405 - метод не дозволен, Method Not Allowed

407 - требуется аутентификация через прокси-сервер, Proxy Authentication Required

408 - истекло время ожидания запроса, Request Timeout

413 - объект запроса слишком большой, Request Entity Too Large

414 - URL запроса слишком длинный, Request-URL Too Long

500 - внутренняя ошибка сервера, Internal Server Error

501 - не реализовано, Not Implemented

505 - не поддерживаемая версия HTTP, HTTP Version Not Supported

Получив ответ сервера, клиентская часть приложения учитывает код статуса обработки запроса для настройки способа отображения пришедших данных и планирования дальнейших действий.

Заголовок сообщения состоит из совокупности пар «Имя: Значение», которые дополнительно описывают запрос (ответ), уточняя такие характеристики, как формат и кодировку передаваемых данных, параметры кеширования данных, информацию о последнем изменении ресурса, тип используемого клиентом браузера и др.

Наконец, *тело сообщения* содержит непосредственно передаваемые между клиентом и сервером ресурсы или данные. Тело сообщения может быть пустым, если, например, вся информация, планируемая для передачи в рамках запроса, может быть размещена в строке запроса и его заголовке.

Вот как может выглядеть HTTP-запрос к серверу:

```
Host: www.volpi.ru
Authorization: Basic U2Fua292OlNlcmdleQ==
From: vpi@volpi.ru
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWeb-
Kit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safa-
ri/537.36
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/vnd.ms-excel, application/msword, */*
Accept-Language: ru
Accept-Encoding: gzip, deflate
Range: bytes=0-99
```

А так выглядит ответ сервера:

```
HTTP/1.1 200 OK
Date: Thu, 18 Oct 2018 10:15:45 GMT
Server: Apache (1.3.41)
X-Powered-By: PHP/5.6
Expires: Fri, 19 Oct 2018 07:15:44 GMT
Content-Language: ru
Content-Type: text/html; charset=utf-8
Content-Length: 3410
Пустая строка
HTML-код страницы
```

Необходимо отметить, что в протоколе HTTP весьма ограничены средства защиты передаваемых данных (шифрование, аутентификация, контроль целостности). Если Web-приложение требует наличия средств защиты передаваемого по неконтролируемой сети трафика, оно может ис-

пользовать протокол HTTPS, в котором безопасность передачи данных обеспечивается за счет использования защищенных протоколов SSL и TLS [3].

1.4. Особенности функционирования Web-приложений

При проектировании Web-приложений приходится учитывать все перечисленные условия его функционирования, что накладывает на этот тип приложений ряд ограничений и архитектурных шаблонов. Web-приложение строится по двухуровневой архитектуре «клиент-сервер» (на самом деле уровней может быть больше, учитывая, что функции сервера могут быть разнесены на несколько уровней). В качестве среды исполнения клиента выступает web-браузер, который обменивается с серверной частью данными и командами по протоколам HTTP/HTTPS. До сих пор в качестве второй стороны взаимодействия указывался web-сервер, но он служит лишь средством коммуникации клиента с серверной частью Web-приложения (так называемый *backend*). На рис. 1.3 представлен более полный механизм взаимодействия частей Web-приложения.

Как следует из схемы на рисунке, web-сервер (реализующий обмен с клиентом по протоколу HTTP) может выполнять роль посредника. Он транслирует запросы и их параметры в исполняемую среду, в которой запускается серверная часть web-приложения. Backend программным путем формирует динамические данные в ответ на запрос, это может быть HTML-страница, XML-документ, объект, представленный в предопределенном формате. Формируемые данные называются динамическими, потому что их содержание меняется во времени, может зависеть от параметров, от предыдущих действий клиента в рамках данной сессии. Среда выполнения может быть различной (CGI-сценарий, SAPI-модуль), более подробно вопрос о способах формирования динамических данных будет рассмотрен ниже. Динамические данные, сформированные web-

приложением, передаются серверу, который формирует на их основе HTTP-ответ клиенту.

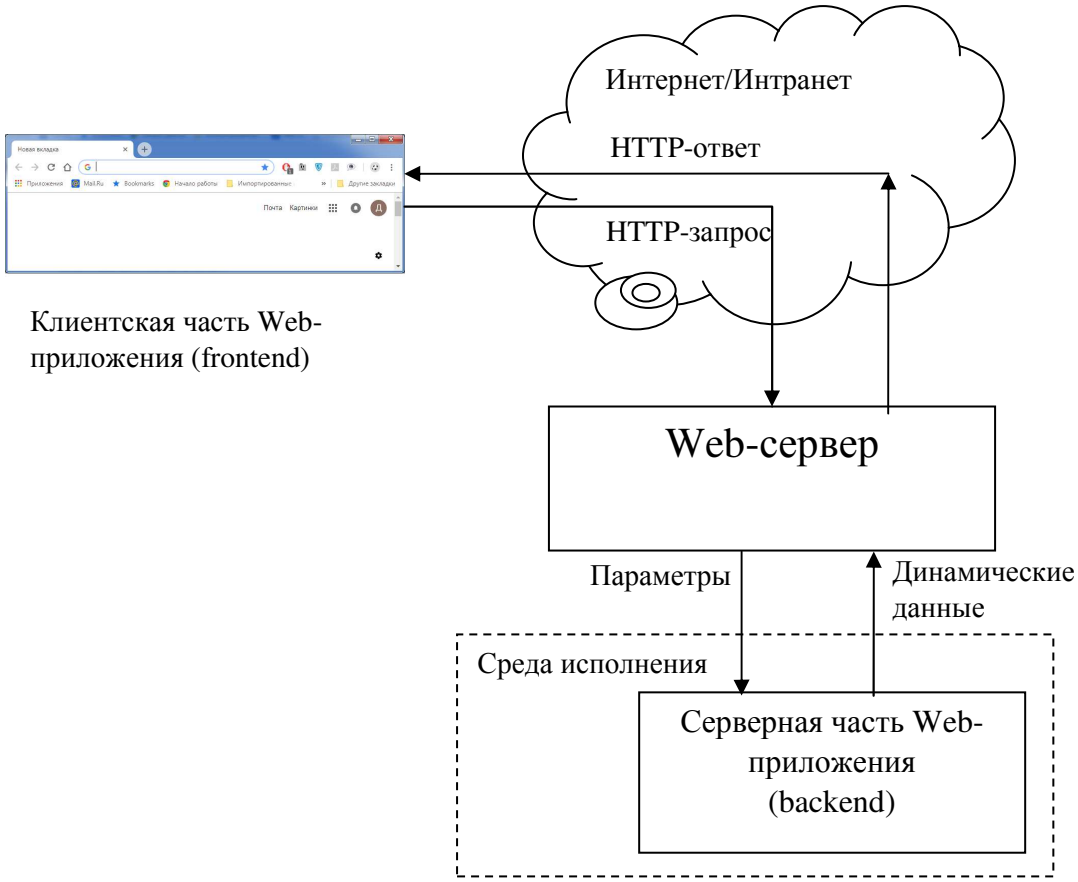


Рисунок 1.3 – Обобщенная схема взаимодействия клиентской и серверной части Web-приложения

Реализация функций корпоративных и ориентированных на конечного пользователя информационных систем в виде web-приложений весьма популярна, что обусловлено их достоинствами, а именно:

– *распределённостью*: пользователю не нужно что-либо ставить на компьютер в качестве клиентского программного обеспечения, он не привязан к конкретному месту оказания услуги, приложение доступно в любом месте, где есть гаджет с доступом в глобальную сеть и установленным браузером;

– *переносимостью*: приложение не ориентировано на конкретной тип операционной или аппаратной платформы. Браузеры, поддерживаю-

щие современные стандарты web-технологий, доступны в любой операционной среде, также реализации web-сервера разворачиваются на большинстве платформ;

— *унификацией интерфейса*: интерфейс работы программы-браузера знаком практически любому пользователю, да и интерфейсы web-страниц на сегодня достаточно де-факто стандартизованы, чтобы с функциями web-приложения смог разобраться даже неискушенный посетитель;

— *простотой установки и обслуживания*: web-приложения очень просто обновлять, они не требуют переустановки на устройствах пользователя, достаточно «перезалить» код приложения на web-сервере, и при следующем обращении клиент получает обновленную версию;

— *простота контроля использования*: тот факт, что серверная часть web-приложения размещается на сервере, не предоставляет возможности его нелегального использования, а также несанкционированного изучения кода.

У web-приложений можно указать и ряд недостатков:

— *ограничения технологий и стандартов*: при проектировании web-приложений необходимо учитывать или обходить ряд ограничений, которые накладываются используемыми протоколами. Примером здесь может служить уже упоминавшееся отсутствие поддержки состояния приложения между отдельными запросами;

— *несколько уровней программирования*: при разработке Web-приложения приходится использовать целый ряд языков разметки и программирования. Для формирования представления используются языки разметки HTML и CSS или компилируемые в них HAML, SASS, LESS, при проектировании клиентской части приложения используется Javascript, при разработке серверной части выбор шире (PHP, Java, Python, Ruby, C#, Javascript и др.), для работы с данными приходится использовать SQL;

— *отклонения от стандартов*: не все браузеры в точности соблюдают стандарты web-разработки (например, HTML5 или CSS3), что требует адаптации кода под среду отображения.

Однако преимущества web-приложений явно превалируют над недостатками, что определяет их год от года набирающую популярность, тем более для решения перечисленных проблем предлагается целый ряд готовых и весьма эффективных решений, которые будут перечислены в следующих главах.

2. Технологии разработки Web-приложений

Современное web-приложение является симбиозом различных технологий, некоторые из которых лежали в основе еще самых первых сайтов, другие появились относительно недавно и сделали процесс разработки и эксплуатации приложений удобным, а сами приложения эффективными с точки зрения потребляемых ресурсов и адаптивными к условиям эксплуатации. Рассмотрим подробнее основные технологии, лежащие в основе web-разработки.

2.1. Язык разметки HyperText Markup Language

Язык HTML (HyperText Markup Language – язык разметки гипертекста) является основным средством представления информации в виде гипертекста в Web. Язык HTML является наследником ранних попыток разработки универсального языка разметки документов (одним из представителей языков-прародителей HTML является язык Standard Generalized Markup Language – SGML). Язык HTML был разработан на рубеже 80-х – 90-х годов XX века ученым лаборатории CERN Тимом Бернерсом Ли. Основное назначение языка HTML – семантическое структурирование информации, представленной в узлах глобальной вычислительной сети, упрощающее процесс описания представляемой информации и ее объеди-

нение в связную разветвленную информационную структуру. Основной отличительной чертой HTML-документов является активное использование *гиперссылок* – специальных строк, адресующих семантически связанные документы. В итоге путем формирования связей между отдельными документами можно сформировать целую сеть из документов и ресурсов с удобными средствами навигации и поиска. С учетом того, что HTML позволяет адресовать не только текст, но и мультимедийные документы (графику, видео, аудио), можно говорить о гипермедиа-среде, в которую во многом благодаря идеям, положенным в основу языка HTML, превратился современный Интернет.

С момента первого появления в 1991-м году язык HTML претерпел существенные изменения: добавлялись новые элементы, некоторые элементы устаревали и объявлялись устаревшими, менялась идеология самого языка – постепенно из него ушли средства, отвечающие за представление информации, предпринимались попытки поставить язык разметки на рельсы строгих XML-форматов, добавлялись новые мультимедийные возможности. В итоге к текущему моменту язык HTML стал мощным и гибким инструментом, позволяющим описывать документы различной степени сложности с поддержкой мультимедийных, многопоточных, геоинформационных технологий. Перечислим основные вехи в развитии языка:

- 1991 год, выходит первая версия языка HTML, которая включала всего 20 элементов;

- 1994 год, создан консорциум W3C (World Wide Web Consortium) – организация, призванная развивать и внедрять новые стандарты Интернет (в том числе – стандарты HTML);

- 1995 год, выходит версия HTML 2.0. В язык добавлена возможность работы с формами;

- 1996 год, принята спецификация 3.0. Главное нововведение – тесная интеграция с иерархическими стилевыми спецификациями (Cascading Style Sheets, CSS);

- 1997 год, HTML 4 объявил ряд тегов устаревшими с прицелом на использование стилей CSS;

- 2000 год, принят стандарт XHTML 1.0. Правила разметки стали более строгими, соответствующими принципам языка XML (eXtensible Markup Language);

- 2014 год, опубликована спецификация HTML5. В дальнейшем планируется отказаться от цифр и расширять один общий стандарт HTML. Основная идея, заложенная в этом стандарте языка, – HTML должен декларировать семантику данных в документе, не предлагая способов их представления, так как эту задачу окончательно отвели в зону ответственности языка CSS.

HTML-документ строится как совокупность тегов, каждый из которых описывает определенную часть документа, причем не обязательно визуально представляемую в браузере. Теги заключаются в угловые скобки < > и помогают структурировать документ и интерпретировать его на стороне клиента в браузере. Обобщенный формат html-тегов:

```
<тег [атрибут1="значение" атрибут2="значение"]>
<тег [атрибут1="значение" атрибут2="значение"]>
…
</тег>
```

Вот как, например, в html-документ можно вставить картинку:

```
<img src="pict.jpg" alt="картинка" />
```

Большинство тегов присутствует в документе в паре «открывающий – закрывающий». Это означает, что тегу соответствует одноименный тег, имени которого предшествует символ **/** :

```
<p>Здесь размещаем текст параграфа</p>
```

Между открывающим и закрывающим тегом располагается содержимое HTML-элемента:

```
<title>Сайт ООО Рога и копыта</title> заголовок
<a href="/">На главную</a>  гиперссылка
<form action="/action/" method="post">…</form>
```

В стандарте языка HTML (на момент написания методических указаний актуален стандарт HTML 5.1) определены также непарные теги, для которых не требуется наличия закрывающего тега:

```
<br>
<hr>
<meta http-equiv="Content-Type" content= "text/html; char-
set=utf-8">
```

В стандарте XHTML нельзя опускать закрывающий тег, в HTML это допустимо:

```
<p>Параграф1
<p>Параграф2</p>
```

Парные теги обычно выступают в качестве контейнеров, в которые можно вкладывать другие теги, формируя таким образом иерархию владения, которая учитывается при формировании представления документа в браузере, а также динамики обработки элементов в случае использования DHTML. В приведенном ниже фрагменте html-документа все элементы, кроме span, выступают в качестве контейнеров, так как в них вложены элементы, описывающие их внутреннюю структуру.

```
<html>
  …
  <body>
    <div>
      <span> Просто текст </span>
    </div>
  </body>
</html>
```

Документ HTML всегда содержит элемент HTML, в свою очередь содержащий элементы HEAD и BODY.

Обобщенная структура html-документа выглядит следующим образом:

```
<!DOCTYPE html>      <!-- Тип документа -->
<html>               <!-- Начало документа -->
<head>               <!-- Начало заголовка документа -->
   <title>Заголовок страницы
  </title>
 </head>             <!-- Конец заголовка документа -->
 <body>              <!-- Начало тела документа -->
  <!- Тело документа -->
 </body>             <!-- Конец тела документа -->
</html>              <!-- Конец документа -->
```

Помимо названия, для каждого тега можно указать атрибуты, позволяющие уточнить функционал и расширить возможности тегов.

Атрибуты делятся на универсальные и специфические. Первые применимы практически к любому тегу ввиду общности тех параметров, которые они декларируют для элементов. Универсальные атрибуты перечислены в таблице 2.1.

Таблица 2.1. Универсальные атрибуты html-тегов

Атрибут	Описание
accesskey	Задает сочетание клавиш, предоставляющее доступ к элементу с клавиатуры
align	Задает выравнивание элемента по горизонтали
class	Указывает название класса (-ов) в используемой таблице стилевых оформлений, который (-ые) будут применены к элементу
contenteditable	Указывает, что элемент может быть отредактирован пользователем
contextmenu	Определяет контекстное меню, соответствующее элементу
data-*	Атрибуты, определяемые пользователем
hidden	Используется для сокрытия элемента при отображении документа
id	Указывает уникальный стилевой идентификатор элемента, используемый для сопоставления стиля оформления или обращения к элементу из скриптов
lang	Используется для указания браузеру на языковые особенности отображения текста
spellcheck	Включает или отключает проверку браузером правописания для содержимого элемента
style	Задает встроенное стилевое CSS-оформление элемента
tabindex	Определяет порядок передачи активности между элементами

	документа с использованием клавиши Tab
title	Определяет всплывающую подсказку при наведении курсора мыши на элемент
valing	Задает выравнивание элемента по вертикали

Пример использования универсальных атрибутов для элементов документа:

```
<div id="infoBlock" style="border: 1px solid red"
spellcheck="true"> … </div>
```

Специфические атрибуты уникальны для того или иного тега (возможно – группы тегов). Изучение подобных тегов возможно в контексте того тега, в котором он употребим. Например, тег `img` может быть использован со следующими атрибутами:

`align` — Задает способ выравнивания рисунка по краю, а также способ обтекания текстом.

`alt` — Задает альтернативный текст для изображения, который будет показан, если, например, в браузере отключено отображение картинок.

`border` — Толщина рамки вокруг изображения.

`height` — Высота изображения.

`hspace` — Горизонтальный отступ от изображения до окружающего контента.

`ismap` — Говорит браузеру, что картинка является серверной картой-изображением.

`longdesc` — Указывает адрес документа, где содержится аннотация к картинке.

`lowsrc` — Адрес изображения низкого качества.

`src` — Указывает путь к графическому файлу.

`vspace` — Задает вертикальный отступ от изображения до его окружения.

`width` — Задает ширину изображения.

`usemap` — Ссылка на тег **<map>**, содержащий координаты для клиентской карты-изображения.

Пример использование тега `<img>` в документе с набором специфических атрибутов:

```
<img src="logo.gif" alt="Логотип" height="100px"
usemap="#pictureMap"/>
```

У других тегов имеется свой набор атрибутов, поэтому их общее количество столь велико, что рассмотрение их всех выходит за рамки настоящего пособия, для поиска нужного атрибута всегда можно воспользоваться справочниками (в том числе online [4]). Если значение атрибута не соответствует стандарту – атрибут игнорируется, при валидации будет зафиксирована ошибка.

В html5 допустимо объявлять пользовательские атрибуты, что позволяет связать с элементом дополнительную информацию. Подобные атрибуты должны начинаться с «data-»:

```
<li data-genre="fantasy">Властелин колец</li>
```

Все html-теги можно классифицировать по своему назначению и особенностям определения и поведения. На уровне организации структуры документа можно выделить *структурные* теги, которые задают общую структуру html-документа, определяя границы документа и его основные составляющие: заголовок и тело. К тегам этого типа относятся <html>, <head>, <body>.

С точки зрения верстки документа важным является понимания различия *блочных* и *строчных* элементов документа.

Блочные элементы задают прямоугольную область в документе, заполненную некоторым содержимым (в том числе – вложенными элементами). Блочные элементы по умолчанию занимают всю доступную ширину в документе, но для них можно задавать ширину, высоту, а также внешние и внутренние отступы. Еще одна особенность блочных элементов – по умолчанию они отделяются от окружения переносами строк сверху и снизу. К блочным элементам относятся: `<address>`, `<article>`, `<aside>`, `<body>`, `<blockquote>`, `<dd>`, `<div>`, `<dl>`, `<dt>`, `<fieldset>`, `<figcaption>`, `<figure>`, `<footer>`, `<form>`, `<h1>-<h6>`, `<header>`,

<hgroup>, <hr>, <main>, <li>, <legend>, <nav>, <noscript>, <ol>, <output>, <p>, <pre>, <ruby>, <section>, <ul>.

Строчные элементы представляют фрагмент документа, являющийся частью строки. При выводе строчных элементов не осуществляется перевод строки ни перед, ни после вывода элемента; для строчного элемента нельзя задавать ширину и высоту, они определяются только содержимым элемента. Отступы для подобных элементов можно задавать только горизонтальные. Строчные элементы могут выступать контейнерами только для текста и строчных элементов. К строчным тегам в html5 относятся: <a>, <abbr>, <acronym>, <area>, <b>, <bdo>,
, <cite>, <code>, <dfn>, , <i>, <img>, <kbd>, <label>, <map>, <object>, <samp>, <script>, <small>, <source>, , , <sub>, <sup>, <time>, <track>, <q>, <u>, <var>.

Некоторые элементы, являясь строчными, сохранили ряд свойств блочных элементов. Это так называемые *inline-block* элементы, для которых сохраняется возможность задавать поля, отступы, ширину и высоту. К этому типу элементов относятся: <button>, <input>, <keygen>, <meter>, <progress>, <select>, <textarea>.

При этом существует возможность отображать блочный элемент как строчный и наоборот. Для этого необходимо использовать для элемента стилевое свойство *display* со значением *«inline»* в первом случае и *«block»* – во втором. Установка свойства *display* в *«inline-block»* преобразует элемент в блочно-строчный.

Для табличного представления информации в стандарт html введено большое количество табличных тегов: <table>, <thead>, <tbody>, <tfoot>, <tr>, <td>, <th>, <caption>, <col>, <colgroup>. Они позволяют гибко группировать и форматировать таблицы в html-документе. Пример таблицы, описанной с использованием html-тегов:

```
<table cellspacing="0" style=" border: 5px double #000;">
```

```
  <caption  style="text-align:right">Результаты  первой  кон-
трольной недели</caption>
  <col width="30px">    <!—описание столбцов таблицы -->
  <col width="200px" style="background:#9FC">
  <col width="100px" span="2" style="background:#CCC">
  <thead style="background:#FFC"><!—описание заголовка таблицы-->
      <tr><th>&#8470;</th><th>ФИО</th><th>ОРВП</th><th>Защита
инф.</th></tr>
   </thead>
  <tbody>   <!—начало тела таблицы -->
    <tr><td>1</td><td>Агеев
A.A.</td><td>20</td><td>10</td></tr> <!—1-я строка -->
    <tr><td>2</td><td>Сергеев
C.C.</td><td>15</td><td>13</td></tr> <!—2-я строка -->
    <tr><td>3</td><td>Мишин
M.M.</td><td>11</td><td>18</td></tr><!—3-я строка -->
 </tbody>
 <tfoot>   <!—описание подвала таблицы -->
   <tr><td></td><td align="right">Средний по груп-
пе</td><td>15.3</td>    <td>13.6</td></tr>
 </tfoot>
</table>
```

В результате будет сформирована таблица следующего вида:

Результаты первой контрольной недели

№	ФИО	ОРВП	Защита инф.
1	Агеев А.А.	20	10
2	Сергеев С.С.	15	13
3	Мишин М.М.	11	18
	Средний по группе	15.3	13.6

Зачастую необходимо сверстать таблицу сложной внутренней структуры, в которой надлежит объединить ряд ячеек в строках или столбцах. В подобных случаях используются атрибуты *colspan* и *rowspan* для ячеек таблицы.

Атрибут ячейки *colspan* применяется для многоколончатых таблиц, он позволяет объединить несколько (количество указывается как значение атрибута) ячеек в одну:

```
<td colspan="3">Ячейка в 3 столбца по горизонтали</td>
```

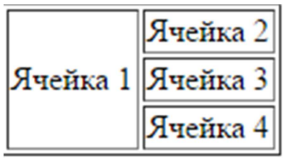
Атрибут ячейки *rowspan* применяется для многострочных таблиц, он позволяет объединить несколько (количество указывается как значение атрибута) ячеек в одну:

```
<td rowspan="2">Ячейка в 2 строки по вертикали</td>
```

Рассмотрим пример таблицы с группировкой ячеек:

```
<table border="1">
 <tr>
  <td rowspan="3">Ячейка 1</td>
  <td>Ячейка 2</td>
 </tr>
 <tr><td>Ячейка 3</td> </tr>
 <tr><td>Ячейка 4</td> </tr>
</table>
```

В результате будет построена таблица:



Еще один часто используемый формат представления информации на странице HTML-документа – формы. Чаще всего они предназначены для организации интерактивного взаимодействия посетителя с Web-приложением, позволяя ему просматривать и изменять данные перед передачей их на серверную сторону для фиксации, изменения или удаления в серверных хранилищах.

В основе функционирования форм лежит тег *form*, являющийся контейнером для ряда элементов представления и редактирования данных, а также средством интеграции процедуры отправки данных на серверную сторону web-приложения:

```
<form атрибуты> ... </form>
```

Атрибуты тега *form* позволяют конфигурировать процессы обработки данных в полях формы и подготовки их передачи на сервер. Наиболее часто используемыми атрибутами формы являются:

`action` — Указывает программу или URI-адрес документа, которому будут переданы данные формы:

```
<form action="/form_handler.php"> ... </form>
```

`enctype` — Задает способ кодирования данных перед отправкой на сервер. Возможные значения:

– `application/x-www-form-urlencoded` – кодировка с

	заменой пробелов и нестандартных символов (например, кириллических) 16-ными кодами; – `text/plain` - кодировка с заменой пробелов 16-ными кодами; – `multipart/form-data` – передача без кодировки. Используется, если вместе с формой передаются файлы.
`method`	Метод протокола HTTP, который будет использован для передачи данных на сервер: `<form action="/form_handler.php" method="post"> ... </form>`
`autocomplete`	Включает или отключает автозаполнение форм: `<form autocomplete="on | off">...</form>`
`novalidation`	Отменяет встроенную проверку валидации формы (корректность заполнения полей).

Пример определения формы в HTML-документе:

```
<form action="http://www.some.com/submit.php" method="post"
name="f">
 <fieldset>
  <legend>Персональная данные</legend>
  <label for="firstname" accesskey="f">Имя: </label>
  <input type="text" id="firstname" name="firstname" tabin-
dex="1" title="first name"><br>
  <label for="lastname" accesskey="l">Фамилия: </label>
  <input type="text" id="lastname" name="lastname" tabin-
dex="2" title="last name"><br>
  <label for="email" accesskey="e">Email: </label>
  <input type="text" id="email" name="email" tabindex="3" ti-
tle="email"><br>
  <label for="hobby" accesskey="h">Род занятий: </label>
  <select name="hobby" id="hobby">
   <option value="0"></option>
   <option value="1">Студент</option>
   <option value="2">Служащий</option>
   <option value="3">Пенсионер</option>
   <option value="4">Безработный</option>
  </select>
 </fieldset>
 <fieldset>
  <legend>Комментарий</legend>
  <label for="comments" accesskey="c">Текст комментария:
</label>
<textarea name="comments" rows="3" cols="23" id="comments" ta-
```

```
bindex="4" title="comments"></textarea><br>
 <label for="kludge"></label>
 <input type="submit" value="Отправить" id="submit" tabin-
dex="5"> <input type="reset" value="Отмена" id="reset" tabin-
dex="6" accesskey="c">
 </fieldset>
</form>
```

Элементы html-форм предоставляют возможность посетителю страницы вводить/определять те данные, которые будут переданы серверу при отправке данных формы. Одним из наиболее распространенных тегов, используемых в рамках html-форм, является *input*. Варьируя атрибут type данного тега, можно размещать на форме следующие типы элементов:

Атрибут type	**Элемент**
`text`	Поле для ввода текстовой информации произвольной структуры
`password`	Ввод пароля
`file`	Выбор файла для отправки
`checkbox`	Список из нескольких альтернатив с множественным выбором
`radio`	Список из нескольких альтернатив с единственным выбором
`submit`	Кнопка отправки данных на сервер. После нажатия на эту кнопку данные из полей формы валидируются (если эта функция не отключена у формы) и отправляются на обработку по адресу, указанному в атрибуте action формы
`reset`	Кнопка сброса значений полей формы к умалчиваемым значениям
`hidden`	Скрытое поле
`range`	Диапазон значений
`number`	Целочисленное значение
`email`	Поле ввода электронного адреса
`url`	Поле ввода URL-адреса
`date`	Поле ввода даты

Также в форму могут быть включены теги:

`textarea`	Многострочное поле для ввода текстовой информации

`select`	Список из нескольких альтернатив, задаваемых тегом `<option>`, допускающий выбор одного или нескольких предлагаемых вариантов
`label`	Метка для элемента формы
`fieldset`	Контейнер для группировки нескольких элементов формы

В состав html-документа включается также ряд директив и тегов, которые не имеют визуального представления, но играют важную роль при обработке документа браузером.

Для указания версии языка HTML, на котором написан документ (версии стандарта, которой он должен соответствовать), следует поместить в первую строку этого документа конструкцию DOCTYPE языка SGML с указанием типа документа и URI реализации версии HTML. Документ в стандарте 4.01 со строгим соответствием правилам описания элементов (не должны использоваться фреймы и устаревшие теги) описывается тегом вида:

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

Документ в стандарте XHTML 1.1 в переходном варианте – содержат устаревшие теги в целях совместимости и упрощения перехода со старых версий стандарта:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Для документов, следующих стандарту HTML5, описание было упрощено:

```
<!DOCTYPE html>
```

Проверить соответствие HTML-кода документа заявленному стандарту можно в процессе его *валидации*. В процессе валидации специализированные инструменты проверяют, насколько соответствует ваш документ заявленному стандарту, и нет ли ошибок в коде (некорректных или устаревших тегов и атрибутов, ошибочных значений атрибутов, незакрытых

тегов, наличие нескольких элементов с одним и тем же стилевым идентификатором и др.). Если документ невалиден, есть опасность, что он будет некорректно отображаться в некоторых браузерах, будут расходоваться лишние ресурсы при его разборе перед отображением, что не лучшим образом скажется на времени обработки страницы. Также необходимо учитывать, что неряшливость в html-коде может негативно повлиять на его SEO-характеристики и понизить рейтинг страницы в рейтингах поисковых систем.

Для проверки валидности можно использовать инструменты:

- онлайн-валидация на сайте консорциума W3C [5] (рис. 2.1);

- плагины популярных браузеров (например, HTML-валидатор в Mozilla Firefox или HTML Tidy Browser Extension для Chrome);

- встроенные инструменты в популярных инструментальных системах для Web-разработчиков (например, функции проверки валидности документа встроены в JetBrains, PHPStorm и Adobe Dreamweaver).

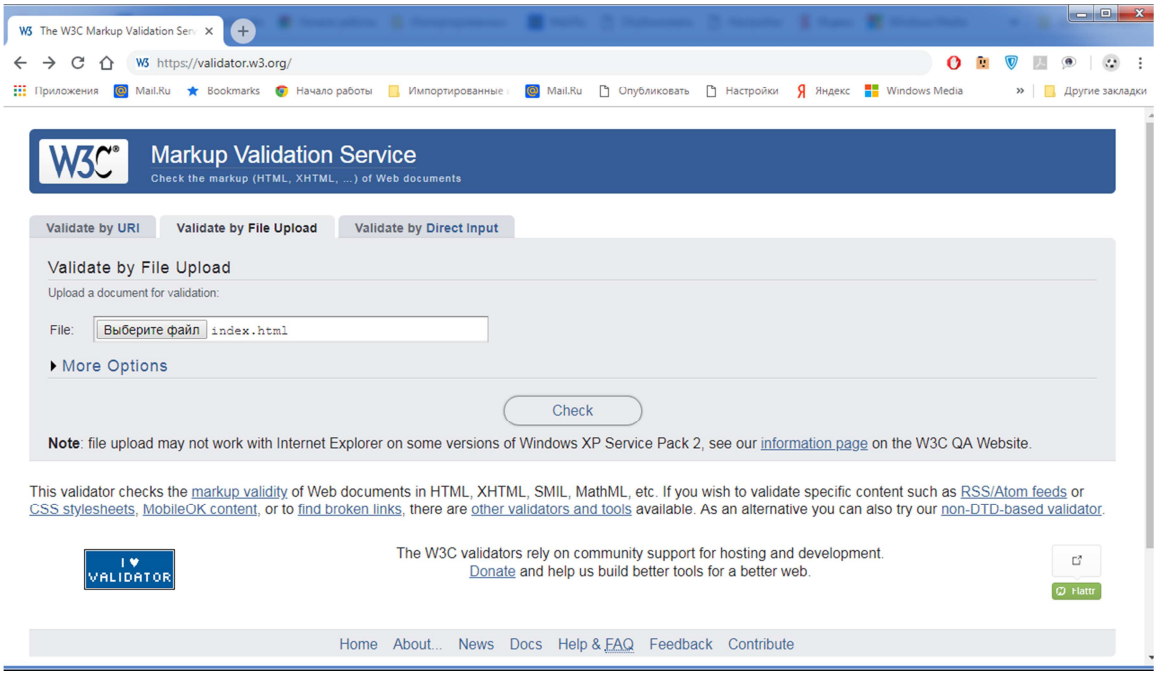


Рисунок 2.1 — Страница online-валидации на сайте W3C

Еще одной важной группой тегов являются теги заголовка документа. Эти теги добавляются в заголовок документа и предназначены для

настройки отображения документа в браузере и взаимодействия с поисковыми машинами. К тегам данной группы относятся:

- тег `<title>`, определяющий заголовок документа в окне браузера;
- тег `<meta>`, определяющий ряд метатегов документа. Какой именно метатег определяет данный тег, зависит от используемых атрибутов. Доступны атрибуты:

`charset`	Указывает кодировку символов для текущего HTML-документа.
`content`	Определяет значения, сопоставляемые с атрибутом *http-equiv* или *name*
`http-equiv`	Задает HTTP-заголовок документа
`name`	Задает имя метатега

Для атрибута *http-equiv* можно задать значения:

`Content-Type`	Указывает MIME-тип документа с опциональным указанием используемой кодировки: `<meta http-equiv="Content-Type" content ="text/html; charset=utf-8">`
`Expires`	Дата устаревания документа. По истечении этой даты документ будет извлекаться с сервера, а не из локального кеша. `<meta http-equiv="expires" content="Tue, 5 Jul 2016 09:30:21 GMT">`
`Refresh`	Задает задержку (в секундах), после которой необходимо в автоматическом режиме обновить документ (опционально - загрузить другой): `<meta http-equiv="Refresh" content="5, URL=http://somesite.com">`
`Cache-Control`	Управляет кешированием содержимого документа. Атрибут content в этом случае может принимать значения: public — документ кэшируется общедоступных кэшах private — только в частном кэше no-cache — запрет кеширования no-store — документ кэшируется, но не может быть сохранен `<meta http-equiv="Cache-Control" content="no-cache">`
`Content-language`	Указывает язык документа: `<meta http-equiv="content-language" content="ru">`
`Pragma`	Отменяет кеширование документа (используется для совместимости с HTTP/1.0): `<meta http-equiv="pragma" content="no-cache">`
`Set-Cookie`	Устанавливает куки браузера. В атрибуте content можно задать для куки имя и значение, до какой даты должен храниться, для какого домена актуален, установить режим безопасной передачи: `<meta http-equiv="Set-Cookie" con-`

	tent="SomeCookie=SomeValue; expiries=Monday, 11-Jul-16 12:12:12 GMT; domain.=www.mysite.com; path=/; secure" />
Content-Script-Type	Определяет используемый на странице язык сценариев: <meta http-equiv="content-script-type" content="text/javascript">

Метатеги, объявленные с атрибутом *name,* предоставляют описание страницы, ключевые слова, информацию об авторе, а также указания для поисковых машин. Для атрибута *name* доступны значения:

Keywords	Задает список ключевых слов, соответствующих содержимому сайта, используется поисковыми системами: <meta name="Keywords" content="программирование, дизайн, верстка">
Description	Задает описание страницы сайта, используется поисковыми системами: <meta name="Description" content="Очень полезная страница">
Robots	Управляет индексацией web-страницы поисковыми роботами. Возможные значения для атрибута content: noindex — запрет индексирования документа; nofollow — запрет перехода по ссылкам в документе; index — разрешает индексирование документа; follow — разрешает проход по ссылкам; all — идентично index, follow; none — идентично noindex, nofollow. <meta name="Robots" content= "noindex, nofollow">
Author	Информация об авторе (создателе) сайта <meta name="Author" content="address@xome.come">
Document-State	Сообщает поисковым роботам, является ли документ статичным (Static), не требующим дальнейшей реиндексации, или динамическим (Dynamic), который в дальнейшем необходимо будет индексировать повторно: <meta name="Document-State" content="Static">
Revisit-After	Указывает поисковым роботам частоту обновления информации для повторного визита: <meta name="Revisit-After" content="20">

Кроме перечисленных в заголовке документа, могут быть размещены теги:

- Тег `<script>` для размещения скриптов клиентской части на языках JavaScript, VBScript.

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

- Тег <style> предназначен для формирования локальной таблицы стилей:

```
<style>
#contentDiv {color: red;border: solid 1 black;}
</style>
```

- Тег <link> для установки связи с внешним файлом, содержащим стилевые таблицы, шрифт, картинку:

```
<link rel="stylesheet" href="site.css" type="text/css">
<link rel="shortcut icon" href="/img/logo.png"
type="image/png">
```

В 2014 году была опубликована спецификация HTML5. Данная версия стандарта языка разметки реализует следующие концепции:

- расширение семантической мощи языка разметки: новые теги (<nav>, <menu>, <article>, <footer>, <header> и др.) описывают основные структурные элементы документа, они носят не абстрактный, произвольно трактуемый дизайнером/верстальщиком смысл, а конкретизируют назначение каждого фрагмента страницы. В макете, сверстанном тегами html5 (см. рис. 2.2), легко идентифицировать назначение каждого блока, просто унифицировать описание элементов на любом уровне вложенности.

- уменьшение зависимости от внешних plugin'ов достигается поддержкой на уровне языка разметки возможностей, которые ранее были доступны только с использованием сторонних расширений (работа с мультимедиа, геопозиционирование, локальные хранилища данных);

- замена скриптов разметкой подразумевает, что многие эффекты и возможности, которые ранее были доступны только благодаря коду клиентской стороны, теперь можно использовать как неотъемлемый функционал тегов-элементов документа. Примерами здесь могут служить управляющие элементы для форм с тегом <input> и атрибутами *number, range,*

date, search, color и т.п., которые позволяют добавить в форму функциональность выбора дат, цветов, чисел в заданных диапазонах.

- обработка браузерами ошибок верстки документов позволяет оставить в прошлом проблемы обеспечения кроссбраузерности верстки HTML-документов, HTML5-совместимые браузеры умеют анализировать код страниц, выявлять и даже исправлять несоответствия стандарту;

- функциональное наполнение инструментария разработчика может быть проиллюстрировано такими возможностями, как работа с локальными хранилищами данных, поддержка многопоточности, двунаправленной передачи данных между клиентской и серверной частью приложения, рисование на канве окна браузера.

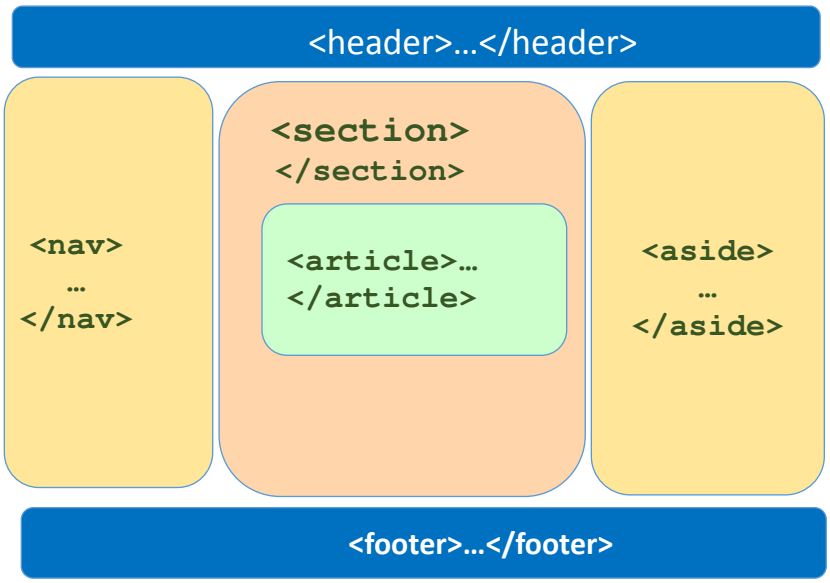


Рисунок 2.2 – Макет верстки страницы тегами html5

Консорциум всемирной паутины (World Wide Web Consortium, W3C) стремится поддерживать стандарт html в состоянии, позволяющем разработчикам web-приложений использовать на страницах современный и ожидаемый посетителями контент. Так, например, запрос на широкое использование мультимедийного содержимого привел к тому, что в стандарт html были включены теги `<video>` и `<audio>`, позволяющие добавлять на страницы видео- и аудио-документы в различных форматах.

Тег <video> позволяет добавлять, проигрывать и конфигурировать видеоролики на странице [6]. Путь к файлу с видео указывается в атрибуте *src*, также можно использовать один или несколько (с различными вариантами кодеков) вложенных элементов <source>. Тег предоставляет широкий выбор атрибутов для всесторонней настройки процесса воспроизведения видео:

autoplay	Включает режим автоматического воспроизведения видео после загрузки страницы.
controls	Добавляет панель управления к видеоролику.
height	Высота области воспроизведения видеоролика.
loop	Включает режим повторно воспроизведения видеоролика после его завершения.
poster	Ссылка на файл с картинкой, которая будет выступать в качестве заставки, когда видео еще не доступно (загружается) или не воспроизводится.
preload	Указывает, что видео необходимо загружать вместе с загрузкой самой страницы.
src	Путь к воспроизводимому видеоролику.
width	Ширина области воспроизведения видеоролика.

Пример использования тега <video> на странице:

```
<video width="500" height="400" controls="controls"
poster ="/img/video/dance.jpg">
   <source src="/video/dance.ogv" type='video/ogg; co-
decs="theora, vorbis"'>
   <source src="/video/dance.mp4" type='video/mp4; co-
decs="avc1.42E01E, mp4a.40.2"'>
   К сожалению ваш браузер не поддерживает тег video
</video>
```

Как видно из примера, допустимо указать несколько файлов с видео, чтобы браузер мог выбрать для отображения тот, который сжат с использованием поддерживаемого кодека. Нельзя забывать и о тех браузерах, которые еще не поддерживают данный тег, для них предусмотрена специальная строка-предупреждение.

Тег `<audio>` похож на тег для видеороликов с поправкой на тип отображаемого контента (для него, например, не имеют смысла атрибуты height и width). Достоинствами мультимедийных тегов HTML5 являются поддерживаемая для них событийная модель, а также достаточно мощный API, который позволяет кастомизировать и стилизовать аудио- и видеопроигрыватели на странице, но эта тема относится к динамическому режиму работы с контентом и будет подробнее обсуждаться в следующих главах пособия.

Перечисленные возможности языка разметки HTML позволяют убедиться в его широких возможностях описания и обработки данных документа. Однако идеология, положенная в основу языка HTML в старших версиях стандарта (и особенно – в HTML5), не позволяет обойтись только средствами языка разметки для полноценного описания документа, поскольку в нем не заложены средства *представления* данных в приложении-клиенте. За внешний вид структурных элементов HTML-страницы сегодня отвечают таблицы CSS.

2.2. Каскадные таблицы стилей (Cascading Style Sheets, CSS)

Изначально язык HTML создавался как средство тотального описания внутреннего содержимого и внешнего вида страниц для Интернет-ресурсов. Однако очень скоро стало понятно, что семантика ресурса и способ его отображения хоть и являются сущностями, связанными в единстве восприятия страницы пользователем, но по сути своей имеют разное назначение и, соответственно, могут и даже должны описываться раздельно. Преимущества отдельного описания способа представления страницы будут приведены далее. Подобный подход привел к появлению еще одного языка, которым должен владеть современный специалист в Web-разработке, – языка описания каскадных таблиц стилей (Cascading Style Sheets, CSS). Первая версия языка (CSS1) вышла в 1996-м году, на момент

написания настоящего пособия актуальна версия CSS3, ведутся работы и активно анонсируются возможности CSS4. Появление и повсеместное использование CSS привело к тому, что за ненадобностью (а главное – из-за несоответствия принципам разделения описания Web-страниц) в стандартах HTML были объявлены устаревшими и сегодня считаются рудиментами теги, с помощью которых на заре сайтостроения оформлялся внешний вид HTML-страницы: `<font>` (установка шрифта), `<u>` (подчеркнутый шрифт), `<big>` (увеличение размера шрифта н единицу), `<strike>` (зачеркнутый шрифт) и др. Данные теги все еще можно использовать, но их применение считается дурным тоном в верстке, поскольку с теми же задачами гораздо эффективнее справляются директивы языка CSS.

Что же представляет собой язык CSS? Это язык описания стилей, с использованием которого можно легко задавать оформление имеющихся на странице HTML-тегов, причем в настоящее время как в статике, так и в динамике, как в двумерном, так и в трехмерном пространстве. Учитывая, что язык CSS, так же как и HTML, не является языком программирования, а также использует декларативный подход к описанию решения задач, надо признать эти языки очень эффективными и во многих проявлениях эффектными. Каждое подобное описание может быть сопоставлено одному из элементов документа, нескольким элементам, удовлетворяющим некоторому общему условию, или даже всем элементам документа.

CSS позволяет задать для элементов документа такие параметры отображения, как размер и начертание шрифта, цвет текста, поля, размеры и положение на странице, фоны и картинки, анимировать изменение параметров отображения, адаптировать внешний вид документа к условиям отображения, то есть решает вопросы формирования представления элементов в окне браузера или на других устройствах отображения (например, версии для печати на принтере).

Основные преимущества CSS [7]:

– отделение представления от разметки позволяет гибче настраивать внешний вид документа – изменение стилевого оформления не влияет на разметку, новые элементы на странице получают унифицированный внешний вид без дополнительных объявлений;

– стилевое оформление позволяет адаптировать внешний вид к носителю, на котором будет отображаться страница (учитывая такие параметры, как ширина и высота среды отображения, количество поддерживаемых цветов, тип среды отображения, ее разрешение);

– стилевое оформление позволяет вносить динамику в отображение документа (например, псевдокласс :*hover* или свойство CSS3 *animation*);

– иерархия описания стилей позволяет наследовать способы представления элементов от контейнера к внутренним элементам, эффективно повторно применяя описание оформления от родительских элементов к дочерним;

– возможность определения альтернативных стилей для документа позволяет варьировать его внешний вид, применяя в конкретный момент тот набор правил, которые формируют понравившийся посетителю вариант внешнего вида страницы.

Таблицы стилей строятся как совокупность правил (позиция 1 на рис. 2.3). Каждое правило описывает правила отображения того или иного документа. К какому(-им) элементу(-ам) относится то или иное правило, определяют один или несколько **селекторов (3)**. **Блок определения (2)** правила выделяется фигурными скобками и содержит набор **свойств (4**), отделенных друг от друга точкой с запятой (;) (после последнего свойства точка с запятой необязательна). Для каждого свойства указывается **значение (5)** отделенное от названия свойства знаком двоеточия (:).

Рисунок 2.3 – Структура CSS-правила

Например, следующее правило установит для всех блоков документа, заключенных в тег <p>, красный цвет начертания текста (в правиле 1), синюю сплошную рамку шириной 1 пиксель (правило 2), светло-желтый цвет фона (правило 3) и размер букв текстового содержимого размером 13 пикселей (правило 4):

```
p
{     color: red;
      border: 1px solid blue;
      background: #fffae0;
      font-size: 13px;
}
```

Для того чтобы формировать CSS-правила, необходимо знать свойства и их возможные значения. На сегодня список CSS весьма обширен, он увеличивался по мере развития технологии. Полный перечень свойств CSS можно посмотреть на сайте W3C [8]. В таблице 2.2 приведен ряд наиболее часто используемых при верстке документов CSS-свойств.

Таблица 2.2. Свойства CSS

Свойство	Описание	Версия CSS
background	Устанавливает значения свойств фона (-ов) элемента в одном объявлении	1
background-color	Устанавливает цвет фона элемента	1
border	Позволяет задать совокупность характеристик рамки элемента: цвет, толщину, стиль начертания.	1

	Эти же характеристики можно задать по отдельности с использованием свойств border-color, border-width, border-style. Имеется также возможность определить свойства каждой из сторон рамок (верхней – top, нижней – bottom, левой – left и правой - right). Например, для левой границы используются свойства: border-left, border-left-color, border-left-width, border-left-style	
color	Изменяет цвет текста внутри элемента	1
display	Указывает, как будет отображаться элемент в браузере (например, может изменить тип элемента со строчного на блочный и обратно)	1
font	Позволяет задать несколько характеристик шрифта для текста внутри элемента	1
height	Задает высоту для элемента	1
letter-spacing	Определяет межсимвольное расстояние в тексте	1
line-height	Определяет межстрочный интервал (интерлиньяж)	1
list-style	Управляет видом и способом отображения маркеров в списках	1
margin	Задает внешние отступы для элемента относительно границ родительского элемента	1
padding	Задает внутренний отступ в элементе от рамки до содержимого элемента	1
text-align	Определяет выравнивание по горизонтали текста в элементе	1
text-decoration	Добавляет эффекты отображения для текста элемента (подчеркивание, надчеркивание, зачеркивание, мигание)	1
width	Задает ширину для элемента	1
word-spacing	Определяет интервал между словами в тексте	1
bottom	Определяет положение нижней границы элемента относительно некоторой точки отсчета (положение точки отсчета задается свойством *position*)	2
clip	Определяет область обрезки для сужения видимой часть абсолютно позиционированных элементов	2
cursor	Определяет вид курсора мыши при наведении на элемент	2
left	Определяет положение левой границы элемента относительно некоторой точки отсчета (положение точки отсчета задается свойством *position*)	2
max-height	Указывает максимальную высоту для элемента	2
max-width	Указывает максимальную ширину для элемента	2
min-height	Указывает минимальную высоту для элемента	2

min-width	Указывает минимальную ширину для элемента	2
overflow	Определяет, как должен отображаться элемент, если он не умещается в заданных границах. Отдельно определены свойства для превышения размеров по горизонтали (<overflow-x>) и вертикали (overflow-y)	2
position	Определяет метод позиционирования элементов относительно окна браузера или соседних элементов	2
right	Определяет положение правой границы элемента относительно некоторой точки отсчета (положение точки отсчета задается свойством *position*)	2
top	Определяет положение верхней границы элемента относительно некоторой точки отсчета (положение точки отсчета задается свойством *position*)	2
visibility	Позволяет скрыть или отобразить элемент на странице	2
z-index	Указывает порядок наложения элементов друг на друга по оси Z	2
animation	Позволяет задать несколько параметров анимации элемента	3
box-shadow	Добавляет эффект отбрасывания одной или нескольких теней к элементу	3
box-sizing	Изменяет тип используемой на странице блочной модели	3
opacity	Устанавливает уровень прозрачности элемента	3
text-overflow	Определяет способ отображения текста, переполняющего область отображения	3
text-shadow	Определяет параметры тени для текста	3
transform	Применяет 2D или 3D преобразование (масштабирование, вращение, наклон, сдвиг) к элементу	3

Необходимо отметить, что многие свойства имеют достаточно сложную структуру, описывая сразу несколько характеристик элемента. В качестве примера здесь можно привести свойство `background`, определяющее фон для элемента. Обобщенно это свойство определяется следующим образом:

```
background: [<фон>, ]* <последний_фон>
```

где

```
<фон> = [background-attachment || background-image || background-position || background-repeat] | inherit
```

```
<последний_фон> = [background-attachment || background-color || background-image || background-position || background-repeat] | inherit
```

Таким образом, свойство позволяет задать (начиная с версии CSS3) совокупность фонов, для каждого из которых можно определить цвет, картинку, положение и смещение фона относительно края элемента, повтор фонового изображения по обеим осям. Последний фон позволяет наряду с фоновыми изображениями задать фон предопределенного цвета. Вот как может выглядеть описание фона элемента:

```
.body {
 background: #ffffff url("back_img.png") no-repeat right top;
}
```

Отдельные элементы фона можно задавать, используя более конкретные CSS-свойства:

```
.body {
  background-image: url("back_img.png");
  background-repeat: repeat-x;
}
```

Из приведенного (причем далеко не полного) описания одного CSS-свойства можно понять, что детальное описание всех свойств выходит далеко за рамки настоящего пособия, за подробностями необходимо обращаться, например, на специализированные сайты [9].

Стили элементам документа можно задавать тремя способами:

1. В виде ссылки на внешний файл. В этом случае правила размещаются в текстовом файле с расширением *css*, который загружается вместе с документом благодаря использованию тега `<link>`:

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

тег `<link>` содержит три атрибута:

- `rel="stylesheet"` – конкретизирует, что ссылка указывает на файл с таблицей стилей;
- `type="text/css"` – указывает MIME-тип содержимого подключаемого файла;

– `href="styles.css"` – определяет путь к подключаемому внешнему CSS-файлу. Это может быть в том числе URL-ссылка на внешний ресурс, что позволяет использовать стили из файлов, размещаемых на других Интернет-ресурсах:

```
<link rel="stylesheet" type="text/css" href=
                    "http://some.com/css/styles.css" />
```

Использование таблиц стилей в виде ссылок на внешний файл предпочтительно в связи с тем, что их содержимое кешируется браузером на случай повторных загрузок страницы.

2. Определение таблиц стилей в теге `<style>` в теле самого документа (обычно в теге `<head>`, но необязательно). В этом случае все правило необходимо поместить между открывающимся и закрывающимся тегом `style`:

```
<style>
   * {margin: 2px; padding: 0; font-size: 20px;}
   .infoBlock {padding: 10px; width: 75%;}
   #menu ul {color: blue; padding: 5px; font-size: bold;}
</style>
```

Для таблиц стилей, определенных в теге `<style>`, а также подключенных тегом `<link>`, можно использовать инструкцию включения внешних файлов с таблицами стилей:

```
@import "dialogs.css"
@import url("http//www.sit.com/outer.css")
```

3. Использование встроенных стилей элемента. Под встроенным стилем подразумевается метод отображения, заданный для элемента индивидуально с помощью атрибута `style`. Данный атрибут позволяет вписать нужные для формирования представления CSS-правила непосредственно в определение HTML-элемента:

```
<div style="border: 1px solid red; background-color: yellow;">Содержимое
</div>
```

Подобного способа определения стилей необходимо по возможности избегать: он приводит к увеличению HTML-кода, противоречит принципу разделения описания семантики и представления документа, не позволяет использовать стили повторно, неудобен для визуального восприятия.

Для третьего способа определения CSS-стиля сразу понятен целевой элемент разметки, оформление которого этим стилем задается. Если используется первый или второй из перечисленных выше способов определения таблиц стилей, то сразу встаёт проблема привязки того или иного правила к тому (тем) элементу (-ам), которому (-ым) он предназначен по замыслу верстальщика. Для этих целей в правиле указывается селектор. Какого вида селекторы могут использоваться в CSS-таблицах? В этом качестве могут использоваться:

Селекторы тегов. Если в качестве селектора правила указать наименование тега, то оно будет применяться ко всем элементам, добавленным в документ с использованием данного тега.

```
p {
  color:green;
}
div {
  border: red solid 1px;
}
```

Селектор-идентификатор. Селектор данного типа начинается со значка ‘#’, и связанное с ним правило будет применено к элементу, имеющему аналогичное значение в атрибуте *стилевого идентификатора* id. Если учесть, что два разных элемента на странице не могут иметь одинаковый атрибут id, можно утверждать, что подобным образом оформляются уникальные стили элементов документа.

```
#menuBar {
    color: green;
```

```
    font-family: verdana;
    font-size: 1.2em;
}
```

В документе элемент, к которому будет применено данное правило, должен быть объявлен следующим образом:

```
<div id="menuBar">…</div>
```

Селектор класса. Данный вид селекторов позволяет сопоставить правилу оформления группу элементов, причем необязательно с одним и тем же тегом. Главное, чтобы у всех этих элементов был общий атрибут *стилевой класс* `class`.

```
<p class="greenBack">…</p>
<div class="greenBack">…</div>
```

В таблице стилей правило для класса задаётся с использованием символа «точка»:

```
.greenBack {
   background-color: green;
}
```

Селекторы псевдоклассов. Псевдоклассы привязывают правило к некоторому состоянию элемента. Если элемент находится в этом состоянии, то правило к нему применяется. Примерами состояний могут быть: нахождение курсора мыши над элементом, получение элементом фокуса ввода, нахождение элемента в определенной позиции иерархии DOM элемента-родителя. В связи с тем, что состояние изменчиво, правило может применяться к элементу не постоянно: пользователь увел курсор с элемента – состояние изменилось, правило больше не актуально. Объявление селектора с псевдоклассом включает название некоторого селектора и имя псевдокласса через символ двоеточие:

```
a { font-size: 14px;  }
a:hover { font-size: 16 px; }
```

В приведенном примере используется псевдокласс :*hover*, который соответствует состоянию с наведенным на элемент курсором мыши: ссылки, когда курсор на них не наведен, будут иметь размер шрифта 14px, а при наведении он будет увеличиваться до 16px.

В стандарте CSS определен достаточно обширный список псевдоклассов, рассмотрим некоторые из них:

`:hover` – на элемент наведён курсор мыши.

`:active` – ссылка, которая становится активной, но переход по ней еще не совершен.

`:visited` – ссылка, которую уже посещали.

`:link` – ссылка, которую еще не посещали.

`:focus` - элемент находится в фокусе.

`:first-child` – элемент является первым дочерним у своего родителя.

`:last-child` – элемент является последним дочерним у своего родителя.

`:nth-child` – элемент является в заданной позиции у своего родителя (n-м по счету, четным, нечетным и др.).

`:enabled` – доступный (незаблокированный) элемент формы.

`:not` – элемент не относится к перечисленным селекторам.

Например, так можно задать особенное оформление для каждого второго (четного) дочернего элемента `<div>` для блока с идентификатором `newsList`:

```
#newsList>div:nth-child(even) { background: gold; }
```

Селекторы псевдоэлементов. Псевдоэлементы описывают объекты, которых нет непосредственно в разметке документов, но которые либо являются частью существующих элементов, либо жестко к ним привязываются. На сегодняшний день в CSS доступны следующие псевдоэлементы:

`:first-letter` – стиль применяется к первой букве текстового блока.

:first-line – стиль применяется к первой строке текстового блока.

:after – добавляет контент после элемента, к которому привязан псевдоэлемент.

:before – добавляет контент перед элементом, к которому привязан псевдоэлемент.

:selection – стиль выделенного пользователем текста.

Некоторые браузеры добавляют свои псвдоэлементы (например, :ms-clear или :ms-value в Internet Explorer). Использование стилей с подобными псевдоэлементами не будет иметь эффекта в других браузерах. Рассмотрим пример использования псевдоэлементов в CSS-правилах:

```
.info:first-line {
  font-style: italic;
}
.info:first-letter {
  color: #F00;
  font-size: 20px;
}
```

В приведенной разметке курсивом будет выделен текст у всех элементов со стилевым классом info, первые буквы у этих же элементов будут выделены более крупным размером шрифта и красным цветом начертания.

Селекторы на основе атрибута позволяют отобрать элементы для применения правила на основе значений атрибутов элемента (например, атрибут href у ссылки или alt у изображения). Для формулировки селектора на основе атрибута используются квадратные скобки. Так, например, выделить все параграфы, у которых установлен атрибут title можно селектором p[title]. Если необходимо выделить элемент с конкретным значением атрибута, используется другой формат селектора. Вот как можно выделить все ссылки, которые открывают страницу по ссылке в том же окне: a[target='_self']. А так можно выделить все элементы section, у которых атрибут class содержит 'news': section[class~='news']

Универсальный селектор «*» означает, что данное правило будет применено ко всем элементам. Например, вот как можно отменить все внешние и внутренние отступы для элементов документа, которые, возможно, объявлены в таблицах стилей, по умолчанию используемых браузером:

```
*{
   margin:0; padding: 0;
 }
```

Для более точного позиционирования правил по отношению к элементам селекторы приходится комбинировать. Действительно, вряд ли можно предложить правило, которое описывало бы, как должны оформляться ссылки или параграфы во всем документе – в разных структурных блоках документа может быть предусмотрено различное представление для этих достаточно популярных тегов. Но правило можно было бы сформулировать по-другому: «ссылки в блоке заголовка должны выглядеть так» или «все параграфы, вложенные в такой-то блок, оформляем следующим образом». CSS позволяет комбинировать селекторы для контекстной привязки правил. Рассмотрим способы комбинирования селекторов в CSS (таблица 2.3).

Таблица 2.3. Принципы группировки селекторов в CSS

Группировка селекторов	Правило использования стиля
`Селектор1 Селектор2`	Применяется к элементам с *Селектором2*, которые помещены внутрь контейнера с *Селектором1*
`Селектор1 + Селектор2`	Применяется к элементам с *Селектором2*, которые являются непосредственным соседом, следующим за элементом с *Селектором1*
`Селектор1 > Селектор2`	Применяется к элементам с *Селектором2*, которые являются прямыми дочерними по отношению к элементу с *Селектором1*

`Селектор1, Селектор2, …, СелекторN`	Применяется ко всем элементам, селекторы которых перечислены
`Селектор1 ~ Селектор2`	Применяется к элементам с *Селектором2*, которые являются соседями, следующими за элементом с *Селектором1*

Рассмотрим примеры комбинирования селекторов:

```
/* Правило для всех параграфов, размещенных внутри элемента с class="news" */
.news p {
   border: 1px solid black;
   background: #9CC;
}
/* Каждая разделительная линия, расположенная сразу вслед за элементом с class="news", будет иметь красную рамку: */
.news + hr {  border-color: red;  }
/* Остальные разделительные линии – черную */
   hr {  border-color: black;  }
```

Наследование является механизмом, с помощью которого определенные свойства передаются от предка к его потомкам. Спецификацией CSS предусмотрено наследование свойств, относящихся к текстовому содержимому страницы, таких как color, font, letter-spacing, line-height, list-style, text-align, text-indent, text-transform, visibility, white-space и word-spacing. Во многих случаях это удобно, так как не нужно задавать размер шрифта и семейство шрифтов для каждого элемента веб-страницы. Свойства, относящиеся к форматированию блоков, не наследуются. Это background, border, display, float и clear, height и width, margin, min-max-height и -width, outline, overflow, padding, position, text-decoration, vertical-align и z-index.

В названии технологии присутствует слово «*каскадные*». В чем заключается идея каскадности стилей? Каскадность означает, что значение для свойств элемента может быть сформировано не одним правилом, а це-

лой совокупностью правил. Подобная ситуация может возникнуть в случаях:

- использования различных таблиц стилей, которые для одних и тех же селекторов задают свои правила отображения;

- наложения для одного элемента нескольких правил по разным селекторам (по тегу, стилевому идентификатору, стилевому классу);

- наследования значений свойств дочерних элементов от родительских в дереве иерархии DOM.

Если к конкретному элементу веб-страницы применено несколько стилей, то свойства объединятся при условии, что они не конфликтуют между собой.

Рассмотрим небольшой пример: пусть внешняя таблица стилей задает три свойства для элемента <p>:

```
p {
  color: green;
  text-align: center;
  font-size: 10px;
}
```

а внутренняя таблица задает два свойства для этого же элемента:

```
p {
  text-align: left;
  font-size: 12px;
}
```

В итоге наш элемент получит текст зеленого цвета, выровненный по левому краю и в 12px высотой при условии, что внешняя таблица стилей была подключена к файлу перед внутренней.

Другой пример. Допустим, в таблице стилей есть следующий фрагмент:

```
.className { background: red;}
#idBlock { background: green;}
div { background: yellow;}
```

Какой фон будет у следующего блока в документе?

```
<div id="idBlock" class="className">
   Содержимое блока
</div>
```

У блока будет зеленый фон. Ему походят все три правила, поэтому они конкурируют за право установки цвета фона для элемента. Для определения, какое из правил будет доминировать, помогает более полное погружение в стандарт. Согласно ему, к элементу будет применяться стиль, обладающий большей *специфичностью*. Специфичность – это кортеж из четырех цифр, причем отношение больше-меньше для различных кортежей опирается на числовые значения элементов кортежа с приоритетом элементов слева направо, т.е. для следующих 6-ти кортежей выполняются следующие неравенства:

(1, 0, 2, 1) > (0, 2, 1, 1) > (0, 0, 4, 2) > (0, 0, 2, 1) > (0, 0, 0, 3) > (0, 0, 0, 1)

Что означают числа кортежа и как они связаны с CSS-правилами? Кортеж, характеризующий специфичность того или иного правила, составляется по следующим правилам:

– каждый присутствующий в селекторе идентификатор добавляет к специфичности (0, +1, 0, 0);

– каждый класс, псевдокласс или атрибут добавляет к специфичности (0, 0, +1, 0);

– каждый элемент или псевдоэлемент добавляет к специфичности (0, 0, 0, +1);

– встроенный стиль элемента (атрибут `style` элемента) добавляет к специфичности 1 в старший разряд (1, 0, 0, 0)

– универсальный селектор и комбинаторы не учитывается.

Рассчитаем специфичность для нескольких css-правил:

```
div {…}                    /* (0,0,0,1) */
div a {…}                  /* (0,0,0,2) */
#idEl>p.info {…}           /* (0,1,1,1) */
P .className a:hover {…}   /* (0,0,1,3) */
```

Еще раз посмотрим на приведенный выше пример с несколькими подходящими элементу правилами, но уже с указанием их специфичности:

```
.className { background: red;}   /* (0,0,1,0) */
```

```
#idBlock { background: green;}    /* (0,1,0,0) */
div { background: yellow;}        /* (0,0,0,1) */
```

Специфичность второго правила по указанным выше принципам сравнения выше, поэтому правило для стилевого идентификатора в приведенном примере будет доминировать.

Имеется возможность повысить приоритет свойства в том или ином правиле, если добавить ему атрибут `!important`. Если изменить одно из правил предыдущего примера на

```
.className { background: red !important;}
```

то у блока цвет фона станет красным. Если у нескольких правил будет указан атрибут `!important`, то правило будет выбирать по величине специфичности уже среди них.

Значения свойств в CSS-правилах имеют различные типы и могут задаваться строками (`color: black`), числовыми обозначениями цветов (`background: #9CC`), функциями, вычисляющими конкретное значение или выполняющими заданное предобразование (`rgb(255, 100, 0)` или `rotate(-90deg)`), числовыми значениями, выраженными различными единицами измерения. Последний тип свойств используется для указания размеров и позиций расположения элементов разметки (например, `width, margin, top`) и используется очень часто. При этом доступны несколько единиц измерения для числовых значений свойств. Их перечень приведен в таблице 2.4.

Таблица 2.4. Единицы измерения для числовых свойств CSS

Единица измерения	Наименование	Пример использования	Рекомедуемая область вывода
Абсолютные			
in	Дюймы	height: 1.5in	Печать
mm, cm	Миллиметры, сантиметры	margin: 10mm	Печать

pt	пункт (1pt = 1/72in)	padding: 2pt	Печать
pc	пики (1pc = 12pt)	max-width: 50pc	Печать
Относительные			
em	Размер вычисляется относительно высоты текущего шрифта	line-height: 2em 2 размера текущего шрифта	Вывод на экран
px	Пиксел	font-size: 10px	Вывод на экран
%	Процент	width: 75%	Вывод на экран
ex	Размер задается относительно высоты строчной буквы x для текущего шрифта	margin: 2ex	Вывод на экран
vw	1% от ширины окна	padding-top: 1vw	Вывод на экран
vh	1% от высоты окна	width: 1.2vh	Вывод на экран
vmin, vmax	Наименьшее (наибольше) их vw и vh	word-spacing:0.2vmin	Вывод на экран
rem	Размер относительно величины шрифта, заданного для элемента html	letter-spacing:0.3 rem	Вывод на экран

Как следует из таблицы, размеры в CSS задаются либо в абсолютных, либо в относительных величинах. Разница заключается в том, что для абсолютных величин размер задается в четких единицах, значения которых известны заранее, а относительные вычисляются в зависимости от контекста формирования страницы (размер окна, заданный в данный момент, браузерный или текущий размер шрифта). В этой связи абсолютные размеры являются менее гибкими, их используют гораздо реже и чаще всего при формировании представления страницы для печати. Относительные же величины позволяют адаптировать размеры элементов условиям отображения, уменьшая их, например, при уменьшении размера окна.

Все элементы в CSS являются прямоугольными блоками. Каждый такой блок имеет зону content, в которой располагается содержимое элемента (т.е. текст, изображения и т.д.). Вокруг зоны content могут располагаться необязательные зоны: padding, border и margin (рис. 2.4).

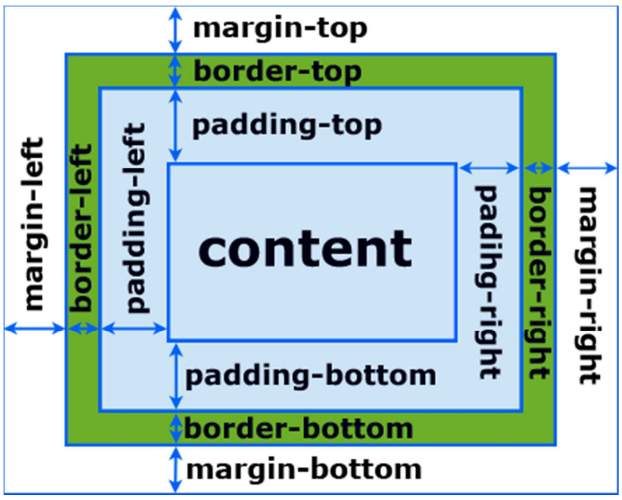


Рисунок 2.4 – Блоковая модель CSS

Внутреннее поле `padding` непосредственно окружает зону `content`. Данное поле позволяет задать отступ содержимого элемента от его границы. Допускается задавать величину внутреннего поля как целиком (свойство `padding`), так и независимо для каждой стороны элемента (свойства `padding-top, padding-right, padding-bottom, padding-left`).

Граница `border` окружает внутреннее поле. Для границы (рамки) элемента можно задать ширину, стиль отображения, цвет и даже (в версии CSS3) картинку для отображения. Границу можно определить в целом (свойство `border`) или для каждой стороны независимо (свойства `border-top, border-right, border-bottom, border-left`).

Внешнее поле `margin` окружает границу и задаёт величину внешнего отступа данного элемента от окружающих. Отступ можно определить свойством *margin* или задать независимо для каждой стороны (`margin-top, margin-right, margin-bottom, margin-left`).

Необходимо помнить, что по умолчанию свойства `width` и `height` устанавливают ширину и высоту только блока `content`, а не элемента целиком. Итоговый размер элемента в документе будет рассчитан с учетом величин внутреннего поля, отступа и границ. Если же для элемента поменять свойство `box-sizing` на `border-box` (вместо `content-box` по умол-

чанию), то ширина и высота элемента будут включать размеры самого содержимого, внутреннего поля и рамки (но по-прежнему не будут включать размер внешних отступов).

Блочная вёрстка страницы предполагает использование блочных элементов (`<div>` до эпохи HTML5, семантические блоки `<head>`, `<nav>`, `<section>`, `<footer>` в HTML5). Главная проблема, с которой сталкиваются начинающие web-разработчики при переходе к подобному типу вёрстки, – преодоление стандартного поведения блочных элементов в потоке HTML-страницы. Как уже было отмечено, по умолчанию блочные элементы отображаются с новой строки, поэтому расположить их в колонки по горизонтали можно, отменив умалчиваемое поведение. Одним из вариантов здесь будет использование *всплывающих* блоков. Всплывающий элемент объявляется со свойством `float`. Значение свойства (`left` или `right`) определяет, к какой границе будет прижиматься элемент, другие элементы будут обтекать его с остальных сторон. С использованием свойства float можно осуществить многоколончатую верстку:

```
<style>
   .column {
     width: 200px;
     height: 300px;
     float: left;
     margin: 0 5px;
   }
  </style>
…
<div class="column">1-я колонка</div>
<div class="column">2-я колонка</div>
<div class="column">3-я колонка</div>
<div style="clear: left">Уже не колонка</div>
```

Блоки, для которых установлен стилевой класс `column`, будут обтекать друг друга с правой стороны, т.е. выстроятся в три колонки на странице. Для четвертого блока указано свойство `clear: left`, чтобы отменить эффект обтекания и извлечь этот элемент из ранее организованной колоннады.

А вот так можно блочно сверстать обобщенный макет страницы, представленный на рис. 1:

```
<!DOCTYPE html>
<html >
 <head>
   …
     <style>
        #page {
          margin: 0 auto;
          width: 900px;
        }
        header { height: 100px;}
        footer {
          height: 50px;
          clear: both;
        }
        nav {
          float: left;
          width: 200px;
        }
        #content {
          margin-left: 200px;
          margin-right: 200px;
        }
        #aside {
              float: right;
              width: 200px;
        }
     </style>
 </head>
 <body>
     <section id="page">
             <header>Заголовок</header>
             <nav>Меню</nav>
             <section id="aside">
               <aside style="height: 200px;">
                   Сайдбар 1
               </aside>
               <aside style="height: 300px; background: #6C9">
                    Сайдбар 2
               </aside>
             </section>
             <section id="content">
                Контент
             </section>
            <footer>Подвал</footer>
     </section>
 </body>
</html>
```

Предложенный вариант блочной верстки не является единственно возможным, современные разработчики используют и гораздо более гибкие инструменты вёрстки, которые обеспечивают ее адаптивность, кроссбраузерность, визуальную гармоничность. Одним из основных инструментов адаптации верстки документа устройству его отображения являются *медиазапросы*. Начиная с версии CSS3 при описании стилей можно учитывать тип и характеристики устройств отображения, на которых будет демонстрироваться документ. Это позволяет определить для документа или некоторых его элементов различные стили оформления, например, для случаев, когда он будет отображаться на большом экране монитора, на смартфоне, на версии для печати на принтере. Адаптация к устройству может заключаться в использовании на альтернативных устройствах отображения различных размеров шрифта, скрытии некоторых элементов, изменении сетки отображения контента и др.

В медиазапросах можно ставить применение того или иного правила оформления для элемента (-ов) в зависимость от выполнения некоторых предопределенных условий, формулируемых как выражение от типа устройства отображения или его параметров. В стандарте CSS3 актуальны следующие типы устройств:

Таблица 2.5. Типы устройств для медиазапросов

Тип	Описание
`all`	Все типы. Это значение используется по умолчанию
`print`	Принтеры и другие печатающие устройства
`screen`	Экран монитора
`speech`	Речевые синтезаторы, а также программы для воспроизведения текста вслух. Сюда, например, можно отнести речевые браузеры

Кроме типа устройств, в медиазапросе могут фигурировать характеристики устройства, которые перечислены в таблице 2.6:

Таблица 2.6. Характеристики устройств для медиазапросов

Параметр	Описание
`width`	Ширина области просмотра. Обычно в правилах проверки указываются минимальные и максимальные значения ширины. Характеристика `min-width` применяет CSS-правило, если ширина области просмотра больше значения, указанного в медиазапросе, а `max-width` — если меньше
`height`	Высота области просмотра. Обычно в правилах проверки указываются минимальные и максимальные значения высоты. Характеристика `min-height` применяет CSS-правило, если ширина области просмотра больше значения, указанного в медиазапросе, а `max-height` — если меньше
`aspect-ratio`	Соотношение ширины к высоте области просмотра. `min-aspect-ratio` проверяет минимальное соотношение ширины к высоте области просмотра, а `max-aspect-ratio` — максимальное
`orientation`	Ориентация области просмотра. Может принимать два значения: `portrait` и `landscape`
`resolution`	Разрешение экрана, которое можно проверять как количество точек на дюйм (dpi) или количество точек на сантиметр (dpcm). Характеристика `min-resolution` проверяет минимальное разрешение экрана, `max-resolution` — максимальное
`color`	Количество бит на каждый из цветовых компонентов текущего устройства вывода. `min-color` задает минимальное количество бит, а `max-color` — максимальное
`monochrome`	Проверяет количество битов на пиксель монохромного устройства. `min-monochrome` задает минимальное количество битов, а `max-monochrome` — максимальное

Медиазапросы могут быть добавлены к документу следующими способами:

1) С помощью метатегов HTML:
```
<link rel="stylesheet" media="screen and (max-width: 1000px)" href="style.css">
```
2) С помощью правила `@import` внутри элемента `<style>` или внешней таблицы стилей:
```
@import url(color.css) screen and (min-width: 1000px);
```
3) Непосредственно в коде страницы:

```
<style>
@media (max-width: 568px) {
  #headVideo {display: none;}
}
</style>
```

4) Внутри таблицы стилей в файле `*.css`:

```
@media (max-width: 568px) {
  #headVideo {display: none;}
}
```

Как можно заметить в приведенных примерах использования медиазапросов, их можно комбинировать для формулировки сложных условий соответствия сразу нескольким критериям. Здесь применимы традиционные подходы формирования логических выражений с использованием операций конъюнкции (`and`), дизъюнкции (в медиазапросах оформляется запятой между условиями), отрицания (`not`). Например, следующий запрос определяет стиль оформления блока для размеров экрана от 400px до 800px (оператор `only` используется здесь для того, чтобы скрыть правило от старых браузеров, не поддерживающих стандарт CSS3):

```
@media (only screen (min-width: 400px) and (max-width: 800px)){
  div { font-size: 12px;}
}
```

А вот так можно скрыть некоторый блок для портретной ориентации на малых размерах экранов устройства отображения:

```
@media (only screen and (max-width: 500px) and (orientation: portrait)){
  #someBlock { display: none;}
}
```

Медиазапросы являются мощным и эффективным, но не единственным инструментом адаптации оформления HTML-документа свойствам устройства отображения. Для более логичного и формализованного определения внешнего вида документа можно использовать дополнительные технологии. Примерами здесь могут служить CSS-модули FlexBox [10] и

GridLayout [11] или фреймворки Twitter Bootstrap [12] или Skeleton. Рассмотрим, как реализуется адаптивная верстка HTML документов в Twitter Bootstrap. Этот инструмент очень популярен у Web-разработчиков, он позволяет, в том числе, создавать адаптивные макеты сайта, подстраивающие расположение контента под размеры окна. В основу адаптивного поведения HTML-страниц здесь положены предопределенные стилевые классы, которые можно комбинировать в объявлениях элементов верстки:

`container` – контейнер с фиксированной шириной, значение которой зависит от ширины экрана;

`container-fluid` – контейнер с плавающей по размеру окна шириной с полями 15px;

`row` – ряд контента, занимающий всю ширину контейнера;

`col-…` – колонки в рамках ряда, в которые можно размещать контент. При этом конкретное название класса из этого семейства класса определяют желаемую ширину элемента для того или иного размера области отображения. Всего Bootstrap поддерживает 5 различных размеров устройств отображения, к которым адаптируется верстка (они определяются внутри библиотеки с использованием механизма медиазапросов):

Размер экрана	Extra small <576px	Small ≥576px	Medium ≥768px	Large ≥992px	Extra large ≥1200px
Название breakpoint	xs	sm	md	lg	xl
для экрана					
Название стилей колонок	col-	col-sm	col-md	col-lg	col-xl

Bootstrap использует 12-колончатую сетку размещения контента: каждая строка (`class "row"`) разметки условно делится на 12 ячеек, при этом элемент строки может занимать произвольное количество (от 1 до 12) подобных ячеек. Рассмотрим пример:

```
<div class="container">
 <div class="row">
  <div class="col-lg-4 col-md-6 col-sm-12" id=”block1”>
     Блок №1</div>
  <div class="col-lg-4 col-md-6 col-sm-6" id=”block2”>
     Блок №2</div>
  <div class="col-lg-4 col-md-12 col-sm-6" id=”block3”>
    Блок №3</div>
 </div>
</div>
```

В приведенном примере блоки с идентификаторами `block1, block2` и `block3` будут занимать треть строки (4 из 12 ячеек) для размеров области отображения от `large` и более, на размере экрана от md до lg блоки `block1` и `block2` разделят одну строку пополам (6 из 12 ячеек), а `block3` займет всю отдельную строку (12 из 12 ячеек), а на малых размерах отображения sm уже блоки `block2` и `block3` будут делить одну строку, а блок с идентификатором `block1` займет целую строку:

Размер lg

Блок №1 Блок№2 Блок №3

Размер md

Блок №1 Блок№2

Блок №3

Размер sm

Блок №1

Блок №2 Блок№3

Таким образом можно очень эффективно адаптировать размеры блоков для отображения на экранах различных размеров. При этом необходимо отметить возможность реализовать вложенность блоков, т.е. если для блока, имеющего размер половину строки отображения (класс `col-*-`

6), объявить вложенный блок-строку, то внутри этой строки на 12 ячеек будет разбиваться ограниченная родительским блоком область, составляющая половину внешнего контейнера-строки. У сетки bootstrap есть еще достаточно много других интересных возможностей, таких как сокрытие блоков для некоторых размеров экрана, формирование отступов элементов в строке в терминах ячеек col, изменение порядка следования элементов в рамках строки, вертикальное и горизонтальное выравнивание элементов в контейнере и многое другое. Подробнее познакомиться с возможностями bootstrap можно на сайте проекта [12].

Классический CSS, как уже можно было убедиться, обладает достаточно мощными и универсальными средствами для описания отображения элементов на HTML-странице. Но зачастую ему не хватает инструментальных средств для обобщенного описания связанных или повторяющихся представлений или поведений элементов на странице. Здесь на помощь могут прийти языковые надстройки над CSS, примерами которых являются LESS, Sass, Stylus. Сохраняя базовые особенности определения стилей, принятые в CSS (и, в конечном счете, компилируясь в нативный CSS-код), эти языки предоставляют целый ряд дополнительных языковых конструкций, которые позволяют (на примере языка Sass):

– использовать вложенность селекторов, приближающую описание представления к описанию семантики в HTML.

Описание Sass

```
.menu
 font-size: 14px
 ul
   background-color: blue;
   li
      color: yellow
```

Описание CSS (компиляция Sass)

```
.menu {
     font-size: 14px;
}
.menu ul {
background-color: blue;
}
.menu ul li {
     color: yellow;
}
```

– использовать переменные и математические операции сложения +, вычитания -, умножения * и деления / для форсирования идентичных или математически связанных значений в разных правилах:

Описание Sass

```
$margin: 20px
$font: Geneva, Helvetica
.news-block
     margin: $margin
     padding: $margin/2
     font-family: $font
```

Описание CSS (компиляция Sass)

```
.news-block {
  margin: 20px;
  padding: 10px;
  font-family: Geneva, Helvetica;
}
```

– за счет примесей (миксинов) определять фрагменты описания, которые могут многократно использоваться в различных правилах (в том числе варьироваться через механизм параметров):

Описание Sass

```
@mixin title-text
  font:
    family: Arial
    size: 20px
    weight: bold
  color: lightgray

.title
  @include title-text
  margin: 0 auto
```

Описание CSS (компиляция Sass)

```
.title {
    font-family: Arial;
    font-size: 20px;
    font-weight: bold;
    color: lightgray;
    margin: 0 auto
}
```

– разбить Sass-описание на ряд мелких файлов и подключать один к другому с использованием директивы @import;

– объявлять наследование стилей, формируя иерархию описания от обобщений, свойственных множеству элементов, до детального описания конечного элемента (группы элементов):

Описание Sass

```
%rounded-border
  -webkit-border-radius: 2px
  -moz-border-radius: 2px
  -ms-border-radius: 2px
  border-radius: 2px
%shadowed
```

```
    -webkit-box-shadow: 10px 10px 5px 0px rgba(0,0,0,0.75)
    -moz-box-shadow: 10px 10px 5px 0px rgba(0,0,0,0.75)
    box-shadow: 10px 10px 5px 0px rgba(0,0,0,0.75)

.info-block
    @extend %rounded-border
    @extend %shadowed
    color: black
    font-size: 14px
```

Описание CSS (компиляция Sass)

```
.info-block {
     -webkit-border-radius: 2px;
     -moz-border-radius: 2px;
     -ms-border-radius: 2px;
     border-radius: 2px;
}
.info-block {
   -webkit-box-shadow: 10px 10px 5px 0px rgba(0,0,0,0.75);
   -moz-box-shadow: 10px 10px 5px 0px rgba(0,0,0,0.75);
   box-shadow: 10px 10px 5px 0px rgba(0,0,0,0.75);
}
.info-block {
     color: black;
     font-size: 14px;
}
```

Как уже было отмечено выше, код Sass (как и других языковых надстроек над CSS) не интерпретируется браузером напрямую, а сначала должен быть преобразован в нативный CSS. Для этого можно использовать специализированный инструментарий (приложения, модули сред разработки, online-конвертеры). Примером здесь может послужить приложение командной строки `sass`, которое можно установить из репозитария GitHub [13].

2.3. Динамическая обработка данных в HTML-документах

Одним из основных недостатков языка HTML является пассивность и статичность документов на этом языке, то есть невозможность алгоритмического построения содержимого документа в соответствии с программой, реализующей алгоритм или в зависимости от действий пользователя, про-

сматривающего документ. Определенную динамику в HTML-документ позволяет внести CSS-оформление, особенно возможности 2D- и 3D-анимации, которые добавлены в версии CSS3. Но эта динамика относится к области представления данных, если же говорить о динамике семантической обработки данных, организации взаимодействия с пользователем и серверной стороной, то здесь HTML-документу не обойтись без скриптов клиентской стороны. Эти скрипты представляют собой программы, загружаемые с сервера вместе с документом HTML и выполняемые браузером при просмотре этого документа. Существует несколько языков, на которых может быть написан сценарий. Наибольшей популярностью пользуется язык JavaScript, разработанный в 90-х годах компанией Netscape, первой реализовавшей поддержку сценариев в своём браузере.

Поскольку программа-сценарий выполняется пользовательским агентом на машине клиента, то данную технологию следует отнести к технологиям стороны клиента. Когда говорят о динамически работе с документом в контексте клиентского приложения, то подразумевают технологию, получившую название DHTML – Dynamic HTML. DHTML не является отдельным языком разметки или программирования, за этим термином скрывается целая совокупность технологий, включающая язык разметки HTML, язык описания представления документа CSS, объектную модель документа DOM, язык сценариев клиентской стороны JavaScript. С двумя первыми языками мы уже познакомились в предыдущих главах, поэтому необходимо подробнее разобраться с DOM и возможностями языка JavaScript.

DOM (Document Object Model, Объектная модель документа) – иерархическое представление структуры документа как совокупности узлов – элементов. Подобная иерархия очень удобна для манипуляции информационными и визуальными элементами, представленными на странице за счет того, что, используя объектно-ориентированный подход к опи-

санию каждого конкретного элемента, она позволяет инкапсулировать описание и поведение элементов и при этом за счет типизации учитывать их индивидуальность.

Первая спецификация DOM (DOM Level 1) была выпущена W3C в 1998 году. В нее вошли два модуля: DOM Core (ядро DOM), определявший способ описания структуры XML-документа, а также методы манипулирования им и его элементами, а также DOM HTML, в котором описывались объекты и методы работы с HTML-документами. В частности, в DOM Level 1 появились такие объекты, как `HTMLDocument`, `HTMLElement`, `HTMLTableElement`, `HTMLBodyElement`, которые и сегодня используются в скриптах для доступа к элементам документа.

Спецификация DOM2 (2000 г.) расширила базовые интерфейсы DOM1 и добавила к ним поддержку событий и стилей. Она включала уже 6 спецификаций: DOM2 Core (API XML), DOM2 Views (API представления документа до и после использования стилей CSS), DOM2 Events (API событий и их обработки), DOM2 Style (API интерфейсов для оформления документов с использованием стилей), DOM2-Traversal-Range (API фильтров и итераторов), DOM2-Range (API выделения текста и узлов).

Наконец, актуальная на сегодня спецификация DOM Level 3 (2004 г.) дополнила DOM возможностями загрузки и сохранения документов в модуле DOM Load and Save, верификации документа в модуле DOM Validation, поддержки спецификации XML 1.0 в модулях XPath, XML Infoset и XML Base. Спецификация DOM призвана обеспечить единообразную обработку элементов во всех браузерах. На деле каждый браузер по-своему поддерживает тот или иной уровень DOM и его спецификации, вводит свои объекты коллекции, в связи с этим код, обрабатывающий DOM-модель документа в одном браузере, может не работать в другом. В связи с этим, если код должен выполняться на широком диапазоне версий браузе-

ров, в коде необходимо убедиться в поддержке той или иной возможности DOM:

```
if ( /* Имя объекта или метода */ ) //если поддерживается
 { /* Использование объекта */ }
else
 { /* альтернативный вариант */ };
```

Проверить, поддерживает ли браузер ту или иную спецификацию DOM с учетом ее версии, можно следующим образом:

```
if ( document.implementation.hasFeatures("Core", "3.0") )
 {//поддерживает DOM2 Core
 }
```

Согласно DOM-модели, всякий HTML- и XML-документ является иерархией. Каждый HTML-тег образует отдельный элемент-узел, каждый фрагмент текста – текстовый элемент и т.п. С учетом возможной вложенности тегов документа, иерархическое представление документа в виде элементов DOM представляется естественным и логичным. Например, для такого несложного документа:

```
<!DOCTYPE html>
<html>
  <head>
     <title>
          Пример DOM-дерева
     </title>
  </head>
  <body>
     <div>
          Всем привет!
     </div>
  </body>
</html>
```

дерево DOM-элементов будет выглядеть следующим образом (рис. 2.5).

Корнем дерева DOM является узел, соответствующий документу. Далее иерархия узлов повторяет вложенность элементов-тегов документа, начиная с тега `<html>`, в который вложены узлы для тегов `<head>` и `<body>`, которые являются родительскими для узлов, сопоставляемых

вложенным тегам, которые в свою очередь могут быть контейнерами и так далее, заканчивая текстовыми узлами, соответствующими содержимому конечных информационных элементов документа. При этом типы узлов дерева соответствуют конкретным сущностям в HTML-документе: есть тип узла, соответствующий всему документу, есть тип, сопоставляемый с XML- или HTML-элементами, комментариями, атрибутами элементов, тексту внутри элемента также сопоставляются узлы специальных типов.

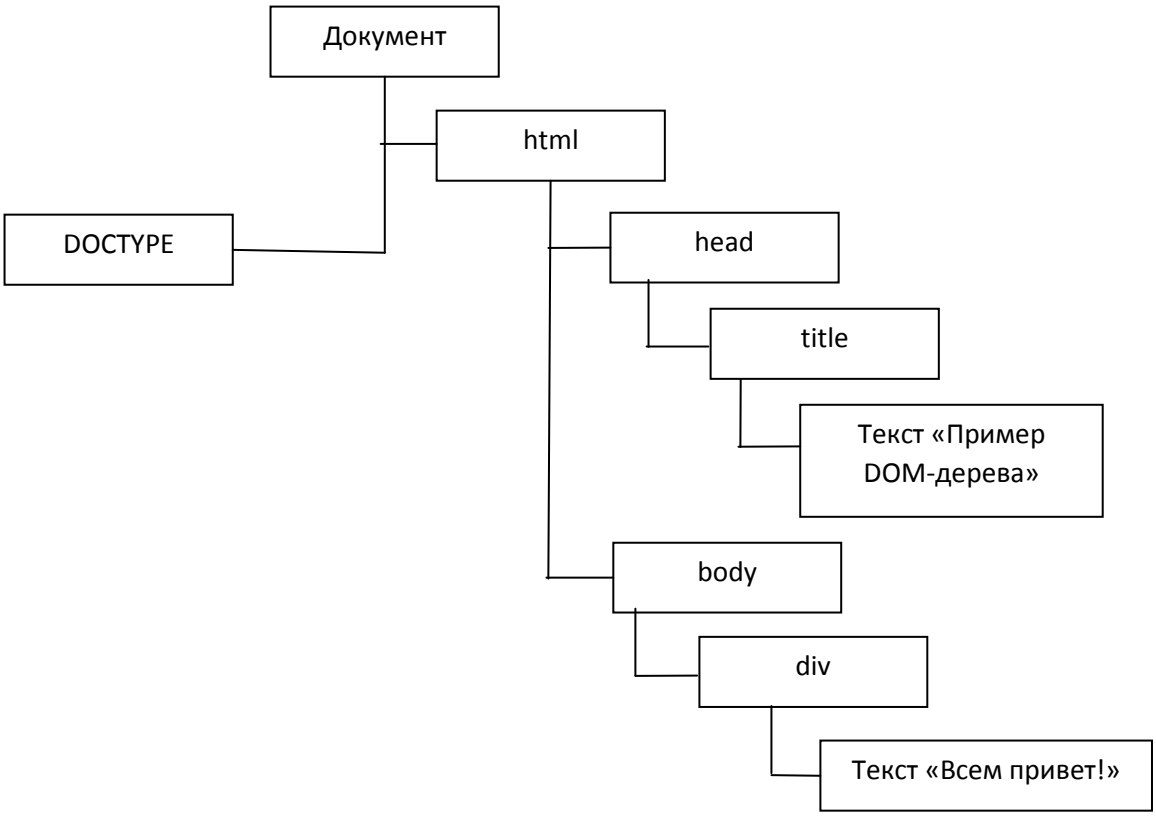


Рисунок 2.5 – Иерархия узлов DOM HTML-документа

DOM определяет стандартный набор интерфейсов узлов, которые может использовать XML- и HTML-анализатор для анализа документа. Типы узлов и соответствующие им идентификационные константы представлены в таблице 2.7.

Таблица 2.7. Типы узлов в DOM2

Имя константы	Значение	Описание
`Node.ELEMENT_NODE`	1	Узел соответствует элементу (тегу

		HTML)
`Node.ATTRIBUTE_NODE`	2*	Узел атрибута (возвращает атрибут элемента XML- или HTML-документа), например `href` у тега `a`
`Node.TEXT_NODE`	3	Текстовый узел, например, для тега `<span>Блок текста<span>` будет создан элемент данного типа для текста «`Блок текста`»
`Node.CDATA_SECTION_NODE`	4	Представляет раздел *CDATA*, который может использоваться в XML для включения расширенных частей неэкранированного текста
`Node.ENTITY_REFERENCE_NODE`	5*	Представляет ссылку на сущность
`Node.ENTITY_NODE`	6*	Представляет объект
`Node.PROCESSING_INSTRUCTION_NODE`	7	Инструкция `ProcessingInstruction` *XML* встраивает в XML инструкции, относящиеся к конкретному приложению, которые могут игнорироваться другими приложениями, не распознающими их. Например, `<?xml-stylesheet ... ?>`.
`Node.COMMENT_NODE`	8	Узел комментария, например `<!– HTML-комментарий –->`
`Node.DOCUMENT_NODE`	9	Узел документа
`Node.DOCUMENT_TYPE_NODE`	10	Определяет тип документа, например: `<!DOCTYPE HTML PUBLIC>`
`Node.DOCUMENT_FRAGMENT_NODE`	11	Фрагмент документа, не имеющий родителя. Позволяет сконструировать фрагмент DOM-дерева и вставить его в основное дерева документа
`Node.NOTATION_NODE`	12*	Представляет обозначение, объявленное в DTD-схеме XML-документа

* - устаревшие типы узлов

Таким образом, каждый элемент дерева имеет определенный тип. Некоторые элементы дерева являются своеобразными листьями, то есть не

могут иметь дочерних элементов. К таким относятся: `Text`, `CDATASection`, `Comment`, `Attribute`... Доступ к узлам дерева можно осуществлять через соответствующий им интерфейс. Например, ниже приведен интерфейс Element:

```
interface Element : Node {
  readonly attribute DOMString tagName;// тег элемента
//Получить атрибут по имени
 DOMString getAttribute(in DOMString name);
 //Установить значение атрибута
 void setAttribute(in DOMString name, in DOMString value);
 //Удалить атрибут по имени
 void removeAttribute(in DOMString name);
 //Аналогичный набор методов, представляющих атрибут как
 //объект Attr, а не строку
 Attr getAttributeNode(in DOMString name);
 Attr setAttributeNode(in Attr newAttr);
 Attr removeAttributeNode(in Attr oldAttr);
 //Получить список вложенных элементов по названию тега
 NodeList getElementsByTagName(in DOMString name);
  //Методы доступа к атрибутам с использованием
  //пространства имен
 DOMString  getAttributeNS(in DOMString namespaceURI,
                                  in DOMString localName);
 void setAttributeNS(in DOMString namespaceURI,
     in DOMString qualifiedName, in DOMString value);
 void removeAttributeNS(in DOMString namespaceURI,
                               in DOMString localName);
 Attr getAttributeNodeNS(in DOMString namespaceURI,
                                  in DOMString localName);
 Attr setAttributeNodeNS(in Attr newAttr);
 NodeList getElementsByTagNameNS(in DOMString namespaceURI,
                                 in DOMString localName);
 //Проверить присутствие атрибута
 boolean hasAttribute(in DOMString name);
 boolean hasAttributeNS(in DOMString namespaceURI,
                                in DOMString localName);
 readonly attribute TypeInfo schemaTypeInfo; //схема XML
 //Методы добавления пользовательских атрибутов элемента
 void setIdAttribute(in DOMString name, in boolean isId);
 void setIdAttributeNS(in DOMString namespaceURI,
             in DOMString localName, in boolean isId);
 void setIdAttributeNode(in Attr idAttr, in boolean isId);
};
```

Представленное выше определение содержит не все поля и методы, доступные для узлов-элементов DOM. Интерфейс Element объявлен как наследник более обобщенного интерфейса Node и для него доступны поля

и методы базового интерфейса в соответствии с принципами ООП. Подробнее со структурой интерфейсов узлов DOM можно ознакомиться в официальной документации [14].

Если вернуться к рассмотренному ранее фрагменту HTML-документа, слегка изменив его:

```
<!DOCTYPE html>
<html>
  <head>
     <!—Заголовок документа -->
     <title>
          Пример DOM-дерева
     </title>
  </head>
  <body>
     <!—Тело документа -->
     <div class="hello-block">
          Всем привет!
     </div>
  </body>
</html>
```

то дерево DOM тоже изменится и с учетом типов узлов будет выглядеть, как показано на рис. 2.6.

Как уже отмечалось, современное Web-приложение состоит из клиентской и серверной части. Для клиентской стороны характерна, прежде всего, манипуляция элементами HTML-страницы как узлами DOM-дерева (добавление, удаление, изменение содержимого на основе данных, полученных с серверной стороны). Реализуются подобные сценарии с использованием скриптов преимущественно на языке JavaScript, хотя доступны альтернативы, например, языки VBScript и JScript. JavaScript – это язык, разработанный фирмой Netscape для своего браузера Navigator в середине 90-х годов, на сегодня стал мощным, гибким инструментом, на котором разрабатываются frontend и backend-приложения для Web, поддержка которого включена во все современные браузеры. Язык активно развивается, в него добавляются все новые и новые возможности, расширяющие семантику и возможности его конструкций. В рамках пособия не представляется

возможным рассмотреть синтаксис этого языка, можно лишь сказать, что он во многих своих проявлениях (за исключением, возможно, реализации объектно-ориентированной парадигмы) похож на языки Java и C++. Для более близкого знакомства с JavaScript можно, например, обратиться к специальным изданиям [15].

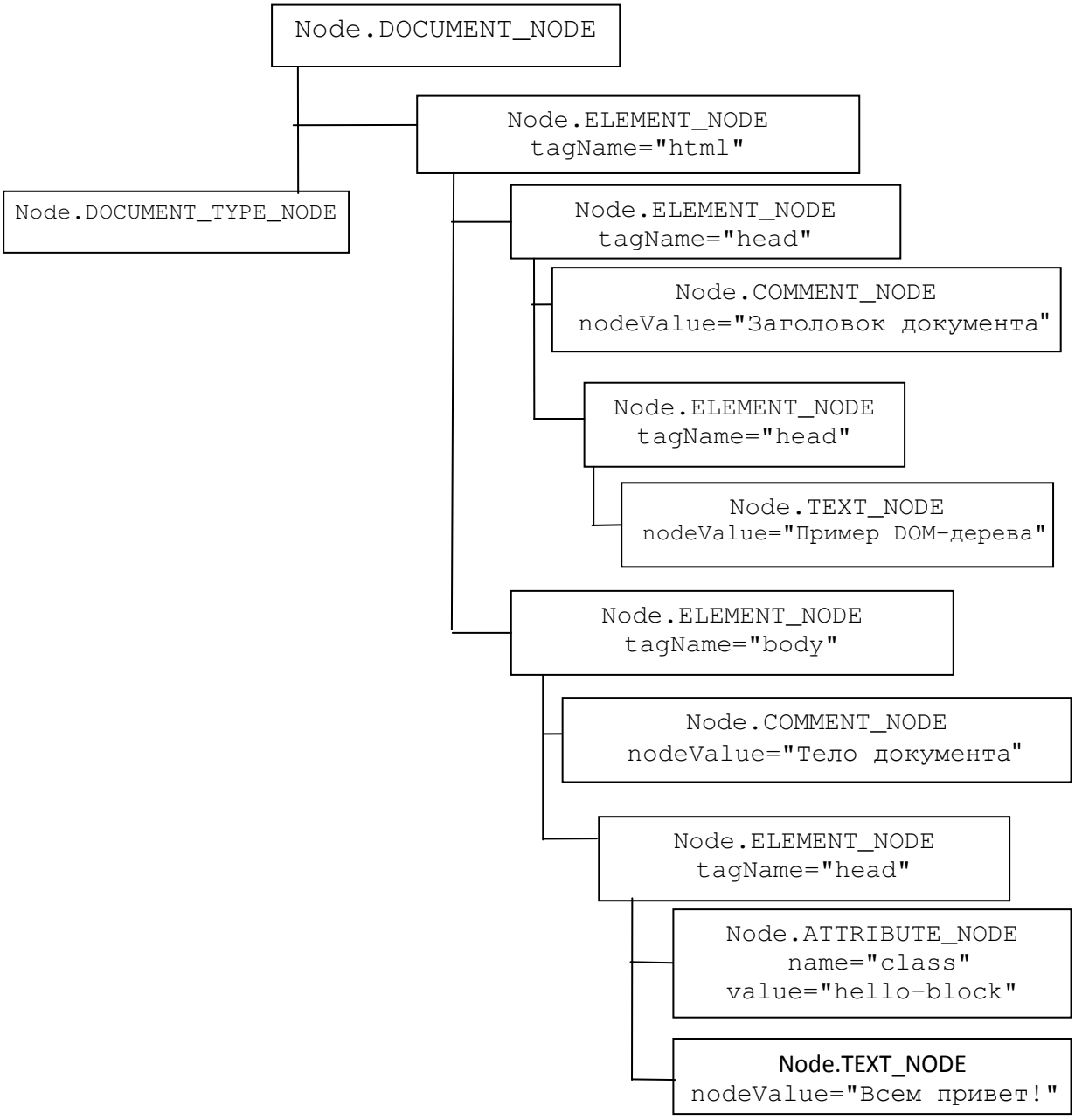


Рисунок 2.6 – Иерархия узлов DOM HTML-документа с учетом типов узлов

Для размещения сценариев JavaScript внутри документов HTML используется элемент script.

```
<script type="тип">
  //Тело скрипта
</script>
```

Для тега можно указать целый ряд атрибутов:

- атрибут `type` указывает MIME-тип содержимого скрипта. Для кода на языке JavaScript указываем `text/javascript`. Альтернативой может быть тип `text/vbscript` для кода на языке vbscript.

- в атрибуте `src` можно указать URI файла со сценарием, если скрипт содержится во внешнем по отношению к HTML-документу файле или даже на другом ресурсе:

```
<script type="text/javascript" src="myscript.js" </script>
```

- атрибут `defer` откладывает выполнение скрипта до момента полной загрузки страницы;

- используя атрибут `async` можно загружать скрипт асинхронно, то есть загрузка страницы и выполнение скрипта будут осуществляться одновременно.

Рассмотрим пример использования скрипта клиентской стороны на странице. Этот пример демонстрирует синтаксис подключения js-скрипта, а также пример кода на javascript, который перебирает поля и методы объекта window – ключевого объекта в иерархии объектов, доступных для скрипта объектов, который инкапсулирует состояние и поведение окна браузера и является корнем для всей иерархии DOM-узлов, а также объектов браузерной модели и ядра JavaScript. HTML-документ, подключающий скрипт, может выглядеть следующим образом:

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html;
 charset=utf-8">
<TITLE>Пример работы js-скрипта</TITLE>
<STYLE>
 * {margin: 0; padding: 0;}
 .object {font-weight: bold; font-size: 19px;}
 .object p {font-weight: 100; font-size: 15px;}
```

```
</STYLE>
</HEAD>
<BODY>
</BODY>
<SCRIPT language="javascript" src="js/script.js">
</SCRIPT>

 </HTML>
```

Файл со скриптом script.js в этом случае должен находиться в папке js:

```
function enumObject(object) {
document.writeln("Компоненты объекта "+object.toString());
for(var el in object) {
  switch(typeof object[el]) {
    case'object': document.writeln("Объект "+el+" " ); break;
    case  'function':  document.writeln("Функция  "+el+"  "  );
break;
    default: document.writeln("Свойство "+el+""); } }
  document.writeln("");
 return true; }
 document.writeln(" ");
 enumObject(window);
 document.writeln("");
```

В Web-приложениях скрипты выполняют следующие задачи:

- валидация форм перед отправкой на сервер;
- манипуляция элементами документа для создания визуальных эффектов;
- асинхронное взаимодействие клиентской и серверной части (AJAX);
- рисование на канве (поддерживается как двумерная, так и трехмерная графика);
- обработка событий для реализации логики приложения.

Приведенный список далеко не полон, стандарт HTML5 предлагает широкий выбор JavaScript-API для решения самого широкого круга задач: работа с мультимедиа, геолокация, поддержка локальных хранилищ данных на клиентской стороне, поддержка двунаправленной передачи данных с использованием WebSocket, параллельное выполнение скриптов [6].

Как уже было отмечено, скрипту доступны объекты из трех источников:

– DOM представляет собой объектную иерархию объектов документа с корневым элементом-контейнером `document`;

– BOM – Browser Object Model (Объектная Модель Браузера) – набор объектов, описывающих сущности браузера (`navigator, history, location, frames` и др.);

– JavaScript Core – ядро интерпретатора, в котором определены стандартные объекты языка – `Object, Array, String, Array, Function, ...`

В скрипте единой точкой доступа ко всем этим объектам является объект `window`. Если какой-либо объект (например, `document` или `navigator`) используется в скрипте без упоминания объекта window, то в контексте выполнения в браузере использование `window` подразумевается по умолчанию. Фрагмент иерархия доступных js-скрипту объектов графически иллюстрируется на рис. 2.7.

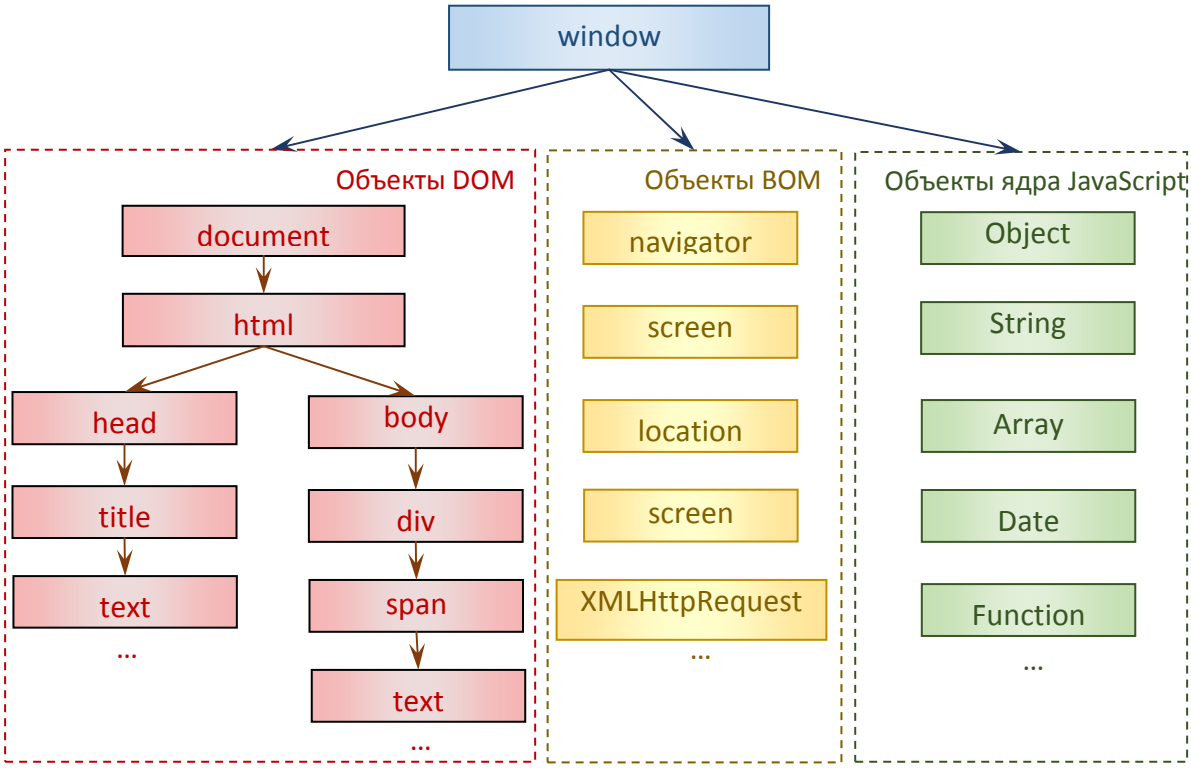


Рисунок 2.7 – Программное окружение браузерного скрипта

Скрипт может получить доступ к любому элементу дерева DOM и для него:

- Получить/изменить атрибуты,
- Получить/изменить контент,
- Добавить дочерние элементы,
- Удалить элемент,
- Настроить обработку событий.

Точкой опоры в дереве DOM для скрипта является элемент `document`. От этого элемента можно получить доступ к элементу `documentElement`, представляющему собой тег `<html>`. Прямое соответствие продолжается элементами `head` и `body`:

```
alert(document.documentElement.body.innerHTML);
```

Начиная с объекта `documentElement`, скрипт может использовать модель HTML-элемента, позволяющую как получить доступ к внутреннему состоянию элемента, так и осуществлять навигацию по дереву DOM, получив ссылку на дочерние, соседние, родительские элементы относительно текущего. Графически эта модель представлена на рис. 2.8. Как следует из этого рисунка, для элемента определен ряд свойств, которые позволяют получить или изменить ту или иную характеристику элемента, а также осуществлять навигацию по дереву DOM.

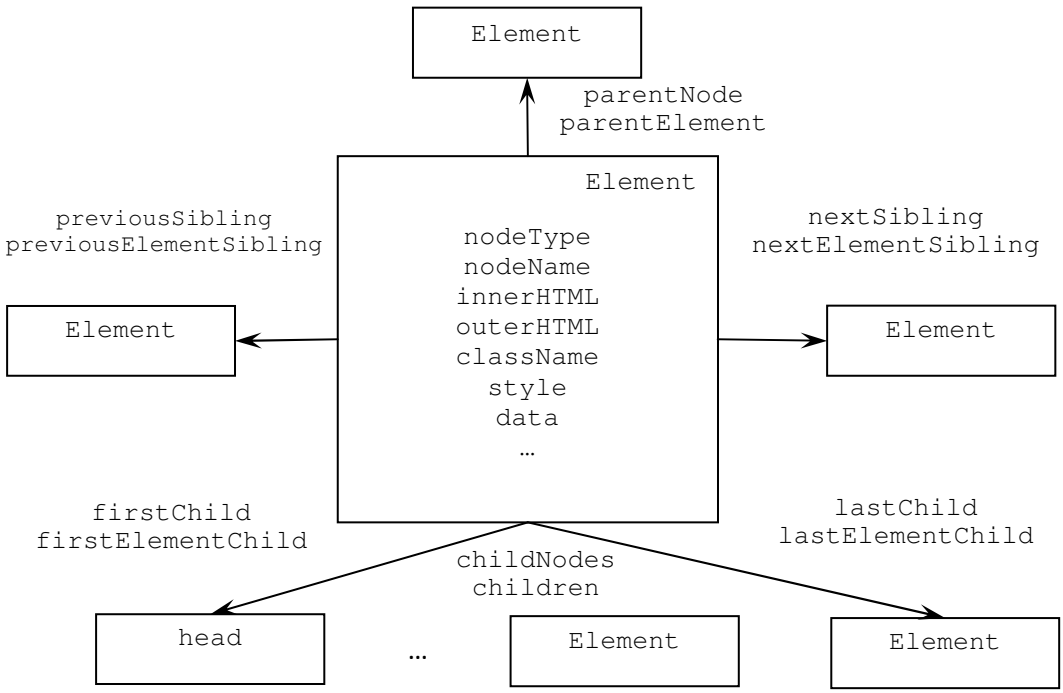


Рисунок 2.8 – Программная модель DOM-объекта

Для навигации можно использовать ссылки на связанные с узлом элементы. Рассмотрим подробнее их назначение:

Таблица 2.8. Навигационные свойства элементов DOM

Назначение свойства	С учетом текстовых узлов и комментариев	Без учета текстовых узлов и комментариев
Ссылка на родительский элемент	`parentNode`	`parentElement` отличается по значению от `parentNode` только у `documentElement`
Следующий элемент (сосед в дереве справа)	`nextSibling`	`nextElementSibling`
Предыдущий элемент (сосед в дереве слева)	`previousSibling`	`previousElementSibling`
Коллекция дочерних элементов	`childNodes`	`children`
Первый дочерний элемент	`firstChild`	`firstElementChild`
Последний дочерний элемент	`lastChild`	`lastElementChild`

Необходимо отметить, что все свойства из таблицы 2.8 предназначены только для чтения, их изменение для целей добавле-

ния/перемещения/удаления элементов в дереве DOM недопустимо, для этих целей используются специализированные методы, которые будут рассмотрены далее.

Помимо навигационных свойств, у элементов дерева DOM доступны свойства, отражающие их внутренне состояние. Перечислим универсальные свойства этого класса для DOM-элементов, которые можно применять в контексте широкого спектра типов элементов:

`outerHTML`	HTML-элемент целиком (вместе с обрамляющими тегами)
`innerHTML`	позволяет получить HTML-содержимое элемента в виде строки, доступно как для чтения, так и для записи
`textContent`	Текстовое содержимое элемента за вычетом всех тегов
`style`	Доступ к стилю элемента
`nodeName`	Название элемента узла
`className`	Имя класса элемента

У элементов конкретных типов могут быть дополнительные свойства, например, `href` у тега-ссылки `a` или `src` у тега-картинки `img`.

Таким образом, для нашего документа со стр. 75 можно изменить текст в блоке `div` следующим образом:

```
document.body.children[0] = "hello world"
```

А такой командой можно изменить первый комментарий внутри тега `body`:

```
document.body.childNodes[0]=”Новый текст комментария”
```

Однако перебирать все узлы дерева DOM от корня в поиске целевого для манипуляции затруднительно. Действительно, очень сложно писать и разбирать код такого вида:

```
let nDiv = document.body.children[2].children[0].children[3];
```

Последовательный доступ неудобен, поскольку требует прохождения по всей цепочке отношений «родитель – потомок» для доступа к требуемому элементу. Существует возможность прямого доступа к элементу.

Для этого используется функция `getElementById` Она позволяет обращаться к элементам по их идентификатору (атрибут `id`), например:

```
var txtInfo=document.getElementById("pInfo1").innerHTML;
document.getElementById("pInfo2").innerHTML="Новое содержимое";
```

Кроме того, существует возможность поиска элементов по названию тега `getElementsByTagName(TagName`. Эта функция по очевидным причинам возвращает не ссылку на единичный элемент, а целую коллекцию элементов дерева, имеющих указанный тег. При этом поиск можно осуществлять относительно некоторого конкретного узла, локализуя область отбора только вложенными элементами:

```
let newsBlock=document.getElementById("newsWrapper");
let els=newsBlock.getElementsByTagName("span");
for(i=0; i<els.length; i++)
{  //работа с i-м элементом коллекции по ссылке els[i]
}
```

В качестве критерия отбора при поиске может выступать имя стилевого класса (атрибут class) элемента(-ов). При таком поиске используется функция `getElementsByClassName(ClassName)` и функция возвращает коллекцию ссылок на все элементы, дочерние по отношению к указанному в качестве корневого при поиске, для которых установлен стилевой класс `ClassName`:

```
let menuItems=document.getElementByClassName("menuItem");
```

Наконец, еще одна опция получения ссылок на элементы DOM-дерева – поиск по селектору. В этом случае используются функции `querySelectorAll(selector)`, которая возвращает коллекцию элементов, соответствующих указанному селектору, или `querySelector(selector)`, возвращающая только первый отобранный элемент. В качестве параметра этих функций указывается селектор элементов, составленный по тем же принципам, что и селектор CSS-правил (см. таблицу 2.2).

```
var blocks=document.querySelectorAll("div>p");
```

```
var elm=document.querySelector (".news, .articles");
```

Скрипт клиентской стороны допускает не только просмотр текущего состояния дерева DOM документа, спецификация предоставляет средства для манипуляции его элементами: добавления, перемещения, удаления элементов. Для этого можно использовать методы:

`createElement(тег)`	Создание нового элемента `var el=document.createElement("div");`
`appendChild(элемент)`	Добавление дочернего элемента в конец указанного родителя `parent.appendChild(elem);`
`insertBefore(вставляемый элемент, существующий элемент)`	Вставка элемента в коллекцию дочерних элементов перед заданным элементом `parent.insertBefore(elem, nextElem);`

`cloneNode()`	Клонировать (создать полную копию)
	`var el=document.getElementById("some");`
	`var copy=el.cloneNode();`
	`el.parentNode.insertBefore(copy, el);`

`removeChild(элемент)` `replaceChild(элемент, заменяющий элемент)`	Удалить/заменить дочерний элемент `el.parentNode.removeChild(el);`

Следующий фрагмент иллюстрирует применение некоторых из указанных методов: в документе объявляются две ссылки, щелчки по которым удаляют или вставляют элемент в список с идентификатором `"list"`.

```
<script>
  function addElem()
  { let el=document.createElement("li");
    el.innerHTML="Новый элемент";
    el.style.color="red";
    list.appendChild(el);
  }
  function removeElem()
  { list.removeChild(list.lastChild);}
</script>
<ol id="list">
  <li>0</li>
  <li>1</li>
```

```
</ol>
<a href="javascript:addElem()">Добавить</a><br />
<a href="javascript:removeElem()">Удалить</a>
```

Одним из ключевых требований к скриптам клиентской стороны Web-приложений является возможность интерактивного взаимодействия с ними, предполагающего отклик элементов документа на действия пользователя (перемещения курсора мыши, щелчки по кнопкам мыши, нажатия клавиш на клавиатуре) или реагирование в ключевые моменты жизненного цикла приложения (например, окончание загрузки документа, закрытие окна браузера). Достичь подобной интерактивности помогает механизм событий, который также является частью спецификации DOM. Событие сообщает скрипту о том, что в окружении браузера с открытым документом произошло некоторое действие (пользователь работает с мышью или клавиатурой, элемент обрел ли потерял фокус, истек период ожидания и др.). Благодаря поддержке браузером спецификации механизма событий DOM во время выполнения скрипт оповещается о том, что произошло в его окружении. Это оповещение со стороны браузера имеет вид программного объекта типа `Event` или унаследованного от него (например, `KeyboardEvent`, `AnimationEvent`, `DragEvent`), свойства которого позволяют узнать детали произошедшего (например, координаты курсора мыши или код нажатой клавиши), а методы помогают реализовать реакцию скрипта.

Событие характеризуется:

- типом события, описывающим, что произошло (`mousemove`, `click`, `submit`, `keydown`, …);
- целью события, описывающим, какому объекту пришло событие;
- обработчиком события – функцией, реализующей код обработки события;
- объектом события, описывающим параметры события (код нажатой клавиши, координаты курсора мыши и др.);

Для того чтобы тот или иной элемент мог реагировать на возникающие события, для него необходимо зарегистрировать обработчик события. Сделать это можно несколькими способами [16]:

1. С использованием атрибута элемента. В спецификации HTML для объявления обработчиков событий определены специальные атрибуты вида `on<событие>` (например, `onclick`, `onsubmit`, `onblur`), которым можно присвоить ссылку на метод или код, который будет обрабатывать соответствующее событие. Сделать это можно, напрямую присвоив атрибуту-событию строку с исполняемым кодом:

```
<input onfocus="this.value=''" />
```

В приведенном примере элемент-строка ввода очищает свое содержимое (атрибут `value`) в момент получения фокуса ввода (события `focus`). Объект `this` в данном фрагменте будет автоматически инициализирован ссылкой на DOM-объект, который вызвал обработчик (на котором сработало событие). Если код обработчика слишком велик для размещения внутри тега, его можно вынести в отдельную функцию, а код обработки оформить как вызов этой функции:

```
<script>
  function handlerClick()
   { alert ("Был щелчок!!!"); } //выводим информационное
                                //окно в браузере
</script>
…
<div onclick="javascript:handlerClick()">
Кликни здесь!</div>
```

2. Связать функцию-обработчик события с конкретным элементом, для которого необходимо отслеживать и обрабатывать событие, можно в коде скрипта, изменяя значение свойства DOM-элемента:

```
document.getElementById("name").onfocus=handleFunc;
```

В качестве обработчика можно объявить, в том числе, и анонимную функцию:

```
document.getElementById("name").onfocus=()=>{console.log("Полу
чен фокус ввода!")}; //Вывести сообщение в консоль браузера
```

```
                    //при обретении фокуса элементом с id="name"
```

Если несколько элементов должны реагировать на события единообразно, то допустимо присваивать один и тот же обработчик свойствам различных элементов:

```
let els=document.getElementsByTagName("div");
for(i=0; i<els.length; i++)
  {
    els[i].onclick=handlerClick;
  }
```

В любой момент обработчик может быть изменен или даже отменена обработка события элементом путем сброса в `null` свойства `on<событие>`:

```
elem.onclick=null;
```

3. Все перечисленные выше способы не позволяют связать с одним событием несколько обработчиков. Между тем, такая возможность позволит гибко настраивать поведение объектов, динамически добавляя или удаляя им функционал реагирования в зависимости от контекста исполнения кода. Добавить несколько обработчиков одному событию элемента можно, используя функцию:

```
element.addEventListener(событие, обработчик[, фаза=false]);
```

где `событие` – тип события, для которого определяется обработчик;

`обработчик` – функция, которая будет обрабатывать возникающие события;

`фаза` – определяет, в какой фазе будет обработано событие: событие вложенного элемента перехватывается (при `true`) или будет обработано на этапе всплытия события (при `false`). Вот как можно привязать к событию щелчка по элементу с идентификатором `"sp1"` две функции, которые будут выполнены последовательно, одна за другой:

```
var sp1=document.getElementById("sp1");
function clickFunc2()
{   this.innerHTML="было нажатие"; }
function clickFunc()
{   this.style.backgroundColor = "#CCC";}
sp1.addEventListener("click", clickFunc);
```

```
sp1.addEventListener("click", clickFunc2, );
```

При этом в дальнейшем, если один из методов обработки событий станет неактуальным, его можно удалить из списка обработчиков:

```
sp1.removeEventListener("click", clickFunc);
```

Список событий, доступных для обработки элементами DOM-дерева, весьма обширен, рассмотрим наиболее часто востребованные:

- события мыши: `mousedown` (нажата кнопка мыши), `mouseup` (отпущена кнопка мыши), `mouseover` (курсор мыши наведен на элемент), `mouseout` (курсор мыши выходит за границы элемента), `mousemove` (курсор мыши перемещается над элементом), `click` (щелчок левой кнопкой мыши на элементе), `contextmenu` (щелчок правой кнопкой мыши на элементе), `dblclick` (двойной щелчок мыши на элементе), `wheel` (прокрутка колесика мыши);

- события клавиатуры: `keydown` (нажата клавиша на клавиатуре в момент, когда элемент имеет фокус ввода), `keypress` (нажата символьная клавиша на клавиатуре), `keyup` (отпущена ранее нажатая клавиша на клавиатуре);

- события элементов управления: `focus` (элемент получил фокус ввода, например, щелчком мыши или нажатием клавиши `Tab`), `blur` (элемент управления потерял фокус ввода), `submit` (отправлена форма `<form>`);

- события загрузки документа: `DOMContentLoaded` (браузер загрузил HTML-документ и построил дерево DOM), `load` (в плюс к предыдущему событию еще и загружены все ресурсы, может возникать также, например, у элемента с тегом `img`), `beforeunload, unload` (пользователь покидает страницу);

Приведенный список далеко не полон, каждый из многообразных API спецификации HTML предлагает свой набор событий. Примерами здесь могут быть события механизма перетаскивания (drag’n’drop)

dragstart и dragover, соответствующие моменту начала и окончания перетаскивания элемента мышью, или событие pause мультимедийных элементов, соответствующее остановке воспроизведения контента. Более подробную информацию о типах событий, особенностях их возникновения и обработки можно получить, например, здесь [17].

Если же говорить об обработке событий обобщенно, то нельзя не отметить объект event, доступный обработчикам событий. Этот объект является реализацией интерфейса Event или одного из его наследников. Свойства данного объекта содержат обобщенную информацию о событии, а методы позволяют управлять процессом его обработки. Ссылка на объект event является первым параметром обработчика события:

```
elem.addEventListener('click', function(event) {
//event хранит информацию о событии
});
```

Рассмотрим свойства и методы объекта Event:

- Event.type – название события;
- Event.target - ссылка на целевой объект, на котором произошло событие;
- Event.eventPhase - фаза обработки события;
- Event.timeStamp – время возникновения события;
- Event.bubble – событие может всплывать в соответствующей фазе (см. подробности далее);
- Event.cancelable – событие можно отменить вызовом preventDefault (см. подробности далее);
- Event.isTrusted – создано ли события в браузере или имитировано вызовом Event.initEvent();
- Event.preventDefault() – отмена события;
- Event.stopPropagaion() – отмена распространения события в дереве DOM;

- `Event.Event(имя_события)` – позволяет создать пользовательское событие или имитировать возникновение стандартного.

Кроме перечисленных выше, конкретный экземпляр события event, пришедший на обработку, может иметь еще целый ряд дополнительных свойств, но они характеризуют событие конкретного типа, их описание и даже перечисление занимает очень большой объем, поэтому с ними лучше ознакомиться в справочной документации. Для примера можно перечислить дополнительные параметры, которые описываются в свойствах события, связанного с мышью:

`clientX`, `clientY`	Координаты курсора мыши относительно левого верхнего угла клиентской области окна браузера
`pageX`, `pageY`	Координаты курсора мыши относительно левого верхнего угла документа (с учетом величин прокрутки)
`which`	Какая кнопка мыши была нажата (1-левая, 2 – средняя, 3 - правая)
`shiftKey`, `altKey`, `ctrlKey`	Была ли нажата соответствующая клавиша

Еще одним важным аспектом распространения и обработки событий в DOM-дереве документа является `фаза распространения`. На самом деле событие не возникает для одного конкретного элемента. Оно проходит по иерархии дерева DOM в рамках 3-х фаз распространения:

- `Фаза перехвата` – от корня DOM к цели;
- `Фаза цели` – срабатывание события для целевого элемента;
- `Фаза всплытия` – событие идет обратно от элемента к корню DOM.

Весь путь объекта события вдоль иерархии DOM проиллюстрирован на рис. 2.9. Здесь предполагается, что произошло событие на первой ячей-

ке второй строки таблицы (например, щелчок по кнопке мыши в момент, когда курсор мыши находился над этой ячейкой).

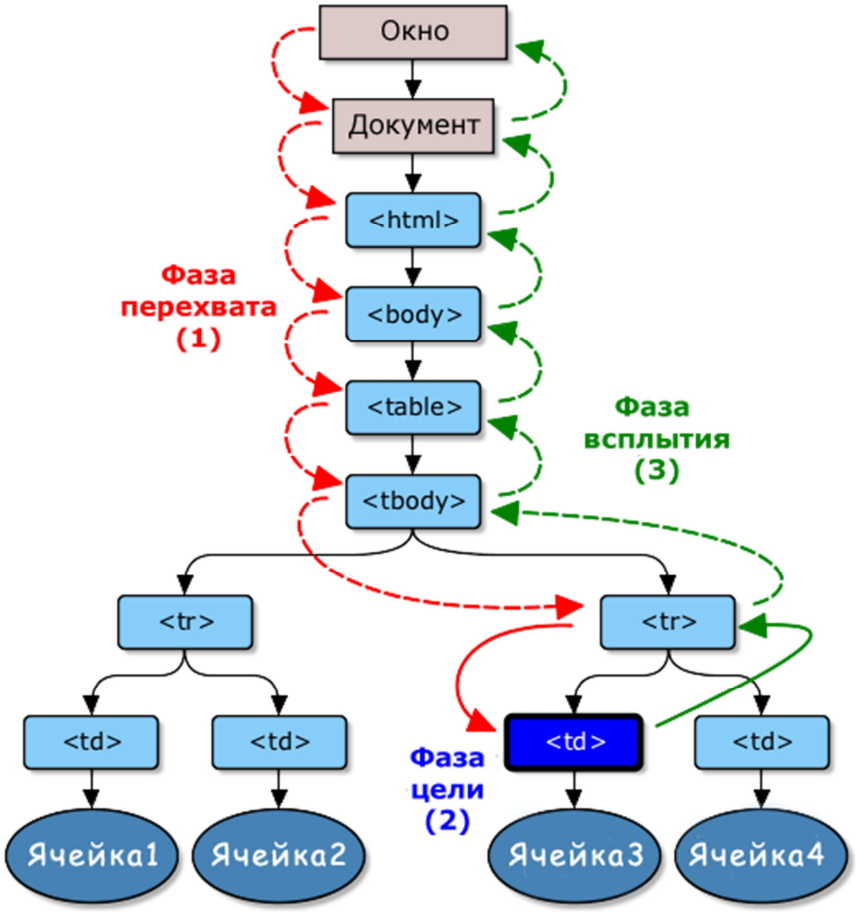


Рисунок 2.9 — Фазы распространения события в документе

Как следует из рисунка, событие проходит по всем элементам, являющимся прямыми или косвенными родителями целевому (фаза распространения), доходит до самой ячейки (фаза цели), после чего в рамках фазы всплытия идет обратно до объекта `window`. Необходимо отметить, что каждый из элементов может обработать это событие, после чего оно продолжит свой путь. Распространена практика, когда обработчик события для всех дочерних элементов регистрируется в родительском элементе. Узнать, какому конкретно их дочерних элементов направлялось событие можно, анализируя поле `target` объекта `event`. Такой подход, получивший название `делегирование обработки`, позволяет не регистрировать

обработчик во множестве дочерних элементов, если они должны реагировать на событие единообразно.

Распространение события вверх и вниз по дереву DOM можно прервать, вызвав в обработчике соответствующего элемента метод `stopPropagation`:

```
element.stopPropagation();
```

Еще один часто используемый метод `preventDefault` позволяет предотвратить обработку события, принятую в браузере по умолчанию. Например, вот так можно предотвратить автоматический переход при щелчке по ссылке:

```
var linkelem = document.querySelectorAll(".info>a");
linkelem.addEventListener('click', function(event) {
  event.preventDefault();
  //свой вариант обработки
});
```

Еще одной замечательной возможностью, предлагаемой стандартом DOM, является генерация пользовательских событий. Для этого необходимо создать новый тип события с использованием шаблона `CustomEvent`, после чего допустимо подписываться и программно генерировать события подобного типа:

```
var someEvent = new CustomEvent("someevent", {
  params:{count:0},
  bubbles: true,  //событие всплывающее
  cancelable: false //нельзя отменить обработку по умолчанию
});

// регистрируем обработчик события 'someevent'
elem.addEventListener('someevent', function(event) {
//Обработка события
});

// Программно инициируем событие 'someevent'
someevent.params.count=10;
document.dispatchEvent(someEvent);
```

Метод `dispatchEvent` в приведенном примере вызывает (программно имитирует) событие. Его можно применять и для стандартных событий (`click`, `keydown`, `submit` и т.д.). В этом случае событие будет

распространяться и обрабатываться так же, как сгенерированное браузером, единственное отличие программно сымитированных событий – у них сброшено в `false` свойство `isTrusted`.

Работа скриптов клиентской стороны очень плотно завязана на манипуляцию с DOM-элементами, но ими не ограничивается. Как уже было отмечено (2.7), окружение скрипта составляют также объекты ядра javascript и BOM (Browser Object Model). В частности, объекты BOM, доступные в скрипте клиентской стороны как свойства объекта `window`, это:

- `navigator` – объект, предоставляющий информацию о браузере, в котором исполняется скрипт. Свойства этого объекта позволяют узнать версию и текущие настройки браузера, исполняющего скрипт, для целей, например, адаптации кода к среде:

```
if(window.navigator.cookieEnabled){//проверка доступности
//использования cookie в браузере
//работа с cookie
}
```

- `history` – средство навигации по посещенным пользователем страницам. Свойство `length` этого объекта хранит количество посещенных в данном сеансе окна браузера страниц, а методы `go()`, `forward()` и `back()` позволяют перемещаться вперед и назад по журналу посещений страниц аналогично тому, как это происходит при нажатии соответствующих кнопок на панели самого браузера. Вот как можно добавить на страницу кнопку, позволяющую вернуться к предыдущей просмотренной странице:

```
<button onclick="history.back()">Вернуться назад</button>
```

- `location` – объект, содержащий URL-адрес документа, загруженного в браузер. Свойства и методы этого объекта позволяют получить доступ к отдельным частям URL-адреса (`host`, `hostname`, `search`, `protocol`, `port`, `pathname`), а также сменить URL-адрес, тем

самым перезагрузив страницу. Например, следующие фрагменты кода позволяют из js-скрипта сменить текущую страницу в браузере:

```
location.href= "http://www.volpi.ru";
location.assign("another.com"); //сменить текущую страницу
location.replace("some.com");//сменить текущую страницу
//без запоминания в журнале браузера
```

- `screen` – предоставляет информацию о характеристиках экрана устройства отображения клиента. Свойства этого объекта хранят, например, высоту (`height`) и ширину (`width`) экрана в пикселях или глубину цвета, установленную для экрана (`colorDepth`).

```
sWidth=screen.width; //получаем ширину экрана в пикселях
```

Рамки пособия не позволяют подробно рассмотреть все объекты BOM. О некоторых из них пойдет рассказ в следующей части пособия (например, объект `XMLHttpRequest`, позволяющий осуществлять асинхронный обмен данными между клиентскими и серверными скриптами Web-приложения). Подробнее о свойствах и методах объектов BOM можно узнать в [15].

Возможности классического DHTML недостаточно гибки и функциональны для реализации больших проектов – зачастую разработчику хотелось бы иметь более выразительные средства для доступа к атрибутам элементов и их изменения, манипуляции деревом DOM, взаимодействия с серверной стороной. При этом большой проблемой остается неравномерное внедрение тех или иных стандартов в различных браузерах, в связи с чем для обеспечения идентичной функциональности в разных типах и версиях браузеров зачастую приходится писать свой фрагмент кода под каждый браузер. Облегчить труд программиста могут JavaScript – фреймворки – библиотеки классов и методов, облегчающих типичные операции FrontEnd-сценариев: поиск по дереву DOM, доступ к элементам документа, обработку событий, реализацию визуальных эффектов, AJAX-

взаимодействие с сервером, обеспечение кроссбраузерности. Примерами подобных библиотек являются jQuery, VanillaJs, Dojo.

Рассмотрим пример скрипта, использующего библиотеку jQuery. Этот фреймворк был написан в 2006 году Джоном Резигом и сегодня является одной из самых популярных библиотек своего класса. Код, использующий jQuery, выглядит очень компактным, простым, сам решает проблемы кроссбраузерности тех или иных вызовов.

Подключить jQuery к документу можно или скачав файл с библиотекой с сайта http://jquery.com/ и включив в документ тег:

```
<script src="/scripts/jquery.js"></script>
```

или используя в теге `script` ссылку на файл с библиотекой в сети доставки контента (Content Delivery Network, CDN), например:

```
<script src="https://code.jquery.com/jquery-3.4.0.min.js"
crossorigin="anonymous"></script>
```

Вся функциональность библиотеки основана на использовании объекта `jQuery()` (или его псевдонима `$()`), методы и свойства которого позволяют решать широкий круг frontend-задач. Подробное знакомство с jQuery выходит за рамки пособия, интересующимся можно посоветовать прочитать книги на эту тему, например [18]. Здесь приведем лишь скрипт, решающий конкретную задачу: реализацию прилипающего к верхнему краю окна браузера при прокрутке документа блока. Этот скрипт дает представление о способах манипуляции элементами и их атрибутами, а также средствах обработки событий, принятых в jQuery.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; char-
set=utf-8" />
<title>Прилипающий блок</title>
<style>
#dSticky {
  background-color: green;
  width: 100%;
}
```

```
#dContent{
  min-height: 1000px;
}
</style>
<!— Подключаем jQuery -->
<script src="https://code.jquery.com/jquery-3.4.0.min.js"
crossorigin="anonymous"></script>
<script>
$(function() {   //когда загрузится весь документ
//Получаем высоту блока с id=dSticky и координаты его
//верхней границы
    var fp_height = $('#dSticky').height();
    var fp_pos = $('#dSticky').position().top;
// Определяем обработчик события прокрутки содержимого окна
    $(window).scroll(function () {
//если окно прокручено на величину, большую верхнего
//смещения блока
    if ($(window).scrollTop() >= fp_pos)
// задаем блоку фиксированное позиционирование и прижимаем к
//верхнему краю окна
   $('#dSticky').css({'position': 'fixed', 'top': '0px'});
        else
//иначе – возвращаем блок в поток
     $('#dSticky').css({'position': 'static'});
  });
});
</script>
</head>
  <div id="dHead">Заголовок</div>
  <div id="dSticky">Плавающий блок</div>
  <div id="dContent">Контент</div>
<body>
</html>
```

Польза jQuery в клиентском скрипте является предметом дискуссий, некоторые разработчики считают (и небезосновательно), что он замедляет работу скрипта, возможности современных функций API HTML-стандартов достаточно функциональны, а поддержка браузерами стандартов стала гораздо более последовательной, и это может вызвать сомнения в целесообразности использования jQuery или его аналогов, но в любом случае выбор остается за разработчиком, а сделать его осознанно можно, лишь зная предмет, поэтому более подробное знакомство с этим фреймворком не будет лишним в процессе профессионального роста.

Библиотека jQuery и подобные ей очень удобны, но они решают лишь локальную задачу упрощения типовых манипуляций с элементами DOM. Сегодня на рынке frontend-разработки популярны библиотеки, которые решают более комплексные задачи: задают архитектуру приложения, декомпозируют его на обособленные модули, формируют семантически высокоуровневые теги, из которых может строиться логика представления и обработки документов. Примерами подобных библиотек являются BackBone, React, Anigular, Vue. Использование этих библиотек позволяет создавать легко масштабируемые SPA (Single Page Application)-приложения, с простым и гибким интерфейсом, они используют объектно-ориентированный подход и опираются на эффективные паттерны проектирования, проверенные временем; вокруг этих библиотек-технологий организовались широкие коммьюнити, помогающие новичкам и развивающие сам инструмент; наконец, рынок труда демонстрирует большой интерес работодателей к специалистам, владеющим опытом разработки с использованием перечисленных библиотек. Более подробно о принципах работы современных frontend-фреймворков речь пойдет в следующих частях пособия, желающие познакомиться с ними самостоятельно могут найти информацию в источниках [19, 20].

Заключение

В первой части пособия, посвященного разработке приложений для работы в среде глобальной сети, авторы постарались изложить фундаментальные основы, на которых базируется эта отрасль программирования. Протоколы транспортировки данных между частями Web-приложения, языки формирования семантики и представления данных в рамках этих приложений, динамика изменения и обработки информации на клиентской стороне – вот краткий перечень рассмотренных вопросов. Несмотря на то, что он охватывает лишь малую часть всего, что надо знать современному разработчику, авторы постоянно испытывали затруднения в несоответствии объема информации, который бы в полной степени описывал затронутые темы, и ограниченных рамок учебного пособия. В связи с этим некоторые вопросы были освещены лишь вскользь, для многих аспектов разработки даны ссылки на дополнительные источники, которые дадут возможность читателю подробнее ознакомиться с материалом. Материал пособия должен стать фундаментом, на котором интересующийся читатель сможет в дальнейшем построить здание своих профессиональных знаний в области Web-разработки, чего авторы искренне желают. При этом в самом пособии точка еще не поставлена, в следующих его частях будут рассмотрены, в частности, вопросы разработки скриптов серверной стороны, архитектур современных Web-систем, безопасности работы Web-приложений.

Список литературы

1. Стандарт HTML 5.2 на сайте w3c. — Режим доступа: https://www.w3.org/TR/html52/
2. Лясин Д.Н., Саньков С.Г. Сети и телекоммуникации. Лабораторный практикум. Часть 1: методические указания — Волжский: ВПИ (филиал) ВолгГТУ,2016
3. Д.Н. Лясин, С.Г. Саньков, А.В. Степанова. Защита информации (часть 2): учебное пособие. – Волжский (филиал) ВолгГТУ, 2016г.
4. Online-справочник по языку HTML. — Режим доступа: http://htmlbook.ru/html
5. Валидатор разметки на сайте w3c. Режим доступа — https://validator.w3.org/
6. Х.Д. Гоше. HTML5. Для профессионалов. 2-е изд. — СПб.: Питер, 2015. — 560с.: ил.
7. Э. Майер, У. Эстелл. CSS. Полный справочник. Визуальное форматирование страниц. — М.: Вильямс, 2019. – 1088 с.
8. Cascading Style Sheet home page on w3c. Режим доступа — https://www.w3.org/Style/CSS/
9. Online-справочник по языку CSS. — Режим доступа: http://htmlbook.ru/css
10. Описание технологии Flexbox на сайте Mozilla Development Network. — Режим доступа: https://developer.mozilla.org/ru/docs/Learn/CSS/CSS_layout/Flexbox
11. Описание технологии CSS Grid Layout на сайте Mozilla Development Network. — Режим доступа: https://developer.mozilla.org/ru/docs/Web/CSS/CSS_Grid_Layout
12. Документация по фреймворку Bootstrap 4. — Режим доступа: https://bootstrap-4.ru/

13. Репозитарий sass-транслятора на GitHub. — Режим доступа: https://github.com/sass/dart-sass/releases.
14. Document Object Model Core on w3c — Режим доступа : https://www.w3.org/TR/DOM-Level-3-Core/core.html
15. Н. Закас. JavaScript для профессиональных разработчиков. — СПб.: Питер, 2015. — 960с.
16. Лясин, Д.Н. Создание динамических HTML-документов с использованием технологии Dynamic HTML: методические указания / Д.Н. Лясин; ВПИ (филиал) ВолгГТУ. – Волгоград, 2018. – 30 с.
17. Спецификация: Объектная модель документа уровень 3, События, W3C/ — Режим доступа: https://developer.mozilla.org/en-US/docs/Web/Events
18. Б. Бибо, И. Кац, А. де Роза. jQuery в действии. — СПб.: Питер, 2017. – 528с.
19. П. Дилеман. Изучаем Anigular 2. — М.: ДМК Пресс, 2017. — 354с.
20. М. Азат. React быстро. Веб-приложения на React, JSX, Redux и GraphQL- СПб.: Питер, 2019. — 560с.

Электронное учебное издание

Дмитрий Николаевич **Лясин**
Оксана Федоровна **Абрамова**

ОСНОВЫ ПРОЕКТИРОВАНИЯ WEB-ПРИЛОЖЕНИЙ

Учебное пособие

Электронное издание сетевого распространения

Редактор Матвеева Н.И.

Темплан 2019 г. Поз. № 18.
Подписано к использованию 21.05.2019. Формат 60x84 1/16.
Гарнитура Times. Усл. печ. л. 6,31.

Волгоградский государственный технический университет.
400005, г. Волгоград, пр. Ленина, 28, корп. 1.

ВПИ (филиал) ВолгГТУ.
404121, г. Волжский, ул. Энгельса, 42а.