**МЕТОДИЧЕСКИЕ УКАЗАНИЯ**  
**по дисциплине бакалавриата**

**«Разработка клиент-серверных приложений»**

**(направление подготовки:**

**09.03.04 «Программная инженерия»)**

**Содержание**

[**Введение** 4](#_Toc48604375)

[**﻿1 ﻿Основы ﻿проектирования ﻿клиент-серверных приложений** 9](#_Toc48604376)

[1.1 Основные понятия клиент-серверных приложений 9](#_Toc48604377)

[1.1.1 Интернет и его особенности 9](#_Toc48604378)

[1.1.2 Клиент-серверные приложения 25](#_Toc48604379)

[**2 Технологии построения клиент-серверных приложений** 34](#_Toc48604380)

[2.1 Технология создания клиентской части 35](#_Toc48604381)

[2.1.1 HyperText Markup Language (HTML) 35](#_Toc48604382)

[2.1.2 Cascading Style Sheets (CSS) 38](#_Toc48604383)

[2.1.3 DOM (Document Object Model) 41](#_Toc48604384)

[2.1.4 JavaScript 44](#_Toc48604385)

[2.1.5 JQuery 46](#_Toc48604386)

[2.1.6 AJAX 47](#_Toc48604387)

[2.2 Технологии создания серверной части 48](#_Toc48604388)

[2.2.1 Web-серверы 48](#_Toc48604389)

[2.2.2 Технология MVC 51](#_Toc48604390)

[2.2.3 Технологии объектно-реляционных отображений (ORM) 52](#_Toc48604391)

[**Библиографический ﻿список** 54](#_Toc48604392)

# **Введение**

Учебно-методическое пособие посвящено основам проектирования клиент-серверных приложений. Разработано в рамках дисциплины «Проектирование клиент-серверных приложений» для студентов-бакалавров направления 09.03.04 «Программная инженерия».

Цель курса:

— ознакомление студентов с основными технологиями, необходимыми для создания клиент-серверных приложений;

— подготовка теоретической базы для выполнения практических работ.

В настоящее время интернет является неотъемлемой частью жизни людей, без которой существование уже и не представляется возможным. Но для взаимодействия через интернет нужны приложения, позволяющие это осуществить, то есть клиент-серверные приложения. Клиент-серверные приложения можно встретить на любом современном устройстве: телефоне, планшете, компьютере. Такие приложения включают клиентскую часть: браузеры, чаты, «облачные хранилища», игры, и серверную часть, которая и выполняет функцию хранения и обработки информации. Другими словами, клиент-серверные приложения повсеместно применяются и для приложений уровня рабочей группы, уровня предприятия и т.д. Тому есть несколько причин:

1) *отсутствие фазы развертывания приложения*. Предположим, у нас есть 100 потенциальных участников некой информационной системы. В случае с *классическим* интерфейсом нам нужно было бы установить клиента на 100 рабочих мест. В случае с клиент-серверным приложением достаточно развернуть его в одной точке — на сервере. Клиенты при этом получают возможность работы с приложением, просто набрав в браузере определенный адрес;

2) *кросс-платформенность*. Из указанных 100 человек некоторые вполне могли оказаться дизайнерами, для которых *компьютер* — это только название бренда, например, Apple, о существовании PC они могут и вообще не знать. В случае *классического* оформления приложения пришлось бы иметь две версии: одну для PC/Windows, другую — для Apple/Mac OS X;

3) *отсутствие версионности*. Web-организация взаимодействия с пользователями допускает просто замену приложения на сервере, после чего следующий пришедший запрос будет обработан уже новой версией приложения. Учитывая, что HTTP как протокол не поддерживает состояния сессии, описанное выше решение по замене версий вполне работоспособно и очень удобно в эксплуатации. Представим себе следующую ситуацию: на крупном предприятии установлена учетная система (ERP, Enterprise Resource Planning). Предприятие существует не само по себе, а в рамках государства и действующего законодательства, и обязано выполнять требование последнего. При достаточно кардинальном изменении — налогообложения, порядка расчета себестоимости или чего-нибудь еще — у разработчиков остается только один сценарий: выпуск новой версии программного обеспечения, и она должна быть установлена у всех работающих с данной системой сотрудников. Подобная ситуация случается на практике достаточно редко, тем не менее вызывает чуть ли не временный паралич работы предприятия: смена серверного программного обеспечения; необходимость настройки каждого клиентского приложения;

4) *отсутствие хранения каких-либо данных на клиентской части*. В ходе своей работы с системой пользователь может взаимодействовать с информацией разного рода: с документами, таблицами, графиками и т. д. Но вся информация обычно и хранится и обрабатывается на серверной стороне. Из этого факта вытекают два следствия:

а) пользователь не обязательно должен работать с одного и того же рабочего места. В *классическом* варианте, если пользователь пересаживается за соседний компьютер, то для него меняется все: становятся другими документы, таблицы и т.д. Более того, цвета и внешний вид окон так же могут оказаться разными. В случае с клиент-серверным приложением пользователь получает доступ к тем же самым данным и с другого рабочего места, и с другого устройства. Например, с телефона или планшета;

б) в случае поломки компьютера пользователя вопрос восстановления занимает минимум времени, никакие данные при этом не будут потеряны. Конечно же, сервер — это тоже компьютер и он тоже может ломаться. Но обеспечить (и аппаратно, и программно) резервирование на одном сервере значительно проще и дешевле, чем сделать то же самое на ста рабочих станциях. Стоимость рабочей станции при таком подходе уменьшается, на ней находится минимум необходимого программного обеспечения, необходимо только устройство с операционной системой и браузером, и он сможет продолжать свою работу. Другими словами, разработанное приложение становится значительно доступнее и надежнее;

5) современные средства позволяют разработать пользовательский интерфейс для клиент-серверных приложений, не уступающий *классическому* подходу ни в красоте исполнения, ни в функциональности, ни в эргономике.

Можно также отметить и недостатки клиент-серверных приложений. За указанные выше преимущества приходится платить спецификой разработки: как большим объемом, так и появлением на стадии разработки специфичных для web-фаз дизайна и верстки. При разработке *классического* приложения внутри операционной системы эти две фазы практически полностью отсутствуют: верстка не нужна совсем, а дизайн обычно используется тот, который предоставляется самой операционной системой. Кроме того, в классическом подходе достаточно владеть каким-то одним языком программирования. Например, C++. В случае web-подхода, как правило, одного языка для реализации серверной и клиентской части недостаточно. Более того, кроме языков программирования как таковых при разработке web-приложения используются и специфичные технологии: язык разметки HTML, каскадные стили CSS и т. д.

Есть еще один фактор, который нельзя обойти вниманием при обсуждении web-приложения, — это серверное окружение. Обычно для работы web-приложения нужно некое внешнее *обрамление*: есть один или несколько HTTP-серверов, сам контейнер, в котором выполняется приложение, сервер базы данных и т.д.

Таким образом, разработка web-ориентированного приложения выглядит более сложным процессом по сравнению с классическим вариантом, то есть приложением, выполняющимся под управлением операционной системы.

Выбор того или другого подхода зачастую рассматривается с экономической точки зрения. Например, если планируемое приложение имеет десяток возможных потребителей, то, скорее всего, будет принято решение разрабатывать его по *классическому* пути. При большом количестве возможных участников системы снижение эксплуатационных расходов будет настолько существенным, что есть смысл увеличить затраты этапа разработки, то есть разрабатывать приложения с помощью web-технологий.

Таким образом, web-подход к разработке приложений приобретает в последнее время все большую популярность.

Цель курса «Разработка клиент-серверных приложений», а также цикла практических работ — познакомиться с общими принципами построения клиент-серверных приложений, на практических примерах показать входящие в них компоненты и взаимодействие этих компонентов между собой.

Для изучения данного материала у читателя уже должны быть сформированы знания в области объектно-ориентированного программирования, без этих знаний понимание читателем курса может быть затруднительным и неэффективным.

В данном пособии читатель сможет познакомиться с теоретической частью клиент-серверных приложений, его аспектами и возможными вариантами работы приложений в интернете.

# **﻿1 ﻿Основы ﻿проектирования ﻿клиент-серверных приложений**

## 1.1 Основные понятия клиент-серверных приложений

В данном разделе рассмотрены основные определения, связанные с клиент-серверными приложениями.

### 1.1.1 Интернет и его особенности

Интернет (Interconnected Networks) — объединенные сети, глобальная телекоммуникационная сеть информационных и вычислительных ресурсов. В разгар холодной войны, 4 октября 1957 года, СССР запустил первый искусственный спутник Земли, тем самым получив преимущество в космосе. В США решили, что деньги, отпущенные Пентагоном на научные исследования, тратятся впустую, поэтому было принято решение создать единую научную организацию под покровительством Министерства обороны — ARPA (Advanced Research Projects Agency). Один из разрабатываемых передовых проектов — это распределенная децентрализованная вычислительная сеть, способная пережить даже ядерную войну. Существующие в то время телефонные сети не обеспечивали достаточной надежности: выход из строя одного крупного узла мог разделить сеть на изолированные участки. В декабре 1969 года была создана экспериментальная сеть, построенная на принципах цифровой коммутации пакетов и соединяющая 4 узла:

— калифорнийский университет в Лос-Анджелесе;

— калифорнийский университет в Санта-Барбаре;

— исследовательский университет Стенфорда;

— университет штата Юта.

Это событие считается рождением современного интернета.

Особенности интернета:

— интернет не имеет собственника, является достоянием всего человечества;

— интернет нельзя выключить целиком;

— интернет может связать каждый компьютер с любым другим;

— чтобы из множества подключенных к интернету устройств выбрать нужный, существуют две составляющих части: это *адреса* и *имена участников сети*.

#### 1.1.1.1 Адрес в интернете

Для идентификации компьютера в интернете используются *адреса*. В настоящее время работают одновременно две версии адресации:

— IP v4, представляющая собой четырехбайтное число (32 бита). Для удобства работы цифры адреса разделяют между собой точкой. Например, адрес локальной машины — это 127.0.0.1.

— IP v6, 128 бит. Восемь групп по 4 шестнадцатеричных цифры, разделенных двоеточием. Пример адреса:

2001:0db8:11a3:09d7:1f34:8a2e:07a0:765d

Нули могут заменяться двумя двоеточиями. Тот же адрес локальной машины можно написать как ::1. Используется одновременно с IP v4 за счет так называемого «отображения адресов»:: FFFF: xx.xx.xx.xx дает IP v4 адрес, то есть младшие 32 бита при этом равны ip v4 адресу. Такой подход дает возможность с сетей IP v6 обращаться к тем хостам, которые поддерживают только IP v4.

В обоих стандартах существуют особые группы адресов. Сюда относятся так называемые не маршрутизируемые адреса, предназначенные для организации локальных сетей и служебного взаимодействия самих маршрутизаторов между собой внутри сетей провайдера — это мультикастовые адреса, предназначенные для широковещательных сообщений, когда одно сообщение передается одновременно нескольким получателям (но не одному и не всем участникам сети); адреса, уникальные только в пределах одного компьютера (локальные адреса).

#### 1.1.1.2 Имя в интернете

Другой важной составляющей интернета является служба имен (DNS, Domain Name System), которая предназначена для представления адресов компьютеров в более «человеческой», универсальной форме. Служба DNS преобразует полученное имя хоста (строку) в IP-адрес устройства. Ключевым элементом отсчета является точка («.» так и называется — домен точка, или корневой домен, или домен нулевого уровня). Домен точка разделен на зоны (.com., .org., .net., .ru. и так далее). В связи с тем, что домен точка присутствует в адресе всегда, он опускается в записи адреса (последняя точка не указывается, вместо urfu.ru. пишут urfu.ru).

Существует несколько утилит, позволяющих вручную провести распознавание адреса. Направлены эти утилиты прежде всего на то, чтобы диагностировать работоспособность службы DNS на данном участке сети. В частности, можно отметить команды host и nslookup:

sally ~ # host urfu.ru urfu.ru has address 93.88.190.5 urfu.ru mail is handled by 100 relay1.urfu.ru. urfu.ru mail is handled by 200 relay2.urfu.ru.

У DNS существует как прямое направление распознавания (по имени найти IP-адрес), так и обратное (по адресу найти имя). В общем случае полученные значения могут не совпадать между собой, потому что один физический IP-адрес может иметь несколько имен. В приведенном выше примере адрес urfu.ru и его IP-адрес не совпадают между собой в прямом и обратном направлении: sally ~ # host 93.88.190.5

5.190.88.93.in-addr.arpa domain name pointer ustu.ru.

Для идентификации ресурса в интернете существуют определенные стандарты. Чтобы получить конкретный ресурс (документ, изображение, почтовое сообщение, вызываемую процедуру и так далее), кроме собственно адреса хоста, на котором этот ресурс расположен, нужно указать еще ряд параметров. В общем случае строка идентификации ресурса согласно спецификации URL (Uniform Resource Locator) выглядит следующим образом:

имя\_службы://имя\_хоста.имя\_домена.зона: порт/имя\_ресурса

Кроме того, указанная строка при необходимости может быть дополнена именем входа/паролем, а также дополнительными произвольными параметрами, передаваемыми в запросе ресурса (например, параметрами для вызова RPC-процедуры).

Вот как выглядит адрес ресурса, предоставляющего сведения о погоде в Москве:

https://www.gismeteo.ru/weather-moscow-4368/#wdaily1

Забегая вперед, отметим, что довольно часто при разработке клиент-серверных приложений появляется необходимость обмана службы DNS путем присвоения некоторым соседним компьютерам фиктивных имен, на самом деле DNS не распознаваемых. Либо второй вариант — присвоить одному компьютеру несколько имен. Делается это при помощи файла hosts.

В MS Windows файл hosts расположен в каталоге С:\Windows\System32\drivers\etc\, в \*nix системах — в каталоге/etc.

Например, мы хотим одновременно разрабатывать на одном компьютере два web-приложения. Одно из них представляет собой библиотеку книг с указанием автора и краткой аннотацией и называется library. Другое из них посвящено учету персональных финансов и называется purse. Когда приложения будут готовы, в файлы DNS будут внесены соответствующие изменения, и имена станут доступны всем пользователям интернета, но на этапе разработки это не обязательно. Достаточно внести следующую строчку в упомянутый файл hosts:

127.0.0.1 localhost purse library

Компьютер, за клавиатурой которого непосредственно производится разработка обоих указанных приложений (то есть «локальный» компьютер) будет откликаться на два этих имени.

Такой подход (обман) не позволит подключиться любому пользователю интернета к вашим приложениям. Но он дает возможность организовать на одном компьютере два независимых виртуальных хоста и обращаться к ним из строки браузера, набирая в ней (на локальном компьютере): http://purse или http://library, и получая ответ от разрабатываемых приложений.

#### 1.1.1.3 Службы (сервисы)

Службы (сервисы) — это информационные системы, разработанные для предоставления конкретных информационных услуг в сети интернет. Они включают в себя наборы программ и протоколов прикладного уровня, обеспечивающих пользователей сети возможностью выполнять работу с распределенными информационными ресурсами. Каждый сервис базируется на своем протоколе (или семействе протоколов), который позволяет клиентам и сервисам общаться между собой. Можно привести несколько примеров сервисов, базирующихся на соответствующих протоколах:

— служба DNS (BIND, 53 порт). Базис интернета. Неработоспособность службы DNS приведет к неработоспособности всех остальных служб, поэтому существует ряд технических и организационных решений, направленных на обеспечение круглосуточной и бесперебойной работы этой службы в сети;

— служба электронной почты (SMTP, POP-3, IMAP);

— обмен моментальными сообщениями (WhatsUpp, Telegram, Skype);

— передача файлов между компьютерами (FTP, SFTP);

— управление удаленными серверами, выполнение на них команд удаленным способом, копирование файлов (SSH, RSH, FISH);

— управление сетью (SNMP);

— служба доступа к каталогам справочной информации (LDAP).

Наиболее известным сервисом в интернете является web-сервис (WWW, World Wide Web, «Всемирная паутина»), предоставляющий пользователям доступ к документам с использованием протокола HTTP. Очень часто между понятиями *интернет* и *web* ошибочно ставят знак равенства. Между тем, протокол HTTP, лежащий в основе службы web, безусловно, самый распространенный, но все же один из многих протоколов передачи данных, объединенных между собой названием *интернет*.

#### 1.1.1.4 Сокета

Сокета представляет собой абстракцию, программный объект внутри операционной системы, с помощью которого происходит соединение клиентской и серверной части. При этом клиент и сервер могут находиться как внутри одного и того же компьютера, так и быть на разных компьютерах.

Представим себе такую ситуацию: вам звонят по телефону. При этом и вы, и звонящий вам абонент берете в руки телефонные трубки. Вас не интересуют подробности работы телефонной сети. Вы не знаете, какими базовыми станциями обслуживается ваш разговор, по каким каналам связи идет сигнал, где и какое сработало коммутационное оборудование и так далее. Зато вы знаете, что, если набрать номер, через какое-то время установится соединение, и можно будет говорить. Причем делать это нужно по очереди. То есть для общения вы должны соблюдать некоторый протокол, последовательно обращаясь к некоторым функциям трубки (набрать номер, дождаться ответа абонента, говорить). Можно сказать, что телефонная сеть для вас представлена трубкой. То же самое справедливо и для сокеты: сразу за ней начинается сеть, но подробности работы самой этой сети приложению знать не обязательно, достаточно вызывать некоторые функции объекта сокеты в определенном порядке (то есть с соблюдением протокола).

Сокета — объект универсальный, единый по своей сути для клиентской и серверной стороны. После ее создания при помощи вызова соответствующих методов сокета может стать как клиентской, так и серверной. Если воспользоваться аллегорией, то серверная сокета — это электрическая розетка в стене. У этой розетки есть замечательное свойство: она всегда является свободной. Как только к ней подключается клиент (вилка), сервер сразу же порождает новый процесс, обслуживающий данное соединение. Розетка с включенной в нее вилкой отходит в сторону, где происходит непосредственно обмен данным между конкретным клиентом и сервером. А исходная розетка по-прежнему остается свободной, доступной для следующих соединений. Каждый подключившийся клиент занимает определенные ресурсы операционной системы, поэтому при создании серверной сокеты указывается максимально допустимое число клиентских соединений.

Для работы с созданной сокетой, точно так же, как и с любым другим устройством, операционная система использует дескриптор — беззнаковое целое число, идентифицирующее ресурс внутри операционной системы. После получения дескриптора в сокету можно писать данные, читать из нее, назначать на нее некие обработчики событий (сигналов) и так далее. Существует великое множество библиотек для работы с сокетами, которые предоставляют реализацию функций либо вообще готовых классов клиентской и серверной сокеты. Эти библиотеки представляют собой переход на более высокий уровень работы, предназначены для облегчения труда программистов. Действительно, особого смысла каждый раз реализовывать одни и те же низкоуровневые процедуры нет. Однако на нижнем уровне обращение с сокетой в любой из библиотек все равно происходит с использованием именно дескриптора.

Сокета как программный объект характеризуется тремя параметрами: доменом (областью, семейством протоколов, которые могут быть использованы для данной сокеты), типом и портом.

Наиболее распространенными являются следующие домены:

— AF\_UNIX (AF\_LOCAL) — для организации обмена в рамках одного и того же компьютера;

— AF\_INET, AF\_INET6 — по протоколу IP v4/IP v6 соответственно;

— AF\_NETLINK — для взаимодействия между пространством пользовательских программ и ядром операционной системы. Среди типов сокет наиболее распространены три:

1) SOCK\_STREAM — создаваемая сокета предназначена для организации последовательного потока. При этом между клиентом и сервером устанавливается виртуальное соединение, ошибки в работе которого также обрабатываются сокетой. Соответственно, если мы создадим сокету с доменом AF\_ INET и типом SOCK\_STREAM, такая сокета будет обрабатывать TCP/IP соединение;

2) SOCK\_DGRAM — сокета, предназначенная для передачи дейтаграмм. Дейтаграммы (датаграммы, datagram) — это пакеты информации, которые передаются по сети без установления соединения. Они имеют адрес отправителя и получателя, но факт их доставки не гарантируется. Точно так же не гарантируется и порядок прихода дейтаграмм: далеко не факт, что получатель примет сообщения в том же самом порядке, в каком они были переданы. Основной особенностью работы с дейтаграммами является отсутствие временных задержек, вызванных скоростью работы сети. Являются основой протокола UDP (User Datagram Protocol);

3) SOCK\_RAW — так называемая сырая сокета. Данный тип сокеты используется для низкоуровневого программирования, а также при необходимости дополнения существующих протоколов (например, добавления в сообщение собственных заголовков).

Порт сокеты — это беззнаковое целое двухбайтное число, которое определяет службу сервера. Сочетание порт–протокол является уникальным на данном компьютере в том смысле, что, если данный порт и данный протокол уже прослушиваются каким-либо процессом, другой процесс не сможет повторно создать такую же сокету. Тот же порт, но с другим протоколом при этом захватить можно. Порты диапазона 0–1023 являются привилегированными, то есть для создания сокеты, прослушивающей данный порт, необходимы особые привилегии. Теоретически любой порт свыше 1024 может быть захвачен первым запросившим его процессом. На самом деле число официально зарегистрированных портов гораздо больше, чем 1024. В качестве примера можно назвать сочетание 3306/tcp, которое используется сервером MySQL по умолчанию. Общепринятые присвоения портов не являются официальным стандартом, и любой созданный пользователем процесс имеет полное право захватить порт 3306/tcp. И иногда программисты намеренно этим пользуются с целью ввести в заблуждение возможных злоумышленников. Однако, если у вас нет на то каких-то особых причин, не стоит использовать в своих программах зарегистрированные пары порт–служба.

После создания сокеты (иными словами, получения от операционной системы дескриптора сокеты) на нее могут быть установлены модификаторы — такие же, как и для любого символьного устройства в операционной системе. Например, сокету можно сделать неблокирующей. При этом, если из сокеты на данный момент нечего читать, то оператор чтения сокеты вернет нулевое количество прочитанных байт (а не приостановит работу программы, как это произойдет в случае с блокирующей сокетой).

#### 1.1.1.5 Протокол HTTP. Виды запросов

HTTP (Hyper Text Transfer Protocol) – протокол передачи гипертекста, то есть такого текста, когда вместе с содержимым передается информация по его отображению. Изначально HTTP предназначался для передачи HTML-содержимого. Однако в настоящее время протокол HTTP используется для передачи любой информации произвольного содержания, оставаясь при этом текстовым. Термин «текстовый» означает, что в ходе работы могут передаваться только отображаемые на экране символы. Например, в принимаемой и передаваемой информации не может быть символа «0». Разумеется, что понятие «любая информация» подразумевает, что каждый передаваемый байт может принимать все допустимые значения, то есть 0–255 включительно. Такая информация еще называется бинарной. Передача бинарной информации с помощью текстового протокола осуществляется за счет специального кодирования, когда исходная информация на момент передачи видоизменяется таким образом, что содержит только текстовые символы. На клиенте после получения сообщения осуществляется обратное преобразование. С помощью такого механизма посредством HTTP могут передаваться, например, двумерные или трехмерные изображения.

Кроме того, HTTP широко используется как транспортный протокол для передачи между разными программами объектов (экземпляров классов). Информация о структуре объекта (именах полей, их модификаторах, значениях и т.д.) преобразуется в текстовый вид (например, в XML или JSON). В текстовом виде передается по сети и на клиенте осуществляется обратное преобразование из текста в программный объект. Одним из существенных преимуществ подобного метода передачи является то обстоятельство, что клиент и сервер могут быть написаны на разных языках программирования, выполняться на различных архитектурах и т.д. Таким образом, с помощью протокола HTTP связываются между собой гетерогенные приложения в логически единое целое (рисунок 1.1).

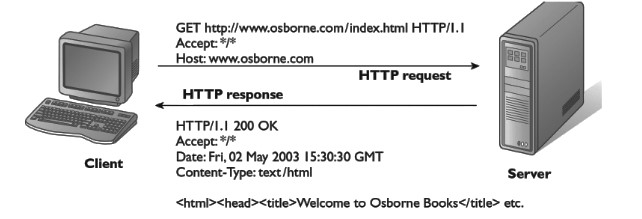


Рисунок 1.1 – Пример общения клиента и сервера по HTTP-протоколу

Основой HTTP является технология клиент-сервер, то есть всегда выделяются два участника обмена: клиент, который делает запрос (HTTP Request), и сервер, который отвечает клиенту на данный запрос (передавая по сети пакет HTTP Response). На этом один шаг протокола заканчивается, и данный цикл может повторяться сколь угодно большое число раз. Особенностью HTTP является тот факт, что сам по себе протокол не поддерживает информацию о сессии, не обязан что-либо знать об истории. Иными словами, протокол не предусматривает сохранения состояния. Есть запрос, и есть ответ на него. Причем, как правило, сразу после ответа сервер разорвет TCP-соединение с клиентом с целью более рачительного использования собственных ресурсов. При необходимости клиенты, использующие HTTP, могут самостоятельно сохранять информацию об истории данного сеанса.

Для этих целей существуют два механизма:

1) cookies (куки) – небольшие пакеты данных, хранящиеся на стороне клиента и передающиеся вместе с запросом. Как правило, cookie имеет два атрибута, определяющие его сохранение клиентом, то есть возможность сохранения информации на диске. В случае, если cookie разрешается хранить информацию у клиента на диске, указывается время жизни данного cookie на клиентской стороне, после чего он будет считаться устаревшим, не актуальным. В частности, в качестве cookie может выступать уникальный идентификатор сессии на стороне сервера;

2) HttpSessions (сессии) – на стороне сервера. Сессия представляет собой программно-доступный объект, который формируется сервером и содержит в себе информацию о работе с данным клиентом. Объект сессии уникален для сеанса работы с каждым клиентом. Обычно сессия имеет идентификатор, который сохраняется в cookies на клиенте. Механизм сессий скрывает подробности реализации, разработчику нет необходимости отслеживать, что из информации запоминается на клиентской части, а что — на серверной. Достаточно просто получить объект сессии и думать о нем, что он однозначно определяет сеанс данного клиента.

Реализация сервера может хранить информацию о заголовках последних запросов, поступивших с данного IP-адреса. Сам протокол не осведомлен о предыдущих запросах и ответах, внутренняя поддержка состояния в нем не предусмотрена. Текущей версией протокола HTTP/1.1 предусмотрен режим постоянного соединения, то есть установленное TCP/IP соединение может оставаться открытым и после отправки ответа на поступивший запрос. Ключевое слово здесь «может»: при разработке собственных клиент-серверных приложений правильнее будет полагать, что HTTP работает как одна-единственная изолированная от контекста пара запрос-ответ.

Вне зависимости от вида – запрос это или ответ – пакеты в HTTP имеют одинаковую структуру и состоят из следующих параметров:

— стартовой строки или строки запроса/ответа. Содержание стартовой строки однозначно идентифицирует тип сообщения (тип пакета). Содержание стартовой строки отличается для запросов и ответов;

— заголовков сообщения. Разделенные двоеточием пары «имя: значение». Характеризуют тело сообщения (какой сервер используется; когда последний раз менялся объект; что конкретно будет передаваться; в какой форме кодироваться и так далее). Существование заголовков в том числе позволяет передавать с помощью HTTP бинарную (двоичную, то есть любую, не обязательно текстовую) информацию, хотя сам протокол является текстовым. На использовании заголовков базируются и расширения HTTP: разработчик серверного программного обеспечения вправе добавить свои собственные заголовки. При этом сохраняется совместимость с существующими клиентами, которые будут просто игнорировать незнакомые заголовки;

— тела сообщения. Тела сообщения может и не быть в запросе в том случае, если стартовая строка и заголовки однозначно идентифицируют запрашиваемый ресурс.

Заголовки от тела отделяются пустой строкой. То есть такой строкой, которая не содержит других символов, кроме возврата каретки и перевода строки, <CR><LF>, «\r\n», коды символов — 13 и 10 в десятичной системе. Довольно часто встречаются решения, ограничивающее данное правило только до одного символа — возврата каретки («\r»).

**1.1.1.5.1 Структура ﻿запроса ﻿(Request)**

Стартовая строка запроса выглядит следующим образом:

Метод URI HTTP/Версия

Метод — это название запроса, одно слово заглавными буквами. Наибольшее распространение имеют следующие методы:

GET — запросить содержимое указанного ресурса. Клиент может передавать параметры указанному ресурсу, перечисляя их после символа «?». Пример стартовой строки метода GET с передачей параметров на сервер:

GET/some\_resource?param1=value&param2=value2 HTTP/1.1

Метод GET является идемпотентным, то есть многократный запрос GET с одними и теми же параметрами должен приводить к одним и тем же результатам.

HEAD — то же самое, что и GET, но само содержимое ресурса при этом сервером не передается, передаются только заголовки. Метод позволяет узнать, существует ли запрашиваемый ресурс на сервере. Если имеется, то не менялось ли его содержимое со времени последнего запроса.

POST — применяется для передачи пользовательских данных (параметров) указанному ресурсу. При этом сами передаваемые параметры включаются в тело запроса. При помощи метода POST можно создать ресурс (например, загрузить файл на сервер). В этом случае сервер выдаст ответ 210 (Created) и в заголовке Location будет указан URI созданного ресурса. Метод POST идемпотентным не является, то есть многократное повторение POST с теми же параметрами может приводить к разным результатам.

PUT — загрузка содержимого запроса на указанный ресурс.

DELETE — удалить указанный ресурс.

URI (Uniform Resource Identifier) — путь к запрашиваемому ресурсу (документ, изображение, файл, службу, ящик электронной почты и т. д.).

Версия — пара разделенной точкой цифр.

**1.1.1.5.2 Структура ответа ﻿(Response)**

Стартовая строка ответа сервера имеет следующий формат:

HTTP/Версия КодСостояния Пояснение

Версия — две цифры, разделенные точкой.

КодСостояния — три цифры, значение которых определяет результат выполнения запроса и дальнейшее поведение клиента. Первая цифра кода состояния определяет его класс, группу, к которой принадлежит данный код состояния. В настоящее время выделяют следующие группы:

— 1xx — информационное сообщение;

— 2xx — успешно выполненный сервером запрос;

— 3xx — перенаправление. Сообщает клиенту, что нужно сделать еще один запрос, как правило, по другому URI. Адрес, по которому клиент должен сделать запрос, как правило, указывается в заголовке Location;

— 4xx — были допущены ошибки со стороны клиента;

— 5xx — возникли ошибки на сервере.

При возникновении ошибок, как правило, тело сообщения содержит гипертекстовую информацию, поясняющую возникшую ошибку.

Пояснение — текстовое короткое пояснение к коду ответа. Ни на что не влияет и не является обязательным. Большая часть библиотек работы с HTTP имеет собственные средства определения «пояснения» по полученному коду, в том числе и локализованного пояснения.

Пример HTTPresponse: HTTP/1.1 200 OK

Date: Wed, 11 Feb 2013 11:20:59 GMT

Server: Apache

X-Powered-By: PHP/5.2.4–2ubuntu5wm1

Last-Modified: Wed, 11 Feb 2013 11:20:59 GMT

Content-Language: ru

Content-Type: text/html; charset=utf-8

Content-Length: 1234

Connection: close

(пустаястрока)

(далее следует запрошенная страница в HTML)

### 1.1.2 Клиент-серверные приложения

Стоит отметить, что термины *клиент-серверные приложения* и *web-приложения* не являются синонимами. Как уже говорилось выше, клиент-серверное приложение не обязано базироваться именно на web-службе и использовать для межкомпонентного обмена протокол HTTP. Однако, поскольку большую часть клиент-серверных приложений составляют именно такие приложения, разница между терминами практически сходит на нет.

#### 1.1.2.1 Web-приложения

Web-приложение построено, как минимум, по двухуровневой архитектуре (то есть по архитектуре клиент-сервер). При этом в качестве клиентской программы используется web-браузер, а обмен с серверной частью происходит с использованием протоколов HTTP/HTTPS.

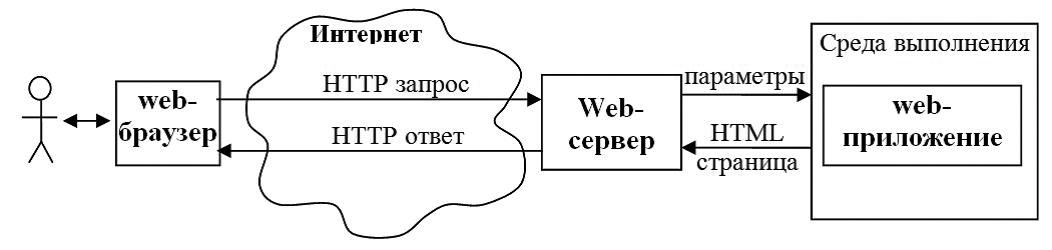


Рисунок 1.2 – Общая схема взаимодействия пользователя   
с Web-приложением

Как видно из рисунка 1.2, web-сервер (блок, реализующий обмен с клиентом по протоколу HTTP) не является единственной составляющей приложения. Он транслирует методы и их параметры в некую среду, которая программным путем формирует HTML-страницу. Такие страницы называются динамическими, потому что их содержание меняется во времени, может зависеть от параметров, от предыдущих шагов клиента в рамках данной сессии. Среда выполнения может быть различной, более подробно вопрос о способах формирования динамических страниц будет рассмотрен ниже.

Основные причины широкого распространения именно web-приложений обусловлены их достоинствами, а именно:

— доступностью (пользователю не нужно что-либо ставить на компьютер в качестве клиентского программного обеспечения, достаточно просто набрать нужный адрес в браузере);

— отсутствием версионности (если разработчик изменил код приложения, клиентам не нужно сообщать, доступна ли новая версия программы. Следующий после изменения кода запрос даст пользователю уже новое содержание).

#### 1.1.2.2 Web-сервисы

Web-приложение направлено на работу с пользователем и имеет пользовательский интерфейс. В противоположность этому, web-сервис работает либо с другими web-сервисами, либо с web-приложениями. Обмен при этом происходит точно так же, как и в случае с приложениями, то есть по схеме запрос-ответ. В качестве клиента может выступать любая программа, которая правильно сформирует HTTP-запрос и расшифрует полученный HTTP-ответ. Однако для общения стандартной версии HTML может оказаться недостаточно, поэтому используются его расширения: JSON, XML-RPC, SOAP, REST и так далее. Общая схема работы web-сервиса представлена на рисунке 1.3.

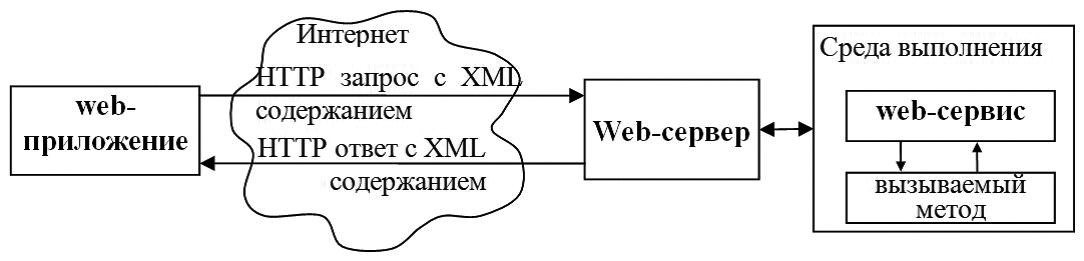


Рисунок 1.3 – Общая схема взаимодействия пользователя с web-сервисом

Для описания сервиса существует специальный язык, называемый WSDL (Web Service Definition Language). При помощи WSDL можно запросить у web-сервиса сведения о существующих методах и необходимых параметрах, то есть получить полную описательную информацию о предоставляемом сервисе. Кроме собственно информативной составляющей, WSDL несет и другую нагрузку: на его основе строятся различные средства автоматизированного проектирования web-сервисов — программные средства, позволяющие из WSDL файла создавать скелеты классов и наоборот.

В современном интернете существует множество web-сервисов. В качестве примера можно указать Яндекс.XML. Существует всем известная поисковая система Яндекс. Однако для ее использования не обязательно заходить на http://yandex.ru. Вы можете использовать ее в любом созданном приложении (и не обязательно web-ориентированном). Сервис Яндекс.XML позволяет обратиться с запросом к самой поисковой системе, получить результат выполнения этого запроса в виде XML и использовать полученный результат в своем приложении. Разумеется, при этом существует вопрос лицензионности: придется пройти ряд предписанных Яндексом шагов и обязательно указать в своем приложении ссылку, на основе чего сформирован данный ответ. Но суть остается прежней: предлагается некий сервис, данные из которого могут использоваться по своему усмотрению.

#### 1.1.2.3 Особенности проектирования

Клиент-серверные приложения, как правило, доступны всем участникам сети Интернет, даже если целью является разработка простой узкоспециализированной информационной системы. Предположим, разрабатывается информационная система предприятия. Одной из поставленных задач будет задача разделения прав пользователей: кто и что сможет видеть внутри этой системы. Независимо от того, что будет происходить в клиентской части, серверная часть, точка входа в разрабатываемую систему, будет расположена в интернете, соответственно, доступна всей многомиллиардной аудитории. В отличие от классических пользовательских программ большую часть клиент-серверных приложений не нужно устанавливать, никто не спросит разрешения на попытку поработать с системой.

Соответственно, при проектировании клиент-серверных приложений приходится уделять повышенное внимание вопросам безопасности работы. При разработке системы безопасности основным принципом является разумная достаточность предпринимаемых мер. Нет никакого смысла проводить сканирование сетчатки глаза, снимать отпечатки пальцев, делать голосовой анализ, вытаскивать из процессора его серийный номер и из сетевой карты ее mac-адрес — и все это только для того, чтобы отобразить пользователю на экране сегодняшние новости.

Другой не менее важной особенностью является непрогнозируемая нагрузка. Сегодня приложение имеет 10 одновременно работающих клиентов в максимуме, а завтра их станет миллион. Соответственно, при разработке приложения ориентируетесь на 10 клиентов, это расчетная нагрузка. Нет никакого смысла сразу же делать приложение из расчета на миллион, потому что нагрузочная способность — это, в конечном счете, деньги и время, что и будет успешно потрачено, а вот миллион клиентов у приложения может и не появиться в силу целого ряда причин. Однако необходимо предусмотреть возможность масштабирования системы таким образом, чтобы удержать лавинообразно нарастающую нагрузку.

Третья рассматриваемая особенность — это круглосуточная работоспособность приложения. Если мы говорим о локальной сети предприятия, то существует нерабочее время для сотрудников этого предприятия, выходные, праздники. Иными словами, у администраторов всегда есть возможность что-то менять, не нарушая работоспособности системы: переставить программное обеспечение, провести сервисные процедуры с базой данных и даже целиком поменять сервер на более мощный. При должной подготовке даже серьезные изменения в системе не скажутся на пользователях, поскольку к началу следующего рабочего дня их будет ждать полностью способная выполнять свои функции система. Совсем другая картина получается с клиент-серверными приложениями, пользователи которых находятся в разных часовых поясах и имеют полное право затребовать функции разработанного сервиса в любое время любого дня. А между тем задачи администрирования и сервисного обслуживания никуда не исчезли, их по-прежнему необходимо выполнять. И одно из возможных решений — это правильная архитектура разрабатываемого приложения, допускающая временную неработоспособность одной или нескольких компонент без нарушения функционала системы в целом. Это возможно.

#### 1.1.2.4 Особенности пользовательского интерфейса

Пользовательский интерфейс web-приложений тоже имеет свои особенности, обусловленные большой (возможной) аудиторией. Web-приложения должны быть простыми, насколько это вообще возможно, то есть интуитивно понятными для большинства пользователей. Предположим, разрабатывается интерфейс приложения, предназначенного для работы в рамках некоторого предприятия. При этом всегда есть возможность устроить для будущих пользователей системы обучающие курсы, рассказать им, что есть на экране и что разработчики хотели этим сказать. В случае с web-приложением такой возможности не будет. Пользователь либо догадается сам, что разработчики имели в виду, либо просто уйдет.

Поскольку у пользователя есть варианты его поведения внутри созданного web-приложения (варианты пути пользователя), нужно выделять желаемый вариант поведения пользователя: цветом, размером, шрифтом. Если довести ситуацию до шутливого абсурда, то получим «идеальный web-интерфейс» — это белый экран и на нем огромная красная кнопка «Оплатить».

Обычно разработчики приложений придерживаются неких правил хорошего тона. Эти правила не являются уникальными именно для интернета, они обусловлены общечеловеческой этикой. На сайт в первый раз попал человек. Возможно, случайно, возможно, по рекомендации своих знакомых. Ситуация точно такая же по сути, но без привязки к сетям и сайтам: вы устроили грандиозный прием (или небольшую вечеринку) и зашел случайный гость. Как вы себя поведете в таком случае, учитывая, что в госте вы заинтересованы? Это же ваш посетитель и вам крайне важно, чтобы у него осталось хорошее впечатление, и он пришел сюда еще раз. А лучше бы еще и друзей с собой привел. Если посмотреть на ведущие web-приложения мира, то все они ведут себя в данной ситуации одинаково. Наверное, так же, как вы бы себя повели в жизненной ситуации с вечеринкой:

1. человека нужно встретить. Причем встретить прямо на пороге, чтобы он не смог уйти из-за того, что просто постеснялся войти внутрь. Объяснить ему, что вы ему очень рады, что он не будет здесь лишним. Попросить его чувствовать себя как дома и так далее;
2. его нужно познакомить с обстановкой и остальными гостями. Вкратце обрисовать ему, куда он попал, что здесь происходит. То есть, возвращаясь к ситуации с интернетом, коротенько пояснить суть приложения;
3. человеку нужно создать благоприятную обстановку. Следует сделать так, чтобы в рамках приложения пользователю стало хорошо и он не пожалел, что сюда зашел, почувствовав свою прямую выгоду;
4. заинтересовать человека бонусами. Пришедшему пользователю нужно дать понять, что если он приведет сюда друзей, то ему станет еще лучше, выгода будет еще больше.

Другой особенностью является отсутствие стандартов на оформление визуальных элементов интерфейса. Если мы возьмем элемент кнопка, или поле ввода, или меню, или любой другой визуальный элемент, то в рамках операционной системы этот элемент выглядит одинаково и ведет себя одинаково в любой программе. Соответственно, пользователь привыкает к расположению, внешнему виду и поведению визуальных элементов данной операционной системы. Меню в любой программе расположены вверху окна (Windows) или в специальной строке меню (MacOsX). Кнопка всегда имеет цвет, определенный текущей палитрой пользователя. Внешний вид кнопки одинаков в разных программах. И так далее.

Описанное выше, не является справедливым для web-приложений в полной мере. Стандартами языка HTML предусмотрены визуальные элементы кнопка, поле ввода и так далее. Однако, как в силу сравнительной бедности этих элементов, так и в силу имеющихся развитых программных средств для их модификации каждый разработчик раскрашивает элементы и изменяет их поведение по своему усмотрению.

Можно попытаться сформулировать некоторое правило хорошего тона для разработки интерфейса web-приложения: не стоит чрезмерно модифицировать внешний вид и поведение визуальных элементов по сравнению с наиболее распространенным, имеющимся на уже существующих широко известных сайтах. Есть смысл сначала посмотреть, как та же самая функция реализована наиболее значимыми, распространенными web-приложениями. В противном случае имеется существенный риск дезориентировать пользователя. Если нарисовать круглую кнопку, пользователи могут просто не найти ее на экране, потому что ранее сформировавшаяся привычка заставляет искать глазами кнопку в форме квадрата. Конечно, вышесказанное — это только правило, а вовсе не закон, ведь вполне возможно, что нестандартный интерфейс как раз и является фишкой разрабатываемого сайта.

Приведенный перечень особенностей не является исчерпывающим, но является достаточным для демонстрации специфичности клиент-серверных приложений, их непохожести на задачи классического прикладного программирования.

# **2 Технологии построения клиент-серверных приложений**

Приложения вообще и клиент-серверные приложения в частности имеют особенность развиваться с течением времени. Изначально задумывалась только одна небольшая функция. Она была реализована, появились первые пользователи. Позже к приложению добавилась еще одна функция, следом по желанию пользователей была добавлена четвертая, пятая и так далее. И может получиться ситуация, когда плохо структурированный, плохо читаемый, плохо поддерживаемый программный код выступит в качестве снежной лавины и засыплет под собой разработчиков. Такие ситуации происходили и с крупными проектами, пример тому — один из первых появившихся браузеров, — NetscapeNavigator. Переписать уже имеющееся приложение — процесс трудоемкий и длительный, за это время конкурирующие продукты неизбежно займут ваше место на рынке. Так, за время переделывания браузер Mozilla Firefox потерял лидирующие позиции.

Чтобы на определенном этапе развития не пришлось полностью переписывать все приложение, необходимо с самого начала его разработки следовать определенным парадигмам программирования. Это подходы, методологии, которые не зависят от конкретного языка программирования и позволяют организовать внутреннюю структуру создаваемого программного продукта определенным образом, способным строить приложения из логически законченных кирпичиков, взаимодействующих друг с другом. Избежать ситуации с необходимостью переписывания определенных частей кода, скорее всего, не удастся. Но вот разграничить между собой различные сущности создаваемого приложения, четко оговорить времена и способы взаимодействия этих сущностей между собой — задача вполне реальная. И затем, при необходимости доработки, изменяется только один такой кирпичик, все остальное приложение остается в его первоначальном виде.

## 2.1 Технология создания клиентской части

Код разрабатываемого приложения может выполняться как на серверной стороне, так и на стороне клиента. Более того, прямо в тексте HTML друг за другом могут идти куски кода, часть из которых будет выполняться на стороне сервера, а часть — на стороне клиента. Применительно к web-приложениям наличие какого-то кода с выполнением на стороне клиента не является обязательным. Вполне возможно обеспечить достаточную функциональность, пользуясь в качестве клиента только браузером и ограничиваясь HTML. Все программирование при этом целиком будет возложено на серверную часть. Однако такое приложение будет выглядеть несколько архаично и вряд ли понравится пользователям.

### 2.1.1 HyperText Markup Language (HTML)

HTML (HyperText Markup Language) — это язык разметки документов. По сути, это обычный текст. Кроме собственно содержимого, он одновременно несет в себе информацию о том, как это содержимое следует отображать или обрабатывать на стороне клиента.

Любой документ в HTML состоит из элементов. Начало и конец каждого элемента обозначаются специальными пометками — тэгами, еще иногда их называют дескрипторами, в том числе таким элементом является и собственно документ, который заключается в тэги <HTML></HTML>. Как правило, тэги парные. То есть на каждый открывающий тэг в документе содержится точно такой же, но закрывающий, обозначающий конец элемента. Элементы могут быть пустыми, например, перевод строки, <BR/>. Тэг в этом случае является одновременно и открывающим, и закрывающим. Регистр букв значения не имеет, большие и маленькие литералы обрабатываются одинаково. Поскольку окончательное форматирование отображаемого документа производится браузером, игнорируются также и символы возврата каретки — перевода строки, также достаточно вольно интерпретируются символы пробела и табуляции. При необходимости браузер можно настоятельно попросить сделать именно так, как задумывалось. Либо при помощи специальных тэгов, либо при помощи так называемых сущностей (entities). Например, появление в документе сочетания &nbsp; приведет к тому, что браузер в этом месте отобразит пробел даже в том случае, если в этом, на взгляд браузера, и не было необходимости. Сущности позволяют отобразить на экране специальные символы. Например, &copy; приведет к формированию на экране знака авторского права ©, &lt; — символа меньше (<), &gt; — символа больше (>). Появление в документе двух последних символов в их обычном виде (<, >) является некорректным в связи с тем, что они используются для формирования тэгов. Вообще говоря, с помощью сущностей можно отобразить на экране любой символ, если указать его в виде &#NNNN; где NNNN — код символа в Unicode.

Элементы могут иметь свои собственные атрибуты, которые указываются в открывающем тэге:

<A HREF=”http://www.mysite.ru/hello.html”>Ссылка</A>

Появление такого тэга приведет к формированию на экране ссылки — выделенной надписи, при нажатии на которую произойдет переход по указанному в атрибуте href-адресу. Существует ряд атрибутов, которые могут быть применены к любому элементу внутри документа. Это такие атрибуты, как ID, NAME, CLASS, STYLE и так далее. Другие атрибуты могут быть применены только к элементам определенного вида. Например, указанный выше HREF применяется к атрибуту A (ссылка).

Тэги могут быть вложенными. В этом случае вложенный элемент наследует атрибуты его родителя. Пример вложенности тэгов:

<b> Этот текст будет выделен жирным шрифтом.

<i> А этот текст, кроме жирного, будет еще и наклонным шрифтом.

</i>

</b>

Поскольку спецификаций HTML, включая его расширения XHTML (Extended Нypertext Markup Language), существует несколько, то правилом хорошего тона считается первым тэгом документа указывать <! DOCTYPE точное\_указание\_используемой\_спецификации >, однозначно сообщающее браузеру, какого типа спецификация используется в документе. Обязательным это правило не является, просто в противном случае корректность отображения всех элементов документа не гарантируется.

Язык HTML дает возможность добавления комментария, для чего предусмотрен специальный тэг. Символы между <!-- и --> игнорируются браузером, не отображаются на экране. Комментарий может включать в себя другие тэги, которые в этом случае также не отображаются на экране. Вложенные комментарии недопустимы.

В общем случае документ делится на заголовок и тело. Заголовок на экране не отображается, но влияет на поведение браузера. В заголовке можно указать, разрешается ли кэшировать данную страницу, в какой кодировке ее желательно отображать, что написать в качестве заголовка окна браузера и так далее.

Ниже приведен пример HTML-документа.

<! DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"

"http://www.w3.org/TR/html4/strict.dtd">

<html>

<head>

<title>Пример страницы</title>

<meta http-equiv="Content-Type" content=»text/html; charset=utf-8»>

</head>

<body>

<h1 align="center">Пример страницы</h1>

<!— Меню —>

<div>Меню</div>

</body>

</html>

### 2.1.2 Cascading Style Sheets (CSS)

CSS (Cascading Style Sheets ) — это язык стилей, определяющий отображение HTML-документов. При помощи CSS нельзя изменить содержимое страницы, но можно полностью изменить ее внешний вид: шрифты, цвета, расположение на странице, реакцию на различные события. В принципе, применение CSS для web-страниц не является обязательным. Тех же результатов можно достичь и средствами одного HTML. Однако CSS более точен, проработан, предоставляет большие возможности при ясном синтаксисе. Кроме того, применение каскадных таблиц стилей делает процесс разработки HTML-страницы более понятным, более гибким, более управляемым. Каскадными они называются потому, что действуют на все дерево вложенных объектов, то есть каскадом: если тело документа содержит заголовок и абзац, и к телу документа был применен стиль «сделать фон красным», то и заголовок и абзац (нижестоящие элементы) тоже будут иметь красный фон. В общем виде CSS выглядит следующим образом:

selector {property: value;}

selector — к какому конкретно HTML-тэгу (или тэгам) применяется новое свойство. Например, body применит ко всему телу документа.

property — какое свойство будем менять. Например, это будет цвет фона, background-color.

value — новое значение изменяемого нами свойства.

Таким образом, в HTML изменение фона страницы можно добиться, указав на странице: <body bgcolor="#FF0000">. Средствами CSS то же самое выглядит следующим образом: body {background-color: #FF0000;}

Существует три способа применить CSS к документу:

1) непосредственно написав его в HTML, применив к данному элементу атрибут style.

<body style="background-color: #FF0000; "></body>

2) описав стиль внутри HTML документа.

<html>

<head>

<title>Example</title>

<style type="text/css">

body {background-color: #FF0000;}

</style>

</head>

<body>

<p>This is a red page</p>

</body>

</html>

3) выделив стили в отдельный файл и подключив данные стили к документу при помощи тэга link:

<linkrel="stylesheet" type="text/css" href="style/style.css"/>

Наибольшее распространение имеет третий способ, потому что он позволяет отделить содержание документа от его оформления.

Содержание может меняться, в том числе и динамически подгружаться на страницу. А стиль при этом остается неизменным. Кроме того, внешние стили позволяют написать один стиль и применить его ко всему сайту сразу (то есть к нескольким HTML-документам).

Широкое применение в CSS имеет понятие класс. Класс, с точки зрения CSS, — это именованный набор атрибутов, который целиком может быть применен к любому объекту на странице.

Например:

.site{

color: navy;/\* Синий цвет текста \*/ font-style: italic;/\* Курсивное начертание \*/

}

После такого объявления все элементы, имеющие установленный класс site, будут иметь синий цвет и курсивный шрифт:

<p class="site">Текст, который будет выведен синим курсивом</p>

<b class="site">Объявление</b>

Последняя строка предписывает браузеру вывести слово «Объявление» жирным шрифтом. Однако, поскольку к ней дополнительно применен класс site, наше «Объявление» будет выведено синим, наклонным и жирным шрифтом одновременно.

### 2.1.3 DOM (Document Object Model)

Полученная в HTML/XHTML/XML страница отображается браузером. При этом с программной точки зрения каждый входящий в документ элемент (в том числе и сам документ) являются объектом.

DOM — это не зависящий ни от платформы, ни от языка программирования программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому документов HTML, XHTML, XML; а также менять содержимое, структуру, оформление таких документов. Несмотря на все усилия специалистов W3C по стандартизации, модель DOM и поведение входящих в нее объектов до сих пор отличаются друг от друга в разных браузерах. Отличия несущественны, но они есть. Поэтому для корректной работы клиентской части web-приложения лучше перед применением какой-либо особенности проверить, поддерживается ли данная особенность данным браузером. Для отображения страницы DOM не нужен, браузер при форматировании страницы и выводе ее на экран не обязан знать о существовании внутри него каких-то объектов. Но факт существования DOM дает возможность оживить клиентскую часть web-приложения, динамически изменяя содержимое и/или оформление страницы.

Согласно DOM-модели, документ является иерархией объекта. Каждый HTML-тэг образует отдельный элемент-узел, каждый фрагмент текста — текстовый элемент и так далее. Проще говоря, DOM — это представление документа в виде дерева объектов. Это дерево формируется за счет вложенной структуры тэгов и текстовых фрагментов страницы, каждый из которых образует отдельный узел.

Пусть есть следующая HTML-страница:

<html>

<head>

<title>Заголовок</title>

</head>

<body>

Прекрасный документ

</body>

</html>

В результате отображения указанной страницы в браузере будет создано следующее дерево объектов (Рисунок 2.1):

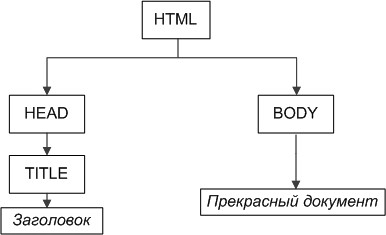


Рисунок 2.1 – Пример структуры объектов HTML-документа

Каждый DOM-элемент является объектом, то есть предоставляет методы и свойства для манипуляции своим содержимым, для доступа к родителям и потомкам.

Существует несколько способов получить для последующих манипуляций данный конкретный элемент. Можно, например, взять родительский элемент, получить список всех его потомков и искать в списке. Можно получить по имени тэга массив всех элементов данного типа. Самый простой из всех возможных способов — это получение элемента по его идентификатору. Для этой цели в тэг добавляется атрибут id (уникальный в пределах данной страницы идентификатор):

<div id="mydiv>

<table id="table1">

<tr id="oneRow">

<td id="firstCell">Первыйстолбец</td>

<td id="secondCell">Второйстолбец</td>

</tr>

</table>

</div>

При таком подходе нужный нам элемент находится однозначно и очень просто.

Пример на языке JavaScript:

var myDiv = document.getElementById ("mydiv");

Кроме элементов, их свойств и набора функций для манипуляции с ними, DOM предоставляет еще и события. Например, событие click происходит при клике мышкой на элементе, событие mouseover — когда указатель мыши перемещается над указанным элементом. На любое из предоставляемых событий можно посадить свой обработчик. Обычно обработчики называют по схеме «on+имя\_события». Указанное выше событие click можно перехватить, назначив на данный элемент функцию onclick:

<tdid="firstCell"

onclick="alert ('Вы нажали на первый столбец');">

Первый столбец

</td>

Реакции на событие click можно достичь и иным способом, просто присвоив свойство onclick нужного нам элемента:

document.getElementById ("firstCell").onclick = function () { alert ("Спасибо, что нажали на первый столбец");

};

C программной точки зрения между двумя приведенными способами не существует различий, в том и в другом случае код Вашего обработчика будет выполнен.

Несмотря на все усилия международного web-консорциума, направленные на стандартизацию, единой DOM-модели в разных браузерах не существует до сих пор. Что-то работает только и исключительно в Microsoft Internet Explorer, что-то — только в Mozilla Firefox. Обойти данное обстоятельство несложно, но это требует определенных трудозатрат и внимания.

### 2.1.4 JavaScript

Есть несколько способов доступа к DOM-объектам. Наиболее распространенные из них на сегодняшний день базируются на языке JavaScript и его расширениях. Причины такого распространения в том, что JavaScript работает практически во всех существующих браузерах и прекрасно интегрирован с HTML/CSS.

JavaScript — это объектно-ориентированный язык. Иногда его называют прототипно-ориентированным: особенность языка заключается в том, что в нем нет классов. Классов нет, но есть объекты. Более того, любая сущность в языке есть объект, который может быть перекрыт и изменен. Можно изменить, например, поведение целого числа (int), добавив ему какие-то специфические, необходимые методы. Достигается это за счет использования прототипов.

JavaScript — слабо-типизованный язык, контроль типов существует, но не является строгим. Поскольку все в языке есть объект и контроль типов слабый, то одной и той же переменной можно присваивать все, что угодно: целые числа, вещественные числа, функции, другие объекты и так далее.

JavaScript — это интерпретируемый язык, так называемый скрипт. Между текстовым представлением программы и ее выполнением нет промежуточных стадий, нет компиляции и компоновки. Написанный Вами текст будет выполняться построчно при помощи специальной встроенной в браузер программы-интерптератора.

JavaScript автоматически работает с памятью и производит сборку мусора. Специально освобождать выделенное под объекты пространство в памяти не нужно, интерпретатор сделает это самостоятельно.

Существует три способа вставить JavaScript в HTML-страницу:

1) внутри страницы:

<scripttype="text/javascript"> alert ('Hello, World!');

</script>

2) внутри тэга: <td id="firstCell"

onclick="alert ('Вы нажали на первый столбец');">Первый столбец</td>

3) вынесение скрипта в отдельный файл. Файл со скриптом подключается при помощи следующей конструкции:

<scripttype="text/javascript"src="someFile.js"></script>

JavaScript применяется не только в клиентской части web-приложений. На его основе построен ряд серверных средств, например, NodeJS. Кроме того, JavaScript широко используется в браузерных операционных системах (IndraDesktop WebOS, IntOS).

### 2.1.5 JQuery

JQuery представляет собой популярную библиотеку JavaScript, которая позволяет писать более высокоуровневые, более управляемые программы на языке JavaScript. Использование JQuery на начальном этапе может вызывать некоторые трудности из-за несколько необычного синтаксиса. А между тем суть довольно проста. Вся работа с JQuery ведется с помощью функции $. Условно можно разделить два способа применения функции $:

$("селектор") возвращает объект JQuery. При этом селектор –выражение, согласно которому отбираются DOM-объекты. Это выражение по своему синтаксису полностью соответствует принятым в CSS соглашениям. Например:

$("#mydiv") — возвращает объект с id="mydiv".

$(".greenPanel") — возвращает все объекты данного HTMLдокумента, для которых установлен class="greenPanel".

$("a") — возвращает все ссылки, существующие в данном документе.

Каждый из операторов JQuery возвращает объект, отобранный или измененный в результате проведенных манипуляций. Эта особенность позволяет проводить каскадирование операций.

$("div.test").add ("p.quote").addClass ("blue").slideDown ("slow");

В результате одной строки сначала находим на странице все элементы <div>, которые имеют установленный класс test. В полученное множество добавляем элементы <p>, имеющие класс quote. Далее всему полученному набору элементов устанавливаем class="blue" и визуально плавно спускаем их вниз по экрану. Приведенные примеры демонстрируют емкость JQuery, возможность выполнения значительного числа операций небольшим количеством строк кода.

Библиотека JQuery состоит из ядра и набора подключаемых (при необходимости) частей — плагинов, выполняющих специализированные функции. Например, в виде отдельного плагина оформлены процедуры работы с пользовательским интерфейсом (JQuery UI).

### 2.1.6 AJAX

AJAX (Asynchronous Javascript and XML) — набор технологий, предназначенных для динамического обмена данными с сервером. AJAX позволяет обмениваться данными в фоновом режиме: пользователь не замечает, когда его браузер запрашивает данные. Это асинхронная технология, то есть ожидание ответа сервера (то есть зависать при этом клиентскому приложению не нужно). Вместо этого в запросе указываются две функции, одна из которых будет отработана в случае успешного завершения запроса, а другая — в случае возникновения ошибки. Сама программа на клиентской части продолжает свое выполнение, не ожидая результата запроса. Совместно с DOM-моделью AJAX позволяет сделать странички web-приложений динамичными, интерактивными, оживить их поведение. Как это следует из названия, AJAX базируется на JavaScript, является его расширением. Технологии AJAX поддерживаются практически всеми современными браузерами. В основе работы AJAX лежит объект XMLHttpRequest. JQuery содержит набор функций обертки для работы с AJAX, что позволяет писать ассинхронные запросы максимально простым способом. Следующий пример загружает с сервера JavaScript код и выполняет его:

$.ajax ({ type: "GET", url: "test.js", dataType: "script"

});

Вызов $.ajax () возвращает объект типа XMLHttpRequest. В большей части случаев возвращаемое значение не используется, но оно все-таки доступно, например, при необходимости прервать выполняющийся запрос. Существует возможность установить параметры запроса по умолчанию, которые будут распространяться на все вызываемые в программе запросы. Делается это при помощи функции $.ajaxSetup ().

## 2.2 Технологии создания серверной части

### 2.2.1 Web-серверы

Web-сервер — это совокупность аппаратно-программных средств, которая принимает HTTP-запросы от клиентов (как правило, web-браузеров) и выдает HTTP-ответы. В простом случае ответ web-сервера — это содержание расположенных на нем файлов: документов, изображений, фильмов и так далее. Такие данные называют статическими: если пользователь запросит несколько раз информацию по одному и тому же адресу, web-сервер выдаст в точности одно и то же содержимое. Для динамического подхода, то есть когда выдаваемая информация будет меняться во времени и/или зависеть от параметров запроса, одного web-сервера недостаточно. В этом случае web-сервер служит переходным звеном между клиентом и некой программной средой, формирующей содержимое HTML-страницы. И здесь напрашивается вопрос: а зачем тогда нужен web-сервер, если динамическую страницу все равно формирует приложение.

При небольшой нагрузке на приложение — несколько одновременно подключенных пользователей (единицы) — web-сервер не является абсолютно необходимым звеном. Практически все существующие на сегодняшний день пакеты разработки web-приложений имеет в своем составе легковесный web-сервер. Достаточно запустить среду разработки на выполнение, и к ней можно подсоединиться браузером и работать с ней по протоколу HTTP.

Проблемы начинаются тогда, когда число одновременно работающих клиентов превышает некоторое критическое значение. И одна из функций web-сервера — это максимально возможное снижение нагрузки за счет ее распределения по разным частям системы. Оптимизация может затрагивать не только работу самого web-сервера, но и начинаться непосредственно с уровня ядра операционной системы. Современные операционные системы могут быть настроены таким образом, что после соединения с клиентом они не будут беспокоить web-сервер до тех пор, пока клиентская сокета не получит данные (с сокеты можно читать входной поток данных), либо пока не придет полный HTTP-запрос.

Такой подход позволяет web-серверу меньше времени простаивать в ожидании получения данных, повышая эффективность системы в целом.

Все современные (полновесные) web-серверы используют асинхронную обработку поступающих от клиентов запросов. То есть после получения запроса от клиента сервер переадресует этот запрос потоку обработчика (worker). Сам сервер при этом продолжает принимать запросы от клиентов и переадресовывать их обработчикам.

Существует несколько стратегий работы с обработчиками:

1) количество worker может быть жестко задано. При старте сервера запускается, например, 10 потоков. При получении 11-го запроса сервер выдаст клиенту ответ временно недоступен;

2) Worker могут порождаться сервером динамически в зависимости от загрузки. При этом оговаривается начальное число worker, максимально допустимое количество, минимальное и максимальное количество свободных worker;

3) Worker может быть как процессом, так и потоком. Различные процессы не взаимодействуют между собой и не могут разделять данные. В отличие от процессов, потоки имеют общее разделяемое пространство данных и могут обмениваться информацией между собой;

4) Worker-процессы могут запускаться как внутри родительского серверного процесса, так и снаружи. При последнем варианте worker могут быть распределены по разным компьютерам;

5) Worker — вне родительского сервера могут представлять собой непосредственно процессы среды web-приложения;

6) сервер может отслеживать разные варианты распределения нагрузки между порожденными worker: равномерную или поочередную загрузку.

Комбинируя эти факторы, можно попытаться добиться наилучшей производительности системы в целом.

Для логического разделения, оптимального распределения нагрузки по вычислительным мощностям Worker могут запускаться и на другом (-их) компьютере (-ах), не обязательно на том же самом, где расположен сам принимающий запросы от клиентов сервер. В этом случае общение между сервером и worker происходит по определенному сетевому протоколу. Это уже не может быть HTTP, потому что вместе с полученным запросом worker передается расширенная информация об окружении сервера. В частности, в качестве такого протокола общения может применяться FastCGI.

В современном интернете существует как минимум несколько десятков решений на тему web-серверов как бесплатных, так и коммерческих. Как правило, коммерческие продукты имеют графический интерфейс управления и настройки сервера, с ними работать гораздо проще. В противоположность этому свободно распространяемые программные продукты обычно конфигурируются при помощи текстовых файлов.

### 2.2.2 Технология MVC

MVC (Model View Controller) — это концепция, парадигма, подход в том плане, что MVC-подход не опирается на какой-то конкретный язык программирования, не зависит от выполняемых задач и может существовать как на клиенте, так и на сервере. Идеология заключается в том, что работу с данными (model), представление на экране (view) и обработку внешних запросов (controller) разделяют между собой (Рисунок 2.2).

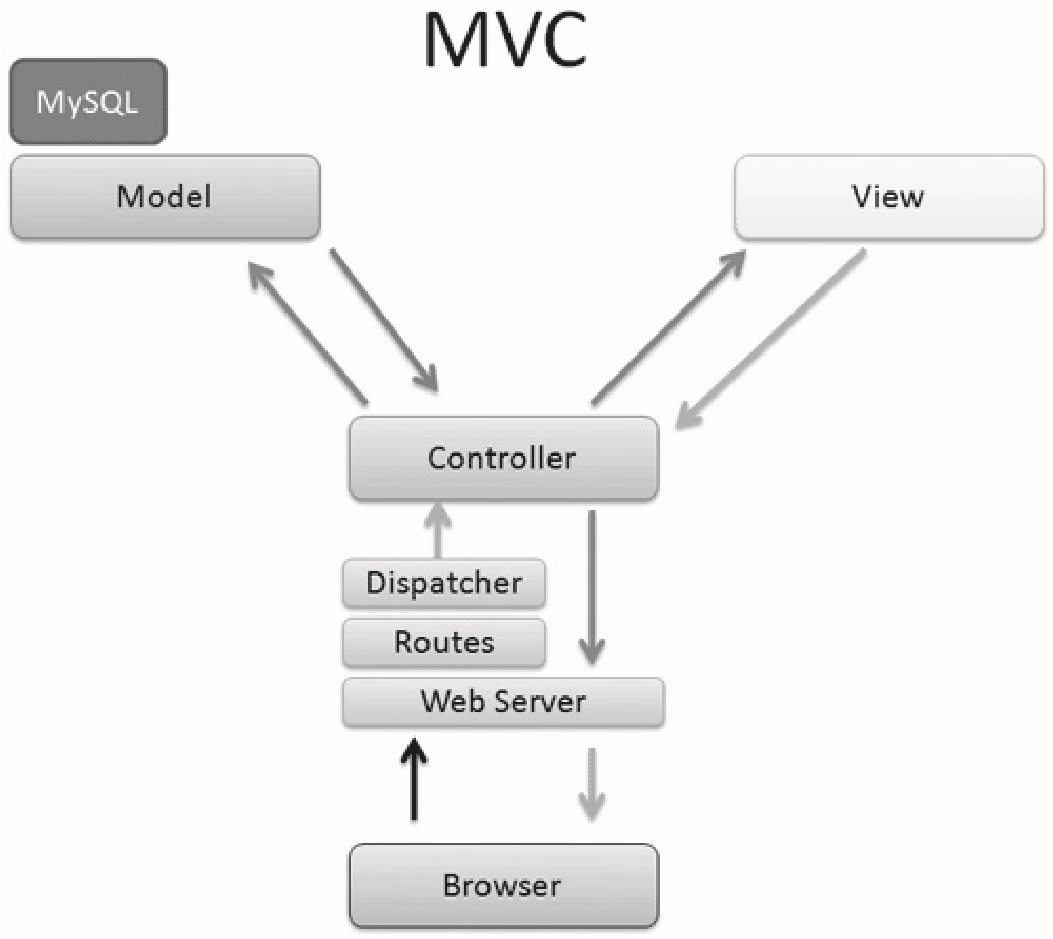


Рисунок 2.2 – Структура MVC-модели

При этом работающая с данными модель может быть либо пассивной, то есть ее каждый раз нужно спрашивать, не изменились ли данные, либо активной, в этом случае она сообщает всем зависящим от нее views, что данные изменились и надо бы их перерисовать.

Такое разбиение позволяет, во-первых, обезопасить себя от неожиданных изменений в поведении приложения в целом. При изменении модели мы можем быть уверены, что вносимые изменения отразятся только на данных и ни на чем больше. Во-вторых, подход позволяет одним и тем же данным сопоставить несколько представлений. И наоборот, сделать одно представление (предположим, статистику по месяцам), которое работало бы с разными типами исходных данных.

Забегая вперед, отметим, что достаточно часто в качестве view фигурируют шаблоны (например, FreeMaker), а в качестве model непосредственное отображение объектов на СУБД, то есть ORM.

Концепция MVC оказалась достаточно удачной, вследствие чего она служит основой для ряда framework как серверных, так и клиентских. Из серверных Java framework можно отметить Play, из JavaScript клиентских – backbone.js.

### 2.2.3 Технологии объектно-реляционных отображений (ORM)

ORM (Object Relational Mapping) — это такая технология работы с базами данных, при которой программа напрямую в СУБД не обращается. Вместо этого программа оперирует объектами, самостоятельно отображающимися на реляционную структуру базу данных (Рисунок 2.3). Сама низкоуровневая работа с базой данных происходит за кадром.

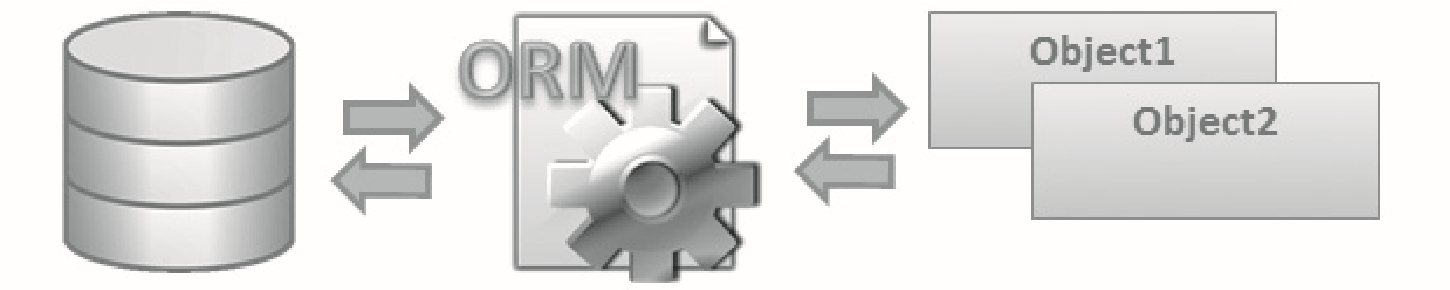


Рисунок 2.3 – Структура ORM-технологии

Достаточно создать объект, например, User, определенным образом описанный. И после этого можно совершать с ним различные действия, специфичные для базы данных: записать его в базу, выбрать из базы по определенным критериям, рассматривать другие объекты не как подчиненные таблицы, а как поля основного объекта и т.д.

Для работы с базами данных Java всегда имеет механизм JDBC (Java Data Base Connectivity).

ORM — это следующий уровень, стоящий над JDBC. Применение ORM скрывает детальность работы с СУБД, позволяет обойтись без громоздких конструкций на языке SQL и без присваивания параметров JDBC-запросу. Все вместе (ORM + JDBC) позволяет писать программы таким образом, что они становятся независимыми от базы данных. Одна и та же программа может работать как с Oracle, так и с MS SQL.

Оборотной стороной применения ORM является снижение скорости работы с базой данных. Правда, к этому утверждению нужно добавить «как правило», так как в реальности на скорость работы оказывают влияние сразу несколько факторов: и структура данных, и наличие/правильное использование индексов, и построение запросов. Но в общем случае ORM снижает скорость работы приложения, повышая при этом его гибкость и мобильность.

# **Библиографический ﻿список**

1. Аншина М.Л. Технологии создания распределенных систем / А.А.Цимбал, М.Л.Аншина. — СПб.: Питер, 2003.
2. Браун Л. Программирование для Web / Л.Браун, М.Холл. — Sun Microsystems, 2002.
3. Дейтел Х.М. Технологии программирования на Java / Х.М. Дейтел, П.Дж.Дейтел, С.И.Сантри. — М.: ООО «Бином-Пресс», 2003.
4. Таненбаум Э. Распределенные системы. Принципы и парадигмы / Э.Таненбаум, М.Ван Стеен. — СПб.: Питер, 2003.
5. Тузовский А.Ф. Проектирование клиент-серверных приложений /А.Ф.Тузовский — Томск: Изд-во Томского политехнического университета, 2010.
6. Java Hibernate. Часть 1 — Введение [Электронный ресурс]. URL: http://javaxblog.ru/article/java-hibernate-1/(дата обращения: 05.08.20). Заголовок с экрана.
7. HIBERNATE — Relational Persistence for Idiomatic Java [Электронный ресурс]. URL: http://samsonych.com/lib/hibernate/index. html (дата обращения: 05.08.20). Заголовок с экрана.