Лясин Д.Н., Абрамова О.Ф.

Основы проектирования Web-приложений
часть 2

Волжский

2020

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
часть 2

Электронное учебное пособие

2020

УДК 004.45(07)
ББК 32.973я73
Л 976

Рецензенты:
кандидат педагогических наук, доцент кафедры методики преподавания математики и физики, ИКТ ФГБОУ ВО «Волгоградский государственный социально-педагогический университет»
Филиппова Е.М.;
кандидат физ.-мат. наук, доцент, зав. кафедрой «Математики, информатики и естественных наук» Волжского филиала ФГАОУ ВО ВолГУ
Полковников А.А.

Издается по решению редакционно-издательского совета
Волгоградского государственного технического университета

Лясин, Д.Н.

Основы проектирования Web-приложений (часть 2) [Электронный ресурс] : учебное пособие / Д.Н. Лясин, О.Ф. Абрамова ; Министерство науки и высшего образования Российской Федерации, ВПИ (филиал) ФГБОУ ВО ВолгГТУ. – Электрон. текстовые дан. (1 файл: 630,18 КБ). – Волжский, 2020. – Режим доступа: http://lib.volpi.ru. – Загл. с титул. экрана.

ISBN 978-5-9948-3703-0

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 инструментов, обеспечивающих работу как самих web-серверов, так и скриптов, обеспечивающих динамическую обработку поступающих запросов. Вторая часть пособия посвящена обзору работы серверной стороны web-приложений: развертывание и конфигурирование web-сервера, описание механизма динамического обмена данными между клиентской и серверной сторонами web-приложений, инструментарий взаимодействия с базами данных, поддержка состояний.

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. 2, табл. 3, библиограф.: 20 назв.

ISBN 978-5-9948-3703-0

Оглавление

Введение

Современные Web-приложения проектируются и разрабатываются в рамках различных парадигм и с использованием разнообразных инструментов. Тут и классические приложения, ориентированные на серверную обработку, и Web-сервисы, в которых серверная сторона служит лишь поставщиком данных, и распределенные системы, использующие каскады вызовов между различными узлами глобальной паутины для доступа к скриптам, данным, сервисам. При всем этом разнообразии наличие серверной стороны, отвечающей на запросы клиентов из браузеров, мобильных и оконных приложений, является по-прежнему неизменной константой. Web-сервер является конечной точкой обработки HTTP-запросов, местом сосредоточения логики управления данными или, по крайней мере, местом, откуда скрипты, реализующие подобную логику, попадают на клиентскую сторону. В этой связи полноценное знакомство с Web-технологиями немыслимо без изучения принципов работы как самого сервера, так и скриптов, на него загружаемых.

Вторая часть пособия знакомит читателя с функционированием серверной части web-приложения, а также с основами взаимодействия клиентской и серверной частей. Она посвящена таким вопросам, как развертывание и конфигурирование web-сервера, написания скриптов для динамической обработки запросов, взаимодействия с базами данных, обработки синхронных и асинхронных HTTP-запросов. Ознакомившись с первыми двумя частями пособия, можно получить представление о базовом инструментарии web-разработки.

1. Web-приложение на серверной стороне

Перенос функционала Web-приложения на сторону сервера может быть обусловлен:

- необходимостью разгрузки компьютера клиента от сложных или ресурсоемких вычислений;

- требованиями обеспечения безопасности работы с ресурсами приложения;

- задачами обеспечением коммерческой тайны для кода приложения;

- стремлением уменьшения объема данных, которыми обмениваются клиент и сервер.

Функции обработки запросов клиентской стороны Web-приложения и формирования ответа реализуются Web-сервером, а также установленными плагинами и сценариями (скриптами) серверной стороны.

Web-сервер – это программное обеспечение, обеспечивающее доставку контента конечному пользователю по сети с использованием протокола HTTP (HTTPS). Основные функции Web-сервера:

- обмен контентом с клиентской стороной;

- поддержка динамически генерируемых страниц;

- аутентификация и авторизация пользователей;

- ведение журнала обращений пользователей к ресурсам;

- обеспечение безопасности обмена данными с клиентской стороной за счет поддержки протокола HTTPS.

Для обработки клиентских запросов Web-сервер применяет два подхода:

1. Находит файл, указанный в запросе клиента (*html, xml, css, jpg*, …) и возвращает его содержимое как результат обработки запроса.

2. Выполняет сценарий, связанный параметрами запроса, и передает клиенту результаты работы этого сценария (динамические страницы).

К настоящему моменту разработчику доступно множество свободно распространяемых Web-серверов, которые можно использовать как программную платформу серверной части Web-приложения. Наиболее популярными Web-серверами на сегодняшний момент являются: Apache, ngnix, MS IIS, Node.js, Google Web Server [3].

Web-сервер Apache является, согласно статистике, самым популярным сервером в глобальной сети, чему способствуют такие его достоинства, как кроссплатформенность, поддержка многих языков программирования, встроенные средства обеспечения безопасности, модульная архитектура, позволяющая расширить функциональность сервера, поддержка механизма виртуальных хостов.

Рассматриваемые примеры выполнения лабораторной работы ориентированы на этот Web-сервер, что, однако, не является ограничением студентам по выбору сервера для реализации задания к лабораторной работе.

Скачать дистрибутив и ознакомиться с инструкцией по установке Web-сервера Apache можно на официальном сайте [1]. Если воспользоваться услугами хостнига, то вы получите возможность работы с уже установленным сервером. Для учебных целей вполне функционален локальный Web-сервер, например, широко известный набор Web-разработчика OpenServer, который позиционируется разработчиками как «портативная серверная платформа и программная среда, созданная специально для веб-разработчиков с учётом их рекомендаций и пожеланий». OpenServer включает набор необходимых Web-разработчику инструментов (Web-сервер Apache, интерпретатор PHP, СУБД MySQL, а также ряд полезных расширений). Скачать OpenServer и ознакомиться с порядком его установки и настройки можно на сайте проекта [7]. Еще одно удобное средство для

знакомства с миром Web-разработки – облачные среды разработки, которые предоставляют собой готовые среды для создания и отладки Web-приложений на облачных серверах, с поддержкой основных языков программирования и СУБД. Примерами подобных сред являются: koding.com или cloud9.

Установленный Web-сервер может потребовать конфигурирования под особенности работы или уникальные требования Web-разработчика. Конфигурирование Web-сервера Apache возможно 4 различными способами:

1. Конфигурирование при инсталляции/сборке.
2. Конфигурирование при запуске с использованием параметров командной строки.
3. Изменение конфигурационного файла *httpd.conf*.
4. Изменение локальных файлов *.htaccess*.

Первый способ конфигурирования позволяет настроить самые общие параметры работы Web-сервера – каталог установки, прослушиваемый порт, доменное имя сервера и др.

При запуске Web-сервера (2-й способ конфигурирования) можно указать ряд опций, управляющих его работой:

Запуск с альтернативным файлом конфигурации

```
/usr/local/apache2/bin/apachectl –f usr/local/apache2/conf/alt
httpd.conf
```

Запуск Web-сервера как службы

```
apache –d путь_к_apache –i
```

Однако самым эффективным средством настройки Apache является правка конфигурационного файла *httpd.conf*, содержащего набор директив, управляющих всеми аспектами работы сервера. Директивы конфигурации *httpd.conf* сгруппированы в три основных раздела [5]:

1. Директивы, управляющие процессом Apache в целом (глобальное окружение).

2. Директивы, определяющие параметры "главного" сервера, или сервера "по умолчанию", который отвечает на запросы, не обрабатываемые виртуальными хостами.

3. Установки для виртуальных хостов, позволяющие обрабатывать HTTP-запросы одним-единственным сервером Apache, но направлять их по раздельным адресам IP или именам хостов.

Примеры директив первого типа:

```
Timeout 300  #время до тайм-аута
KeepAlive on #разрешить устанавливать долговременные соедине-
ния
```

#Корневой каталог сервера

```
ServerRoot "C:/Program Files/Apache Group/Apache"
```

#Подключение модулей

```
LoadModule anon_auth_module modules/ApacheModuleAuthAnon.dll
```

Для настройки основного сервера применяются директивы:

```
Port 80   #Номер порта, к которому подключен сервер
```

#Каталог с документами сервера

```
DocumentRoot “C:\Program Files/Apache Group/Apache/htdocs”
```

Набор опций для настройки кеширования: CacheRoot, CacheSize, CacheMaxExpire, CacheLastModifiedFactor, CacheDefaultExpire, NoCache

Настройка кодировок:

```
AddDefaultCharset WINDOWS-1251
AddCharset WINDOWS-1251 .cp-1251   .win-1251
```

Для каждой директории сервера можно задать индивидуальные настройки, задающие права доступа к папке и подпапкам, разрешенные документам в этой папке функции Apache:

```
  <Directory «C:/Program
Files/ApacheGroup/Apache/htdocs/site1">
   Options Indexes Includes
   AllowOverride All
   Order allow,deny
   Allow from apache.org
   Allow from 195.209
  </Directory>
```

Директива Options позволяет управлять тем, какие функции сервера доступны для использования в каталоге, указанном в секции <Directory> файла access.conf или в файле .htaccess.

Возможные значения Options:

- *None.* В указанном каталоге не разрешается использование каких-либо функций.
- *All* В указанном каталоге разрешается использование всех возможностей.
- *ExecCGI* В указанном каталоге разрешается выполнение сценариев CGI.
- *Includes* В указанном каталоге разрешено использование серверных включений - SSI (Server Side Includes).
- *Indexes* – допускает использование директив управления индексацией каталога, например разрешает выдачу листинга каталога, если в нем нет файла index.html;
- *IncludesNoExec* Разрешает использование в указанном каталоге серверных включений, но запрещает запуск из них внешних программ.

Директива AllowOwerride перечисляет те опции, которые могут быть переопределены для директории в локальном файле .htaccess.

Директивы Allow и Deny позволяют разрешить или запретить доступ к содержимому директории с заданных адресов.

Аналогичные настройки можно задавать для файлов <Files> </Files> и URL <Location></Location>

```
<Files .htaccess> # запретить доступ к .htaccess
  Order allow,deny
  Deny from all
</Files>
```

Директивы для виртуальных хостов позволяют поддерживать одним Apache-сервером нескольких серверов с различными доменными именами или ip-адресами. Объявляется виртуальный хост следующей директивой:

```
<VirtualHost *:80>
 ServerAdmin webmaster@serv1.ru
 ServerName www.serv1.ru
 DocumentRoot "z:/home/serv1/www"
 ScriptAlias /cgi/ "z:/home/serv1/cgi/"
 ErrorLog z:/home/serv1/error.log
 CustomLog z:/home/serv1/access.log common
</VirtualHost>
```

В директиве VirtualHost могут быть указаны любые директивы httpd.conf, только применяться они будут не к основному обслуживаемому серверу, а к описываемому хосту.

Еще одна возможность конфигурирования работы Web-сервера – использование файлов .htaccess, которые задают локальные настройки для той директории, в которой эти файлы размещены. Настройки в .htacess дополняют (или перекрывают) глобальные настройки из файла httpd.conf (если подобное переопределение не запрещены директивой *AllowOwerride*).

Типичными для файлов .htacess являются инструкции:

```
 #запрет доступа с IP-адреса
 Order Allow,Deny
 Allow from all
 Deny from 111.111.111.111

 #запретить доступ к каталогу
```

```
Deny from all

#Перенаправить обращение к страницам
Redirect 301 /index.html http://www.domain.ru/new_addr.html

#запретить выдачу листинга содержимого
Options –Indexes

#защитить страницы каталога паролем
AuthName «Вход на защищенную страницу»
AuthType Basic
AuthUserFile  Путь к файлу htpasswd
require valid-user
```

2. Скрипты серверной стороны Web-приложения

Помимо самого Web-сервера в обработке запросов клиентов и формировании HTTP-ответов могут участвовать плагины и сценарии.

Плагин (*plug* - in) – независимо компилируемый программный *модуль* (чаще всего – динамическая библиотека), динамически подключаемый к основной программе, предназначенный для расширения или использования ее возможностей.

Сценарий (*скрипт*, *script*) – *программа*, которая автоматизирует некоторую задачу, в выполнении которой заинтересован клиент.

Для разработки серверных скриптов существует несколько технологий:

- *Common Gateway Interface* (*CGI*)

Интерфейс связи Web-сервера с внешним приложением. Приложение отвечает за формирование ответа через свой стандартный поток вывода, сервер обеспечивает взаимодействие с клиентом по протоколу HTTP и взаимодействие со скриптом;

- *FastCGI*

Усовершенствованный вариант CGI с улучшенными характеристиками производительности и безопасности;

- *SAPI* (Server Application Program Interface, встраиваемые модули)

Модули (динамические библиотеки), которые выполняются в контексте процесса Web-сервера;

- *JavaEE* (*Java Platform Enterprise Edition*)

Подход, основанный на использовании потоков исполнения в виртуальной машине для обработки серверных скриптов-сервлетов.

2.1. Серверные скрипты в стандарте интерфеса CGI

Рассмотрим пример процесса динамического формирования ответа сервера на HTTP-запрос на примере CGI-скриптов. CGI не является языком программирования, CGI-скрипты могут быть написаны на любом языке. CGI говорит о том, как сервер и приложение общаются друг с другом для обработки запроса и возврата ответа. Вот как, например, выглядит простейший CGI-скрипт на языке Си++:

```
#include <iostream>
using namespace std;
int main()
{
   cout<<"Content-Type: text/html\n\n";
   cout<<"<HTML><HEAD><TITLE>Динамическая страница
</TITLE></HEAD>\n";
   cout<<"<BODY><DIV>Всем привет</DIV></BODY></HTML>\n";
   return 0;
}
```

Сервер, обнаружив, что запрос не направлен к статическому ресурсу (htm-странице, css-файлу, файлу с картинкой, шрифтом и т.п.), запускает скрипт для обработки этого запроса. Результат работы скрипта, выведенный им на свой стандартный поток вывода, принимается сервером и отправляется клиенту как результат обработки запроса. При этом сервер, обрабатывая запрос, формирует ряд переменных окружения, которые дос-

тупны скрипту и содержат параметры пришедшего запроса. К подобным переменным относятся:

Таблица 1. Переменные окружения CGI-скрипта

Переменная окружения	Описание
HTTP_COOKIE	Хранит набор «кук» в виде пар «ключ значение».
REMOTE_ADDR	IP-адрес клиента, выполняющего запрос
QUERY_STRING	Строка запроса (URL-encoded), передаваемая методом GET
SERVER_NAME	Имя сервера
SERVER_ADDR	IP-адрес сервера
HTTP_USER_AGENT	Информация об агенте пользователя (браузере)

Подробнее рассмотрим процесс и нюансы создания серверных скриптов с использованием языка программирования PHP, как одного из наиболее распространенных в среде Web-разработки.

PHP (Hypertext Processor) – скриптовый язык программирования, предназначенный для написания скриптов серверной стороны (BackEnd) Web-приложений. Разработан в 1994г. датским программистом Расмусом Лердерфом на основе языка Perl. В 1997г израильские программисты Энди Гутманс и Зеес Сураски выпустили новый код интерпретатора под версией 3.0. На сегодня актуальна версия 7.X.

Язык PHP использует смешанную схему обработки кода с использованием транслятора и интерпретатора. Сначала текст скрипта транслируется в байт-код. Затем байт код передается интерпретатору и исполняется. Преимущества использования байт-кода:

- Интерпретатор контролирует ресурсы, и нет необходимости в их освобождении;
- Фаза трансляции выполняется один раз, после чего байт-код может быть выполнен многократно;
- Байт-код может быть закеширован и повторно выполнен.

На рис.1 приведена схема взаимодействия Web-севера и движка PHP при обработке запроса клиента.

Для конфигурирования интерпретатора PHP используется файл php.ini [8]. Директивы этого файла определяют режим работы интерпретатора, и их изменение позволяет более гибко настроить работу интерпретатора под требования конкретных проектов.

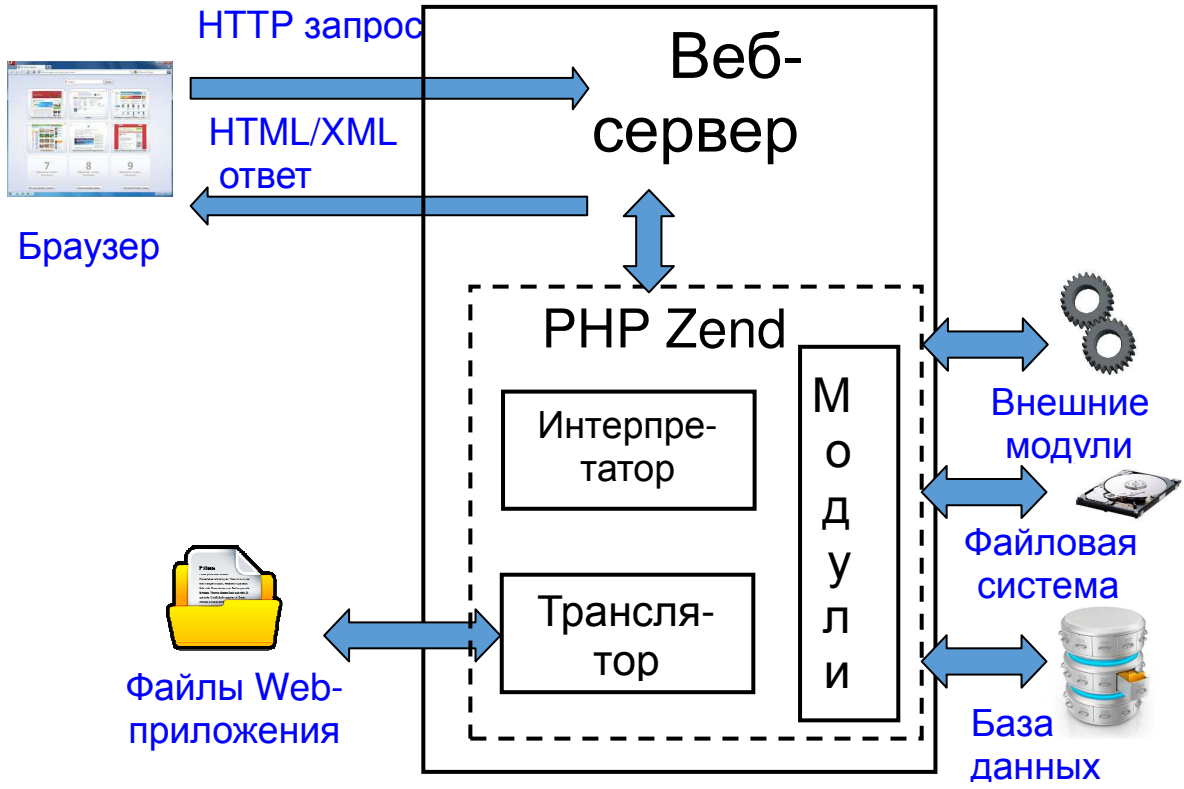

Рисунок 1. Взаимодействие Web-сервера и PHP

Ниже приведен перечень некоторых директив настройки php:

Разрешить/запретить вывод ошибок интерпретатора в браузер

```
display_errors = On | Off
```

Установить уровень ошибок, которые будет отображать интерпретатор

```
error_reporting = E_ALL & ~E_NOTICE
```

Разрешить/запретить сокращенный тег открытия php-скрипта (<? вместо <?php)

```
short_open_tag = On | Off
```

Максимальное время исполнения скрипта

```
max_execution_time = 30
```

Максимальный размер данных, который можно передать через POST-запрос

```
post_max_size = 8M
```

Какие динамически загружаемые расширения должны быть загружены при старте PHP.

```
extension=php_mysql.dll
extension=php_mysqli.dll
```

Максимальный размер закачиваемого файла.

```
upload_max_filesize = 4M
```

Файл php.ini поддерживает гораздо большее число директив, приведенные выше директивы должны дать представление о том, на какие аспекты работы интерпретатора влияют настройки в данном файле, полный список директив можно посмотреть, например, здесь [3].

Основные достоинства языка PHP:

- Простота языка, обеспечивающая низкий порог вхождения для начинающих Web-разработчиков;

- Возможность PHP-скриптов работать как в режиме CGI, так и SAPI-модулей;

- Строгая ориентированность языка на Web-разработку, которая обеспечивает Web-разработчика необходимым инструментарием для решения большинства стоящих перед ним задач;

- Поддержка большинством современных framework'ов и CMS-систем (Joomla, Wordress, Zend);

- Поддержка языком объектной парадигмы: начиная с версии 5 PHP поддерживает полную объектную модель.

Недостатки PHP:

- Отсутствие обратной совместимости может удивить разработчика, когда при переходе на новую версию интерпретатора перестают работать некоторые функции (но в предварительных версиях они объявляются устаревшими, что дает возможность заблаговременно заменить эти функции в коде);

- Не поддерживается многопоточность, при необходимости ее приходится эмулировать различными алгоритмическими способами;

- Отсутствие строгой типизации данных не позволяет контролировать корректность хранящихся в объектах данных (частично устранена в версии 7).

2.2. Скрипты на языке PHP

Серверные скрипты на языке PHP обычно размещаются в файлах с расширением php, но их можно встретить также и в файлах с расширением html. Конфигурация web-сервера определяет порядок обработки запроса, если он направлен к ресурсу (файлу), через вызов php-интерпретатора:

```
AddType application/x-httpd-php .php .htm .html
AddHandler x-httpd-php .php .htm .html
```

PHP-интерпретатор запускает на выполнение файл, к которому осуществлен запрос, в качестве CGI-скрипта, при этом в рамках CGI-интерфейса скрипту передается информация о контексте запроса (параметры, заголовки HTTP-запроса). В тексте документа, обрабатывающего запрос, обычные html-теги, которые непосредственно будут переданы в ответе клиенту, могут соседствовать с директивами, предназначенными для динамической обработки запроса. Подобные директивы оформляются с

использованием парных тегов <?php … ?> (приоритетный вариант) или <? …?> (сокращенная форма). Документ с php-скриптом может выглядеть так:

```
<html>
  <head>
   <title>Страница с php-скритпом</title>
  </head>
  <body>
   <h1>
      <?php print("Привет, мир"); ?>
   </h1>
 </body>
</html>
```

В приведенном примере большая часть документа содержит статический контент в виде html-тегов, определяющих общую структуру документа, но имеет и фрагмент, который будет получен в результате выполнения php-кода. Конечно, в приведенном примере вряд ли можно говорить о динамической обработке, поскольку при каждом запросе будет формироваться страница со строкой *Привет, мир*. Но в реальных скриптах программный код формирует действительно динамическое содержимое, обрабатывая параметры запроса из заполненных в браузере форм, считывая данные из СУБД, обращаясь к внешним сервисам. Для этого у языка PHP (как и у других языков поддержки серверных скриптов) имеется мощная поддержка в виде операторов, директив, типов данных, библиотек функций и классов. Подробное знакомство с возможностями языка php выходит за рамки данного учебного пособия, но интересующиеся без труда смогут найти информации в массе книг и интернет-источниках, например в [12, 13, 14, 15]. С некоторыми инструментальными и языковыми средствами языка php читатель сможет познакомиться в следующих главах.

3. Интерактивное взаимодействие клиентской и серверной частей Web-приложений

Большинство современных Web-приложений являются интерактивными, то есть предполагающими передачу клиентской стороны на серверную информации, принятой от пользователя. Серверная сторона на основе этой информации может сформировать динамическую страницу (например, при реализации поиска или фильтрации контента), модифицировать данные, хранящиеся в базе данных приложения (запросы на добавление, удаление, изменение информации), изменить статус или программный интерфейс приложения (например, при авторизации пользователя). Одним из самых распространенных способов передачи данных от клиентской стороны к серверной является использование *форм*.

3.1. Формы как основной интерфейс взаимодействия

Форма – это интерфейсная часть документа HTML, содержащая органы управления, информация о состоянии которых может быть передана из браузера на Web-сервер. Для размещения формы внутри документа HTML используется элемент FORM [15]:

```
<form  атрибуты> ... </form>
```

Атрибуты формы позволяют задать особенности обработки данных во вложенных управляющих элементах и параметры взаимодействия браузера с сервером при передаче информации. В таблице 2 приведены возможные атрибуты тега FORM.

Таблица 2. Атрибуты тега FORM

Атрибут	Назначение	Возможные значения

`action`	Указывает обработчик, к которому обращаются данные формы при их отправке на сервер. В качестве обработчика может выступать сценарий	`<form action=”auth.php”>`
		…
		`</form>`

	серверной стороны (CGI, PHP, ASP.NET и др). Задача серверного скрипта – принять данные из формы, выполнить их обработку и сформировать новый HTML-документ, который будет возвращен в браузер как результат обработки.	

`auto-complete`	Управляет автозаполнением для элементов форм. Если автозаполнение разрешено в настройках браузера, то при первом вводе значений в управляющие элементы формы они запоминаются браузером, а при повторном вводе автоматически подставляются в соответствующие поля форм.	`<form autocomplete="on">`
		`...`
		`</form>`
		`<form autocomplete="off">`
		`...`
		`</form>`

`method`	Определяет, каким HTTP-методом (get или post) будут отправляться данные на сервер	`<form method="get">...`
		`</form>`
		`<form method="post">...`
		`</form>`

`enctype`	Определяют способ, которым данные из элементов управления будут кодироваться перед отправкой на сервер	`application/x-www-form-urlencoded`
		Вместо пробелов ставится +, символы вроде русских букв кодируются их шестнадцатеричными значениями. Принимается по умолчанию
		`multipart/form-data`
		Данные не кодируются. Это значение применяется при отправке файлов.
		`text/plain`
		Пробелы заменяются знаком +, буквы и другие символы не кодируются.

`name`	Задает уникальное имя формы, которое впоследствии можно использо-	**`<form`** `name="frmAuth"`**`>`**

	вать для обращения к объекту-форме в скриптах клиентской стороны	... **</form>**
novalidate	Отключает проверку значений элементов формы на корректность, если она предусмотрена такими их атрибутами, как pattern и required	**<form** novalidate**>** ... **</form>**
target	Определяет, куда будет возвращен HTML-документ—результат обработки данных формы	_blank Документ будет загружен в новое окно браузера _self Документ будет загружен в текущее окно. Принимается по умолчанию _parent Документ будет загружен в родительский фрейм. Если нет фрейма – в текущее окно _top Игнорирует все фреймы и загружает документ в полное окно браузера

Внутри формы, как и внутри любого другого блочного элемента, могут находиться другие HTML-элементы. Помимо прочих элементов, форма содержит элементы органов управления. Особенностью элементов управления является то, что их значения передаются на сервер, когда подается команда отправки формы. Рассмотрим основные управляющие элементы, используемые в HTML-формах.

Одним из самых распространенных тегов управляющих элементов форм является тег <input>. С его помощью можно поместить на форму различные элементы интерфейса и обеспечить взаимодействие с пользова-

телем. С помощью тега <input> можно создать текстовые поля, различного рода кнопки, переключатели и флажки.

Тип элемента, который добавляется на форму с использованием тега <input>, задается с использованием атрибута type. Возможные значения этого атрибута:

text — текстовое поле для ввода текстовой информации;

password — поле с паролем. Вводимые данные не отображаются в строке ввода, а заменяются звездочками;

radio — переключатель, допускающий выбрать только один вариант из нескольких предложенных;

checkbox — флажок, позволяющий выбрать более одного варианта из предложенных.;

hidden — скрытое поле, которое не отображается на странице, но значение которого отправляется на сервер,

button — кнопка;

submit — кнопка отправки формы;

reset — кнопка для очистки формы, сбрасывающий значения элементов в формы в первоначальное значение;

file — поле выбора имени файла, который будет передан на сервер;

image — кнопка с изображением. При нажатии на кнопку данные формы будут отправлены на сервер.

В стандарте HTML5 определен еще ряд типов элементов для <input>:

color — поле для выбора цвета
date — поле для выбора календарной даты.
datetime — поле для ввода даты и времени
email — поле для ввода адресов электронной почты
number — поля для ввода чисел.
range — ползунок для выбора чисел в указанном диапазоне

search — поле для поиска

tel — поле для ввода телефонных номеров
time — поле для ввода времени.
url — поля для ввода веб-адресов
month — поля для выбора месяца
week — поля для выбора дня недели

Для каждого элемента существует свой список атрибутов, которые определяют его вид и характеристики. Ниже приведены варианты определения элементов с использованием тега <input>:

```
    <INPUT name=“fio” type=“text”  value=“” size=”40”
placeholder=”Введите ФИО”/>

    <INPUT TYPE="hidden" NAME="idUser" VALUE="123">

    <INPUT TYPE="password" WIDTH="10" NAME="passwd">

    <INPUT TYPE="submit" VALUE="Передать">

    <INPUT TYPE="checkbox" NAME="service">Чистка

    <INPUT TYPE="checkbox" NAME="service">Сборка

    <INPUT TYPE="checkbox" NAME="service" CHECKED>Диагностика

    <INPUT TYPE="image" src="/img/button.gif" WIDTH="60" HEIGHT="30">

    <INPUT TYPE="radio" NAME="mf" VALUE="Male" CHECKED> М

    <INPUT TYPE="radio" NAME="mf" VALUE="Female"> Ж

   <INPUT TYPE ="file" NAME="datafile" size="40">

    <INPUT TYPE ="number" NAME ="quantity" min="1" max="5">

    <INPUT TYPE ="date" NAME ="bday" max="1979-12-31">

    <INPUT TYPE="range" NAME ="points" min="0" max="10">
```

Помимо <input> HTML предлагает еще ряд тегов для вставки управляющих элементов в формы. Рассмотрим некоторые из них.

Тег <select> создать элемент интерфейса в виде раскрывающегося списка, а также список с одним или множественным выбором. Конечный вид управляющего элемента зависит от использования атрибута size, ко-

торый устанавливает высоту списка. Набор элементов, из которых пользователю предлагается делать выбор, формируется как последовательность элементов <option>. Текущий выбранный элемент помечается атрибутом selected.

```
<select name="selLevel">
  <option value="novice" >Новичок</option>
  <option value="prof" selected>Профессионал</option>
  <option value="expert">Эксперт</option>
</select>
```

Если у тега <select> установлен атрибут multiple, то пользователь сможет выбрать сразу несколько элементов в списке.

Для ввода объемных многострочных текстов можно использовать тег <textarea>. Для этого элемента с помощью атрибутов rows и cols можно задавать размер элемента в строках и столбцах текста.

```
<textarea rows="5" cols="50" name="txtMsg" wrap="virtual">
</textarea>
```

Еще одним часто употребляемым элементом формы является метка <label>. В отличие от уже рассмотренных элементов метка не передает данные на сервер, она является интерфейсным элементом формы, упрощающим работу со строками ввода, переключателями, списками… Элемент <label> устанавливает связь между определённой меткой, содержащей поясняющий текст-подсказку, и элементом формы (<input>, <select>, <textarea>). Такая связь позволяет устанавливать фокус на связанных элементах форм щелчком курсора мыши на текст метки. Кроме того, с помощью <label> можно устанавливать горячие клавиши на клавиатуре и переходить на активный элемент подобно ссылкам.

Метка связывается с элементом двумя способами:

- установкой атрибута for в значение стилевого идентификатора id связанного элемента управления:

```
<input id="txtFIO" type=”text”>
<label for="идентификатор">Введите фамилию, имя, отчество</label>
```

- заключение связаного элемента внутрь тега <label>

```
<label><input type="text" id="txtFIO" > Введите фамилию, имя, отчество </label>
```

Еще одним интерфейсным элементом для форм является тег <fieldset>, позволяющий группировать прочие элементы, обрамляя их единой рамкой.

Помимо уже упоминавшихся частных атрибутов, индивидуальных для конкретной тегов, существуют общие для всех управляющих элементов атрибуты:

- атрибут name используется для привязки данных, заданных в управляющем элементе с самим элементом в запросе к серверу;

- атрибут disabled позволяет запретить доступ к управляющему элемент;

- атрибут required определяет, является ли это поле обязательным (без его заполнения не будет разрешена отправка данных формы на сервер);

- атрибут autofocus устанавливает фокус ввода на этом элементе.

3.2. Порядок обработки данных формы

Отправка данных формы на сервер может осуществляться при нажатии кнопки класса submit или при нажатии клавиши ENTER в момент, когда фокус ввода в пределах формы.

При отправке формы клиент производит следующие действия [10]:

1. Кодирование содержимого элементов

У каждого органа управления (элемента) формы имеются атрибуты `name` и *`value`*, хранящие, соответственно, имя элемента и значение. Имя задается при вёрстке документа, значение определяет пользователь при (вводе) выборе в рамках управляющего элемента. Кодирование формы заключается в замене некоторых недопустимых символов последовательностями допустимых. Пробелы в значениях кодируются символом '+'. Символы с кодом больше или равным 127, а также некоторые символы с кодом меньше 127, например, '+', '%' и '&', кодируются тройкой символов '%hh', где hh – две шестнадцатеричные цифры кода символа. Код символа зависит от кодировки, в которой создан содержащий форму документ. Таким образом, для каждого элемента получается строка вида `name=value`, где `name` – имя элемента органа управления, `value` – строка с закодированным значением элемента органа управления.

2. Кодирование содержимого формы, которое осуществляется в соответствии с атрибутом `enctype` элемента FORM. В случае способа кодирования `application/x-www-form-urlencoded` полученные строки для каждого элемента соединяются друг с другом символом '&':

```
name1=value1&name2=value2&...&nameN=valueN
```

В случае способа кодирования `text/plain` строки разделяются символами перевода строки и возврата каретки:

```
name1=value1
name2=value2
 ...
nameN=valueN
```

3. Формирование HTTP-сообщения запроса

В случае использования HTTP-метода GET кодированное содержимое формы присоединяется к адресу запрашиваемого ресурса в начальной строке запроса при помощи символа '?':

```
URI?name1=value1&name2=value2&...&nameN=valueN
```

В случае использования метода POST кодированное содержимое формы помещается в тело сообщения. В сообщение запроса включается заголовок `Content-Type`, в котором содержится способ кодирования содержимого, и заголовок `Content-Length`, указывающий на длину кодированного содержимого.

4. Отправка сообщения серверу

Оформленный и закодированный запрос отправляется на обработку серверу.

Например, если мы заполним данными поля такой формы:

```
<form method='GET' action="register.php">
   <input type='text' name='fio'><br>
   <input type='text' name='addr'><br>
   <select name="level">
    <option value="1">Новичок</option>
   <option  value="2">Специалист</option>
    <option value="3">Эксперт</option>
   </select>
  <input type='submit' value="Отправить">
</form>
```

то на сервер будет отправлен следующий запрос:

```
http://mysite.com/register.php?fio=Cuper&addr=12Street&level=1
```

3.3. Валидация форм

Перед передачей информации на сервер требуется проверить полноту и корректность заполнения полей формы. Для контроля полноты заполнения можно использовать атрибут `required`, объявляющий поле обязательным для заполнения. Если обязательное поле пустое, браузер выведет сообщение об ошибке, а форма не будет отправлена на сервер. Данный атрибут доступен начиная с версии стандарта HTML5, если используется стандарт более ранних версий, то придется осуществлять контроль заполнения элемента программно в обработчике события формы `onsubmit`. Например, для элемента

```
<input type='text' name='fio' id=”fio”>
```

Это можно сделать фрагментом javascript-кода:

```
 var iFIO=$("#fio");
var fio=iFIO.val();

if(fio.trim()==’’)

   {

      iFIO.css("border-color", "red");
      return false;
   }
```

Факт заполнения элемента некоторым значением еще не означает, что его значение может быть отправлено на сервер – если задано некорректное значение, то желательно отменить передачу, чтобы предотвратить лишние сеансы взаимодействия клиента и сервера и обработку сервером заведомо неприемлемых данных. Контроль значений для некоторых типов полей можно возложить на браузер, который должен следовать рекомендациям HTML-стандарта при анализе введенных данных. Так, например, если задано поле для ввода `e-mail`:

```
<input size='20' id="mail" name="mail" placeholder="e-mail"
type="email"/>
```

то перед отправкой браузер проверит поле на соответствие правилам написания электронных адресов и отменит передачу с выводом предупреж-

дающего сообщения (см. рис.2), если введенный адрес не соответствует принятому шаблону.

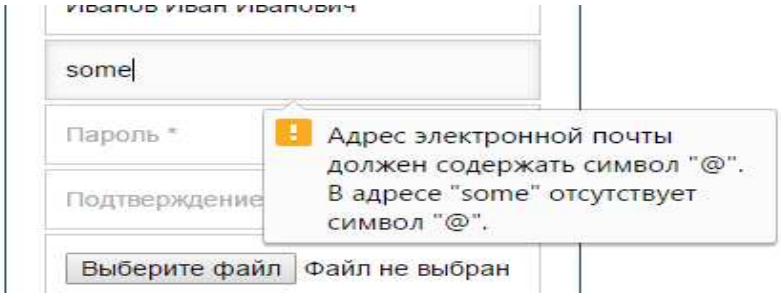

Рисунок 2. Автоматический контроль браузером корректности ввода электронного адреса

В стандарте [1] для контроля корректности ввода электронного адреса задан следующий шаблон:

```
/^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-
]+)*$/
```

Аналогично будет контролироваться корректность ввода адреса url в таком элементе:

```
<input size='20' id="url" name="url" placeholder="Введите
URL" type="url"/>
```

В это поле нельзя будет ввести некорректный URL-адрес: обязательно должен начинаться с названия протокола (http://, ftp://, https:// и т.п.) и содержать доменное имя после названия протокола.

Если для поля формы необходимо определить уникальный шаблон, которому должен соответствовать вводимый текст, то, начиная с версии HTML5, можно воспользоваться атрибутом pattern, который в синтаксисе регулярного выражения оговаривает, как должно выглядеть корректное значение для данного элемента:

```
    <label>
  <input type="text" name="postCode" pattern="[0-9]{6}"
    id=”postCode”>
    Почтовый код: </label>
```

В приведенном примере для текстового поля разрешен ввод строки, содержащей ровно 6 цифр. Если документ верстается для более раннего HTML-стандарта, то анализ на соответствие шаблону необходимо реализовывать программно в обработчике события `onsubmit`. Это событие играет важную роль в процессе функционирования формы. Сопоставленный этому событию обработчик будет вызываться всякий раз, когда пользователь инициирует процесс отправки формы на сервер. Обработчик этого события может проверить данные формы на валидность и, если они неверны, то вывести ошибку, пометить некорректные элементы специальными стилями, вызвать метод `event.preventDefault()` для отмены отправки на сервер. Отменить отправку можно также просто вернув логическую ложь как результат обработки данных формы:

```
<form  method="post" id="frmReg" enctype="multipart/form-data"
action='someScript.php"' onsubmit="return check_form();">
   <p>Форма регистрации пользователей</p>
<input id="login" name="login" type="text" autofocus re-
quired/>
<input id="pass" name="pass" type="password" required/>
<input id="pass_confirm" name="pass_confirm" type="password"
                                                    required/>
        <input type="submit" value="Зарегистрироваться" />
</form>
<script>
function check_form()
{ var bValid=true;
  //Проверка подтверждения пароля
  var pass=$("#pass").val();
  var iConfPas=$("#pass_confirm");
  var cpass=iConfPas.val();
  if(cpass!=pass) {
    iConfPas.css("border-color", "red");
    bValid=false;
   }
   else
    iConfPas.css("border-color","#ccc");
   //Проверка других полей
    return bValid;
}
</script>
```

Как уже было сказано выше, основное назначение обработки события `onsubmit` – проверка корректности заполнения пользователем полей формы. Универсальным и одним из самых востребованных способов выполнить подобную проверку является использование *регулярных выражений*. Регулярные выражения – мощный и гибкий инструмент работы со строковыми данными, позволяющий осуществлять такие операции, как поиск и замена подстрок, проверка соответствия строки предопределенному шаблону, выделение лексем в строке. Из всего разнообразия инструментов поддержки регулярных выражений в javascript рассмотрим те, которые позволят решить поставленную задачу – проверить корректность заполнения элементов HTML-форм. Более подробную информацию о регулярных выражениях можно посмотреть, например, в [2]

Серверная сторона web-приложения получает информацию из формы html-документа в виде http-запроса get или post. Если обрабатывает запрос php-скрипт, то данные запроса будут упакованы в элементы предопределенных суперглобальных массивов. Эти массивы называются суперглобальными поскольку обращение к ним возможно из любой точки и не требует использования ключевого слова *global*, если доступ осуществляется из функции или метода. К суперглобальным массивам php относятся:

$_SERVER Содержит все данные о настройках среды выполнения скрипта и параметры сервера

$GLOBALS Содержит ссылки на все переменные, объявленные в данном скрипте. Это ассоциативный массив, в котором имена переменных являются ключами

$_GET Перечень параметров get- или post-запроса клиента

$_POST

$_SESSION Переменные открытой сессии

$_COOKIE Содержит все cookies, которые сервер установил на стороне пользователя

$_FILES Содержит список файлов, загруженных на сервер из формы

Суперглобальный массив $_SERVER содержит ряд ассоциативных элементов описывающих состояние клиента на момент отправки запроса и состояние сервера на момент его обработки:

'DOCUMENT_ROOT'	Содержит путь к корневому каталогу сервера
'HTTP_USER_AGENT'	содержит информацию о типе и версии браузера и операционной системы посетителя
'REMOTE_ADDR'	IP-адрес клиента
'HTTP_REFERER'	Хранит адрес страницы, с которой посетитель пришел на данную страницу
'SCRIPT_FILENAME'	Хранит абсолютный путь к файлу от корня диска
'QUERY_STRING'	Хранит запрос клиента (часть адреса после ?)
'REQUEST_METHOD'	Хранит метод запроса, который применяется для вызова скрипта: GET или POST
'HTTP_HOST'	Содержит имя сервера, которое, как правило, совпадает с доменным именем сайта, расположенного на сервере
'HTTP_ACCEPT'	Хранит содержимое поля Accept из HTTP-заголовка запроса – предпочтения клиента относительно типа документа

Например, следующий фрагмент позволяет определить тип используемого клиентом браузера:

```
if (strpos($_SERVER["HTTP_USER_AGENT", "Firefox") !== false)
```

```
      $browser = "Firefox";
elseif (strpos($_SERVER["HTTP_USER_AGENT", "Opera") !== false)
      $browser = "Opera";
elseif (strpos($_SERVER["HTTP_USER_AGENT", "Chrome") !== false)
      $browser = "Chrome";
elseif /*Проверка других браузеров*/
```

Для получения данных от форм клиентской стороны Web-приложения чаще всего используются массивы $_GET и $_POST. Они представляют собой ассоциативные массивы, в которых индексами элементов являются имена параметров запроса, а значениями – соответственно, значения параметров запроса. Если сервер получит запрос

```
http://mysite.com/register.php?login=admin&pass=123&remember=1
```

то в скрипте register.php массив $_GET будет инициализирован следующим образом:

```
Array ( [login] =>admin [pass] => 123  [remember] => 1 )
```

Соответственно, анализируя элементы массивов $_GET или $_POST , можно контролировать корректность переданных данных:

```
if(isset($_GET[“login”]) && check_login($_GET[‘login’]))
   $login=$_GET[“login”];
else $login=”anonymous”;
```

Для проверки корректности данных в запросе в простейших случаях можно использовать встроенных методы PHP:

```
   is_numeric(), is_integer(), is_real(), is_string(),
is_bool()
```

Данные методы возвращают true, если значение параметра относится к указанному типу и false в противном случае;

```
 if(isset($_GET[“age”]) && is_integer($_GET[“age”]))
   $age=$_GET[“age”];
else die(“некорректно задано поле ‘возраст’”);
```

Однако, как уже было отмечено для клиентской стороны, зачастую недостаточно просто контролировать тип параметра, необходимо, чтобы он принимал набор предопределенных значений или соответствовал неко-

торому шаблону. В этом случае можно использовать механизм проверки на основе регулярных выражений.

Для проверки соответствия строки (или ее подстроки) регулярному выражения в PHP можно использовать функцию

```
mixed preg_match ( string pattern, string subject [, array
&matches])
```

Ищет в заданном тексте *subject* совпадения с шаблоном *pattern.*

В случае, если дополнительный параметр *matches* указан, он будет заполнен результатами поиска. Элемент *$matches[0]* будет содержать часть строки, соответствующую вхождению всего шаблона, *$matches[1]* – часть строки, соответствующую первой подмаске, также для *$matches[2], $matches[3]* и так далее.

Проверить корректность переданного логина пользователя можно следующим образом:

```
$valid= preg_match("/^[a-zA-Z0-9_]{1,10}$/u", $logstr);
```

Проверит фамилию, имя и отчество в одной строке можно следующим вызовом:

```
preg_match("/^[А-ЯЁ][а-яё]+ [А-ЯЁ][а-яё]+ [А-ЯЁ][а-
яё]+$/u",$str);
```

Если требуется замена подстрок, то можно использовать PHP-метод preg_replace:

```
mixed preg_replace ( mixed pattern, mixed replacement, mixed
subject [, int limit] )
```

Выполняет поиск в строке *subject* совпадений с шаблоном *pattern* и заменяет их на *replacement*. В случае, если параметр *limit* указан, будет произведена замена *limit* вхождений шаблона; в случае, если *limit* опущен либо равняется -1, будут заменены все вхождения шаблона.

Вот как можно в строке заменить все вхождения символа ‘\n’ на тег
:

```
$dicContent= preg_replace("/(\n)/", "<br>", $divContent);
```

4. Контроль состояния Web-приложений

Web-приложения используют в качестве транспортной основы протокол HTTP со всеми его достоинствами и недостатками. К последним можно отнести отсутствие поддержки состояния приложения – как завершился предыдущий запрос, какие значения переменных были установлены и т.п.

В рамках протокола HTTP подобная проблема решается с использованием механизма cookie, PHP усовершенствовал этот механизм до технологии сессий (и не только PHP – сессии поддерживает, например, и ASP.NET)

4.1. Сохранение состояния Web-приложения в cookies браузера

Cookies – это механизм хранения данных браузером удаленной машины для отслеживания или идентификации возвращающихся на сайт посетителей. Данный механизм предполагает целый ряд приложений от хранения индивидуальных настроек и предпочтений пользователя при работе с Web-приложением до аутентификации пользователей на Web-ресурсе. Позволяет этот механизм и сохранять состояние Web-приложения [9].

Создать cookie на серверной стороне можно с использованием функции *setcookie*() :

```
setcookie($name [, $value [, $expire [, $path [, $domain [, $secure]]]]])
```

где $name — имя cookie;

$value — значение, хранящееся в cookie с именем *name*;

$expire — время, по истечении этого времени cookie удаляется с машины клиента;

$path — путь, по которому доступен cookie;

$domain — домен, из которого доступен cookie; $*secure* — директива, определяющая, доступен ли файл cookie по защищенному протоколу HTPPS.

Альтернативный способ установить куки – сформировать заголовок HTTP-ответа с использованием функции header:

```
header("Set-Cookie: name=value;”.
“expires=Thu, 1-Oct-2013 11:00:04 GMT;”.
"path=/; domain=www.domain.com“);
```

Пример определения куки:

```
setcookie(“username", $username, time()+3600*24*31);
```

Установленный cookie добавляется в заголовок HTTP-ответа сервера (поэтому их необходимо устанавливать до начала формирования тела ответа). При повторном обращении браузера к серверу в запросе в том числе передаются и куки, установленные сервером. В PHP-скрипте они будут доступны через суперглобальный массив $_COOKIE

```
if(isset($_COOKIE[“username”]))
  $username=$_COOKIE[“username”];
else
   $username=“anonymous”;
```

Таким образом, установив значение куки на серверной стороне Web-приложения и передав ее клиенту, при повторном доступе из того же браузера на сайт можно получить все установленные сервером куки и таким образом определить, что происходило при обработке предыдущих запросов (имя аутентифицированного пользователя, страница, которую последней посетил пользователь, выбранные пользователем для отображения элементы Web-приложения и т.п.).

Если необходимо принудительно удалить ранее установленный cookie, достаточно присвоить ему значение в виде пустой строки:

```
setcookie(“username", “”);
```

4.2. Сохранение состояние Web-приложение с использованием сессий

Механизм сессий схож с cookie’s – он также позволяет сохранять состояние Web-приложения в виде набора пар “ключ=значение”, но хранятся они в текстовом файле на стороне сервера.

Каждая сессия имеет уникальный идентификатор, по которому скрипт получает доступ к данным сессии. Идентификатор сессии:

1. Сохраняется в куках (если включены)
2. Передается в скрипт через строку запроса (если в браузере запрещены куки)

Начало работы с сессией – вызов функции session_start().

Эта функция:

- Устанавливает cookie с идентификатором сессии;
- Создаст временный файл, который будет хранить данные сессии.

В файле php.ini можно установить директиву

session.auto_start = 1

чтобы сессия стартовала для скрипта автоматически

После создания сессии в скрипте доступен суперглобальный массив $_SESSION, в котором хранятся все данные текущей сессии. Вот так, например, можно зафиксировать в сессии имя успешно аутентифицированного пользователя:

```
session_start();
//Проверяем правильность аутентификационных данных
if(checkUser($_GET[“login”], $_GET[“passwd”]))
{
   $_SESSION[“logUser”]=$_GET[“login”];
}
```

Теперь во всех скриптах Web-приложения можно проверять, установлена ли эта сессионная переменная. Если установлена, то скрипт может предоставить дополнительный функционал, отобразить предназначенный только для зарегистрированных пользователей контент:

```
if(isset($_SESSION[“logUser”]))
{ //код только для зарегистрированных пользователей
}
```

Когда сессия больше не нужна, ее можно уничтожить методом destroy_session().

Для уничтожения всех переменных сессии вызываем метод unset(). Этот же метод позволяет удалить конкретную переменную сессии:

```
unset($_SESSION["имя_переменной"])
```

Например, в скрипте, отвечающем за отмену сеанса работы пользователя в Web-приложения, можно отменить переменную “logUser”:

```
session_start();
if(isset($_SESSION["logUser"]))
  unset($_SESSION["logUser"]);
```

5. Взаимодействие Web-приложений с хранилищами данных

Взаимодействие с серверами баз данных для хранения и извлечения данных при обработке запросов клиентов и динамическом формировании страниц ресурса стало стандартом при проектировании и разработке Web-приложений. В связи с этим современные языки Web-разработки включают инструментарий для работы с базами данных. Не стал исключением и язык PHP, который поддерживает целый ряд расширений, предназначенный для взаимодействия с базами данных различных производителей. Следующий фрагмент файла php.ini для установленного под операционной системой Windows интерпретатора php иллюстрирует, что язык поддерживает ряд расширений для различных баз данных: MySQL, MS SQL, Oracle (oci), SQLLite, FireBird, PostgreSQL:

```
;extension=php_mssql.dll
;extension=php_mysql.dll
extension=php_mysqli.dll
;extension=php_oci8.dll
extension=php_pdo.dll
```

```
;extension=php_pdo_firebird.dll
;extension=php_pdo_mssql.dll
extension=php_pdo_mysql.dll
;extension=php_pdo_oci.dll
;extension=php_pdo_oci8.dll
;extension=php_pdo_odbc.dll
;extension=php_pdo_pgsql.dll
;extension=php_pdo_sqlite.dll
;extension=php_pgsql.dll
;extension=php_sqlite.dll
```

Некоторые из этих расширений, как можно заметить, раскомментированы, что означает включение данного расширения при запуске интерпретатора. Сами расширения (для ОС Windows) реализованы в виде динамических библиотек и хранятся в папке, прописанной в директиве *extension_dir* файла php.ini:

```
extension_dir = "/usr/local/php5/ext"
```

Если требуется включить то или иное расширение, необходимо раскомментировать строку, соответствующую данному расширению, в php.ini, если вы работаете в ОС Windows, и пересобрать PHP со специальными опциями для ОС Unix. Подробнее об установке и настройке PHP с поддержкой различных расширений для работы с базами данных можно посмотреть на этом ресурсе [8].

Рассмотрим вопросы взаимодействия Web-приложения и базы данных будут на примере СУБД MySQL как одной из самых популярных систем своего класса среди Web-разработчиков вообще и начинающих разработчиков в частности [11]. Как можно заметить выше, для работы с MySQL PHP поддерживает три расширения: *mysql, mysqli* и *PDO*. Расширение *mysql* считается устаревшим, начиная с версии PHP 5.5.0, и будет удалено в дальнейшем, поэтому его использование нецелесообразно. С двумя другими расширениями мы познакомимся поближе.

5.1. Доступ к базам данных MySQL с использованием расширения mysqli

Расширение *mysqli* пришло на смену mysql и, сохранив привычный php-разработчикам интерфейс, предоставляет модернизированный API, который поддерживает возможности современных версий СУБД и обеспечивает высокую степень безопасности при работе с базами данных, в связи с чем его назвали *улучшенным (improved)* MySQL. Расширение *mysqli* включается в поставку PHP версий 5 и выше.

К преимуществам *mysqli* можно отнести:

- Поддержку как процедурного, так и объектно-ориентированного интерфейсов;
- Поддержку подготавливаемых запросов;
- Поддержку транзакций;
- Улучшенные возможности отладки;
- Поддержку мультизапросов.

Как уже было отмечено, при знакомстве с расширением *mysqli* необходимо рассматривать два возможных подхода: процедурный и объектно-ориентированный.

Соединение с базой данных в процедурном стиле осуществляется с использованием метода *mysqli*_connect:

```
mysqli mysqli_connect ([ string $host =
ini_get("mysqli.default_host")
[, string $username = ini_get("mysqli.default_user")
[, string $passwd = ini_get("mysqli.default_pw")
[, string $dbname = ""
[, int $port = ini_get("mysqli.default_port")
[, string $socket = ini_get("mysqli.default_socket") ]]]]]] );
```

где *$host* – имя или IP-адрес сервера базы данных, *$username* и *$passwd* задают аутентификационные параметры подключения к серверу, *$dbname* – имя базы данных, используемой по умолчанию, *$port* – порт, на котором работает

сервер базы данных, *$socket* – сокет, используемый для подключения к серверу.

Объектный подход использует аналогичные параметры для конструктора класса *mysqli*, который устанавливает соединение с базой.

Пример установки соединения с базой данных для процедурного стиля:

```
$link = mysqli_connect( 'local-
host', ‘user', 'passwd', ‘dbName');
if ($link==null)
die(“Не могу соединиться с базой данных. Ошибка: ”.
mysqli_cnnect_error());
```

В объектном стиле создается объект класса *mysqli*:

```
$mysqli = new mysqli('localhost', 'user', 'passwd', 'db');
 if ($mysqli->connect_error)
   die(‘Ошибка соединения'.$mysqli->connect_error);
```

Результат, возвращаемый методом установки соединения, в дальнейшем будет использован для работы с базой данных, он должен быть сохранен в переменной или поле класса.

В случае возникновения ошибки при обращении к серверу для получения сведений о ней всегда можно воспользоваться методом *mysqli_error*() в процедурном стиле и свойством *error* объекта класса *mysqli*. При подключении описание ошибки можно получить с помощью метода (свойства) *mysqli_connect_error* (*mysqli::connect_error*). Код возникшей ошибки можно получить из метода (свойства0 *mysqli_errno* (*mysqli::errno*).

После установки соединения с базой данных можно выполнять запросы на чтение, изменение, удаление данных. Здесь опять возникает разветвление повествования, поскольку запросы можно подавать в классической форме и в форме подготовленных запросов.

Классический стиль предполагает непосредственную передачу всех данных запроса на сервер. Выполнить подобный запрос можно с использованием метода *mysqli_query* в процедурном стиле или метода query в объектном стиле.

```
 mixed mysqli_query ( mysqli $link , string $query [, int $resultmode
= MYSQLI_STORE_RESULT ] );
 mixed mysqli::query ( string $query [, int $resultmode =
MYSQLI_STORE_RESULT ] );
```

Параметр *$query* задает текст передаваемого серверу запроса, а в параметре *$resultmode* передаются константы MYSQLI_STORE_RESULT или MYSQLI_STORE_RESULT. Для второй константы обязательно освобождение памяти, занятой результатами запроса методами *mysqli_free_result* (*mysqli::free, close, free_*result в объектном интерфейсе) перед подачей повторного запроса.

Обобщенная передача запроса в объектном стиле:

```
$result = $mysqli->query('текст запроса', MYSQLI_USE_RESULT);
```

Вызов метода *mysqli_query*(*mysqli::query*) можно заменить парой вызовов:

```
$mysqli->real_query('текст запроса');
$result = $mysqli->use_result();  // аналог вызова query с
     //константой MYSQLI_USE_RESULT
    // или
$mysqli->real_query('текст запроса');
$result = $mysqli->store_result();  // аналог вызова query с
 //константой MYSQLI_STORE_RESULT
```

Вот как выглядит реальный запрос на выборку данных из базы в классическом стиле:

```
   $result = mysqli_query($link, 'SELECT *  FROM books
```

```
   ORDER BY Title');
```

или в объектном подходе

```
$result = $mysli->mysqli_query( 'SELECT * FROM books
    ORDER BY Title');
```

Метод *mysqli_query* (или *query*) возвращает **FALSE** в случае неудачи. В случае успешного выполнения запроса на выборку данных *SELECT,* my*sqli_query* вернет объект класса mysqli_result. Для остальных успешных запросов (например, удаления или вставки данных) *mysqli_query* вернет **TRUE**.

Класс *mysql_result* – еще один базовый класс при работе с расширением mysqli. Для объектов этого класса можно вызывать методы для получения результатов выборки данных. В процедурном стиле этим целям служат специализированные методы, принимающие объект *$result* как параметр:

`mysqli::current_field` `mysqli_field_tell`	Получает смещение указателя по отношению к текущему полю
`mysqli::data_seek` `mysqli_data_seek`	Перемещает указатель результата на выбранную строку
`mysqli::fetch_all` `mysqli_fetch_all`	Выбирает все строки из результирующего набора и помещает их в ассоциативный массив, обычный массив или в оба
`mysqli::fetch_array` `mysqli_fetch_array`	Выбирает одну строку из результирующего набора и помещает ее в ассоциативный массив, обычный массив или в оба
`mysqli::fetch_assoc`	Извлекает результирующий ряд в виде

mysqli_fetch_assoc	ассоциативного массива
mysqli::fetch_field_direct mysqli_fetch_field_direct	Получение метаданных конкретного поля
mysqli::fetch_field mysqli_fetch_field	Возвращает следующее поле результирующего набора
mysqli::fetch_fields mysqli_fetch_field	Возвращает массив объектов, представляющих поля результирующего набора
mysqli::fetch_object mysqli_fetch_objec	Возвращает текущую строку результирующего набора в виде объекта
mysqli::fetch_row mysqli_fetch_row	Получение строки результирующей таблицы в виде массива
mysqli::field_seek mysqli_field_seek	Установить указатель поля на определенное смещение
free mysqli_free_result	Освобождает память занятую результатами запроса

Кроме перечисленных методов класс mysqli_result предоставляет ряд полей хранящие параметры результатов выборки.

mysqli::length mysqli_fetch_lengths	Возвращает длины полей текущей строки результирующего набора
mysqli::num_rows mysqli_num_rows	Получает число рядов в результирующей выборке
mysqli::field_count	Получение количества полей в результи-

mysqli_num_fields рующем наборе

Рассмотрим примеры выполнения запросов к базе данных с использованием расширения mysqli в классическом стиле. В процедурном стиле вызываем метод *mysqli_query*:

```
$query = "select * from books where idBook=$idBook";
$result = mysqli_query($link, $query);
 if (mysqli_num_rows($result) > 0)
  {  $bookInfo = mysqli_fetch_assoc($result);
      echo $bookInfo[“Title”];
  }
mysqli_free_result($result); // не забываем очищать
                             //результат выборки
```

Если в результате выборки получаем несколько записей, то их можно занести в смешанный массив (константа MYSQLI_BOTH) методом *mysqli_fetch_all*

```
if($result=mysqli_query($link, "select * from books"))
$arrBooks=mysqli_fetch_all($result, MYSQLI_BOTH );
 foreach($arrBooks as $book)
           echo “<li>”.$book[“title"].”</li>”;
mysqli_free_result($result);
```

В объектном стиле используется метод *query* класса *mysqli*. Предыдущий пример с выборкой информации обо всех книгах в объектном стиле реализуется следующим образом:

```
if($result=$mysqli->query("select * from books"))
 {
   $allBooks=$result->fetch_all(MYSQLI_BOTH );
   echo “table>”;
   foreach($allBooks as $book)
      echo “<tr><td>.$book[“Title"].”</td><td>”.
```

```
$book[“Author”].”</td></tr>”;
       echo “</table>”;
    $result->close(); //результат выборки также необходимо очищать
```

При выполнении запросов классическим методом при использовании в запросе данных из ненадежных источников (например, полученных из запроса клиента) необходимо учитывать возможные SQL-инъекции. Для защиты запроса необходимо экранировать спецсимволы методом *mysqli_real_escape_string* (*mysqli::real_escape_string*):

```
if(isset($_GET[“title”]))
 {
  $title=$mysqli->real_escape_string($_GET[“title”]);
  $result=$mysqli->query("select * from books where Title=$title");
     …
}
```

Запросами на выборку данных работа с базой данных не ограничивается, данные необходимо также добавлять удалять, изменять. Запросы этих типов также подаются с использованием методов *mysqli_query* (*mysqli::query*). Вот как можно удалить данные из базы:

```
    if ($mysqli->query(“delete from books where idBook=$idBook)"))
          echo “Запись удалена”;
    else
    echo “Не удалось удалить запись”;
```

Для добавления книги в таблицу используем тот же метод, но другой запрос:

```
    if($mysqli->query("insert into books (Title, idAuthor, pages) values ($title, $idAuth, $pages)"))
       echo “Запись удалена”;
    else
       echo “Не удалось добавить запись”;
```

После выполнения изменяющих базу данных запросов полезную информацию можно извлечь из свойств:

*mysqli::affected_ro*ws – количество затронутых строк предыдущим запросом не на выборку.

mysqli::insert_id – автоматически сгенерированный идентификатор для последнего запроса вставки.

Например, после выполнения запроса на удаление данных можно узнать, сколько записей было удалено:

```
if ($mysqli->query("delete from books where".
                                    " publishYear>=$year)"))
    echo "Удалено записей: ".$mysqli->affected_rows;
```

При выполнении операций вставки/изменения/удаления данных также необходимо заботиться о безопасности, используя метод экранирования спецсимволов. Для автоматизации решения вопросов обеспечения безопасности и даже отправки запросов к базе можно использовать интерфейс *подготовленных запросов*.

Если при работе в классическом стиле центральное место при обработке результатов выполнения запросов играют объекты класса *mysqli_result*, то при работе с подготовленными запросами приходится иметь дело с объектами класса *mysqli_stmt* (хотя имеется возможность сформировать на основе объекта класса *mysqli_stmt* объект класса *mysqli_result*). Создается объект *mysqli_stmt* методом *mysqli_stmt_init* (*mysqli::stmt_init*):

```
$stmt=$mysqli->stmt_init();
```

Подготовленный запрос выполняется в два этапа:

Первый этап: подготовка. На этом этапе на сервер БД посылается шаблон запроса. Сервер выполняет синтаксическую проверку этого шаблона, строит план выполнения запроса и выделяет под него ресурсы.

Второй этап: исполнение. Во время второго этапа клиент привязывает к псевдопеременным, задекларированным в шаблоне запроса на первом этапе, реальные значения и посылает их на сервер. Сервер, в свою очередь, подставляет их в шаблон и запускает уже готовый запрос на выполнение.

Этап подготовки выполняется с использованием метода prepare:

```
mixed mysqli_stmt::prepare ( string $query )
bool mysqli_stmt_prepare ( mysqli_stmt $stmt , string $query )
```

Пример вызова метода prepare:

```
$stmt = $mysqli->stmt_init();
$stmt->prepare ("SELECT * FROM `books` WHERE `idBook` = ?");
if ($stmt->errno)
   die('Ошибка запроса ' . $stmt->error);
```

Результат подготовительного этапа – объект типа mysqli_stmt, который в дальнейшем можно использовать для подачи запроса с различными наборами параметров. Принципиальная особенность запросов, которые передаются параметром *prepare* – использование placeholder'ов. Placeholder представляет собой параметр запроса, на место которого впоследствии будут подставлены реальные данные. Чтобы связать параметр запроса с переменными, из которых будут извлекаться данные для формирования реального запроса, который будет отправлен на сервер БД, используем метод *bind_param*:

```
bool mysqli_stmt::bind_param ( string $types , mixed &$var1
[,  mixed &$... ] )
bool mysqli_stmt::bind_param ( string $types , mixed &$var1 [,
mixed &$... ] )
```

Параметр *$types* задает типы привязываемых к параметрам запроса данных:

i	соответствующая переменная имеет тип integer

d	соответствующая переменная имеет тип double
s	соответствующая переменная имеет тип string
b	соответствующая переменная является большим двоичным объектом (blob) и будет пересылаться пакетами

Остальные параметры являются ссылками на переменные, значения которых будут использоваться для формирования запроса. Количество символов, определяющих типы данных, в параметре $types должно совпадать с количеством переменных в списке, а также количеству placeholder'ов в подготовленном запросе:

```
    $stmt->prepare   ("SELECT * FROM `books` WHERE  publishYear
> ? and pages>?");
    $stmt->bind_param("ii", $year, $pages);
```

После того как запрос подготовлен и к нему привязаны данные, можно выполнять запрос методом *ececute*:

```
    bool mysqli_stmt::execute ( void );
    bool mysqli_stmt_execute ( mysqli_stmt $stmt )

   $query ="INSERT INTO books (Title, Author, Pages, pun-
   blishYear)
       VALUES (?,?,?,?)";
   $stmt = $mysqli->prepare($query);
   $stmt->bind_param("ssii", $tit, $auth, $pages. $year);
   $tit = 'Муму';
   $auth = 'Тургенев И.С.';
   $pages = 100; $year=20010;
   /* выполняем запрос */
   $stmt->execute();
   $tit = 'Война и мир';
   $auth = 'Толстой Л.Н.';
   $pages = 800;$year=2013;
   /* выполняем запрос */
   $stmt->execute();
   /* закрываем запрос */
   $stmt->close();
```

Приведенный пример иллюстрирует достоинство подготовленных запросов – подготовив его однократно, можно многократно подавать запрос,

присваивая связанным переменным новые значения. При этом безопасность запросов в плане экранирования спецсимволов выполняется автоматически при вызове метода *execute*.

Если запрос предполагает выборку данных, то допускается привязка к переменным скрита результатов выборки. Выполнить подобную привязку можно вызовом метода *bind_result*:

```
bool mysqli_stmt_bind_result ( mysqli_stmt $stmt , mixed &$var1 [, mix
ed &$... ] );
```

```
bool mysqli_stmt::bind_result ( mixed &$var1 [, mixed &$... ] );
```

Эти функции привязывают столбцы выборки из базы данных к заданным в вызове метода переменным. Непосредственно помещает значения из столбцов выборки в переменные метод *mysqli_stsm*::*fetch*:

```
bool mysqli_stmt_fetch ( mysqli_stmt $stmt );
```

```
bool mysqli_stmt::fetch ( void );
```

Каждый новый вызов f*etch* будет извлекать и копировать в связанные переменные значения очередной строки выборки. Когда будут извлечены все строки, *fetch* вернет значение null. Тогда можно организовать следующий цикл обработки результатов запроса:

```
    if ($stmt = $mysqli->prepare("SELECT title, pages FROM books  OR-
DER BY id")) {
    $stmt->execute();
    /* привязка переменные к запросу */
    $stmt->bind_result($title, $pages);
    /* выборка данных */
    while ($stmt->fetch()) {
        printf("%s %d\n", $title, $pages);
    }
    /* закрываем запрос */
    $stmt->close();

    )
```

Альтернативный способ получить доступ к результатам выборки – получить их из объекта класса *mysqli_stmt* в виде объекта *mysqli_result* и в дальнейшем работать с данными аналогично классическому подходу. Для этих целей предназначен метод *get_result* класса *mysqli_stmt*:

```
$stmt = $mysqli->prepare("SELECT * FROM books");
if($stmt->execute())
 {$result = $stmt->get_result();
  $stmt->close();
  if($result)
  {$allBooks=$result->fetchAll(MYSQLI_ASSOC);
   foreach($allBooks as $book)
    echo "<p>".$book["Title"]."</p>";
  }
}
```

Рассмотренные средства расширения mysqli не исчерпывают описания всех его возможностей. С помощью этого расширения можно выполнять мультизапросы (когда в одном вызове отправляются сразу несколько запросов к базе данных). Эту возможность предоставляет метод *mysqli_multy_query* (*mysqli::multy_query*). С использованием метода *mysqli_ssl_set* (*mysqli::ssl_set*) можно обеспечить защиту соединения с базой данных по протоколу OpenSSL. Поддерживает *mysqli* и механизм транзакций, которые обслуживаются набором методов: *mysqli_begin_transaction* (*mysqli::begin_transaction*), *mysqli_commit* (*mysqli::commit*), *mysqli_rollb*ack (*mysqli::rollback*). Подробности этих аспектов работы с базами данных из PHP-скриптов можно узнать, например, в [2].

5.2. Взаимодействие с базами данных с использованием PDO

Расширение *mysqli*, как следует из названия, предназначено для работы с базами данных, управляемыми СУБД MySQL. Если по каким-то причинам тип СУБД изменится, то придется все вызовы-обращения к базе заменять (например, на методы с префиксом *mssql_** для СУБД MSSQL). Если подобная ситуация смены СУБД реальна, то разумным решением будет использовать подход с абстрактным уровнем доступа к СУБД, когда стандартный набор средств расширения позволяет обращаться к различным

типам баз данных. Одним из подобных решений для языка PHP является PDO (PHP Data Object). Это расширение поддерживается, начиная с версии 5.1 PHP. Принцип абстрагирования доступа к данным, реализуемый в PDO, означает, что вне зависимости от того, какая конкретная база данных используется, исполняется один и тот же набор методов для выполнения запросов и выборки данных

Само по себе расширение PDO не взаимодействует с базами данных напрямую. Для поддержки различных СУБД PDO поддерживает драйверы, которые транслируют PDO запросы в обращения к нативному API СУБД. PDO поддерживает драйвера для следующих СУБД:

```
PDO_FIREBIRD ( Firebird/Interbase 6 )
PDO_IBM ( IBM DB2 )
PDO_MYSQL ( MySQL 3.x/4.x/5.x )
PDO_ODBC ( ODBC v3 (IBM DB2, win32 ODBC)
PDO_PGSQL ( PostgreSQL )
PDO_SQLITE ( SQLite 3 and SQLite 2 )
PDO_SQLSRV ( Microsoft SQL Server )
…
```

Список всех поддерживаемых СУБД можно получить вызовом метода getAvailableDrivers класса PDO:

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

Соединение с базой данных устанавливается при создании объекта класса PDO. В конструкторе этого класса передается вся необходимая для установки соединения информация – DSN источника данных, имя и пароль пользователя, а также параметры инициализации соединения:

```
    try { $PDOparams = array(PDO::MYSQL_ATTR_INIT_COMMAND =>
'SET NAMES \'UTF8\'');
    $dbh = new PDO("mysql:host=" . $servername . ";dbname=" .
$dbname, $user, $pass, $PDOparams);
        }
```

```
    catch (PDOException $e) {
     die("Ошибка подключения к базе данных" .$e->getMessage());
}
```

Для отключения от базы данных достаточно уничтожить объект, связанный с этим соединением (*$dbh* в последнем примере), уничтожив все ссылки на него:

```
$dbh=null;
```

В простейшем варианте запрос к базе данных с использованием PDO можно сделать вызовом метода

```
public PDOStatement PDO::query ( string $statement );
```

В качестве параметра можно передать любой CRUD-запрос. Если запрос предполагал выборку данных, строки запроса можно перебирать с помощью цикла *foreach*:

```
foreach ($dbh->query("select * from books") as $row)
echo $row['title']."\t".$row['author']."\t".$row['pages']."\n";
```

PDO поддерживает подготовленные запросы, и в этом классе данный механизм стал еще более удобным и эффективным в использовании. В подготовленных запросах могут использоваться безымянные и именованные placeholder'ы. Схема работы с безымянными параметрами запроса аналогична уже рассмотренной для *mysqli*: сначала подготавливается запрос методом prepare с формированием объекта класса PDOStatement (аналог класса *mysqli_stm*t), затем осуществляется привязка переменных к параметрам запроса (в PDO реализуется методом *bindValue*), и выполняется запрос методом *execute*:

```
$query = "select * from books where idBook>?";
$stmt = $dbh->prepare($query);
$stmt->bindValue(1, $idBook, PDO::PARAM_INT);
$stmt->execute();
$books = $stmt->fetchAll();
```

Для привязки параметра запроса с переменными или значениями класс PDOStatement предлагает целый ряд методов:

```
public bool PDOStatement::bindValue ( mixed $parameter , mixed $value
[, int $data_type = PDO::PARAM_STR ] )
```

Метод связывает параметр запроса с конкретным значением *$value*. Значение *$parameter* для безымянного placeholder'а хранит его порядковый номер в запросе (начиная с 1). Здесь и в следующем методе параметр метода $*data_type* задает тип placeholder'а. Возможные значения:

PDO::PARAM_STR — представляет типы данных SQL CHAR, VARCHAR и другие строковые типы.

PDO::PARAM_BOOL — представляет булевый тип данных

PDO::PARAM_INT — представляет тип данных SQL INTEGER.

PDO::PARAM_LOB — представляет тип данных больших объектов SQL.

Другой тип привязки задает метод *bindParam*, который связывает псевдопеременную (placeholder) запроса с переменного скрипта.

```
public bool PDOStatement::bindParam ( mixed $parameter ,
mixed &$variable [, int $data_type = PDO::PARAM_STR ] );
```

Использование именованных псевдопеременных запросов представляется более удобным – здесь каждому placeholder'у дается идентификатор вида

```
:имя_параметра
```

Подобные идентификаторы впоследствии можно использовать при привязке переменных и формировании запроса. В частности, параметры рассмотренных выше методов *bindValue* и *bindParam $parameter* могут содержать не порядковый номер псевдопеременной запроса, а ее идентификатор:

```
$stmt->bindValue(':publishYear', $year, PDO::PARAM_INT);
$stmt->bindParam(':title', $bookTitle, PDO::PARAM_STR, 30);
```

Именованные псевдопеременные указываются в запросе при его подготовке по идентификатору. Это удобно, поскольку не требует отслеживать порядок следования placeholder'ов при работе с запросами.

```
    $query = "insert into houses (title, author, pages, pub-
lishYear) values (:title, :author, :pages, :year)";
    $stmt->bindParam(':pages', $bookPages, PDO::PARAM_INT);
    $stmt->bindParam(':year', $year, PDO::PARAM_INT);
    $stmt-
>bindParam(':author', $bookAuthor, PDO::PARAM_STR, 30);
    $stmt->bindParam(':title', $bookTitle, PDO::PARAM_STR, 30);
    $stmt->execute();
    $result = $stmt->fetchAll(PDO::FETCH_BOTH);
```

Извлечение результатов запроса из объекта PDOStatement схоже с уже рассмотренным для mysqli: у данного класса есть методы *fetch, fetchAll, fetchColumn, fetchObject* и др. Эти методы позволяют получать результаты выборки построчно в виде массивов или объектов, либо получить всю выборку одним массивом. Например, для метод

```
    public mixed PDOStatement::fetchObject ([ string $class_name =
"stdClass" [, array $ctor_args]] )
```

возвращает новый объект, имена свойств которого соответствуют именам перечисленного в запросе на выборку столбцов или FALSE в случае возникновения ошибки (например, когда все строки выборки были обработаны). Это позволяет организовать циклическую обработку результатов выборки данных:

```
 $stmt = $pdo->prepare('SELECT title, author FROM books WHERE
year=?');
   $stmt->execute(array($year));
   echo "<ul>";
   while($book=$stmt->fetchObject())
    echo "<li>Книга: ".$book->title." автора ".$book->author".
                                                    "</li>"
   echo "</ul>";
```

Последний пример иллюстрирует еще одну возможность – передачу значений для псевдопеременных подготовленного запроса (как именован-

ных, так и безымянных) непосредственно в параметрах метода execute в виде массива:

```
$query = "insert into houses  (title, author, pages) values
(:title, :author, :pages)";
$stmt = $dbh->prepare($query);
$params=array("title"=>$book["title"],"author"=>$book["author
"], "pages"=>$book["pages"]);
$status = $stmt->execute($params);
```

Результаты обработки запроса можно связать с переменными скрипта, чтобы быстрее получать значения из столбцов для каждой строки выборки. Для этих целей можно использовать метод

```
public bool PDOStatement::bindColumn ( mixed $column , mixed &$param
 [, int $data_type [, int $maxlen]] )
```

который привязывает столбец строки выборки с номером или названием *$column* к переменной *$param.*

При извлечении каждой строки из результатов выборки данные по столбцам будут автоматически переноситься в привязанные переменные:

```
try {
    $stmt = $dbh->prepare('SELECT  title,  author  FROM  books
WHERE year=?');
    $stmt->execute([2010]);
  $stmt->bindColumn(1, $title);//можно   связать   по   номеру
столбца
    $stmt->bindColumn("author", $author);//а можно и по назва-
нию
    echo "<ul>";
   while ($row = $stmt->fetch(PDO::FETCH_BOUND))
      echo        "<li>Книга:        "$title .        "автора
".$author."</li>";
    echo "</ul>";
  }
  catch (PDOException $e) {
    print $e->getMessage();
  }
```

Таким образом, можно сделать вывод, что рассмотренные расширения языка PHP для работы с базами данных многофункциональны, гибко настраиваемы, просты в использовании и во многом взаимозаменяемы.

Выбор в пользу того или иного расширения разработчик может сделать по таким соображениям, как возможность смены типа используемого СУБД, установленная версия PHP, имеющиеся наработки в рамках того или иного расширения.

6. Асинхронный обмен данными между клиентской и серверной частями Web-приложения

Традиционный обмен клиента и сервера предполагает, что браузер взаимодействует с пользователем, передает от его имени запросы к серверу, который реализует как бизнес-логику Web-приложения, так и логику представления данных, формирует html-страницу и возвращает ее браузеру. Такой подход порождает ряд проблем:

- вся трудоемкость выполнения Web-приложения перенесена на серверную сторону, что при большом количестве запросов от клиентов может сильно нагружать Web-сервер, в то время как возможности современных браузеров позволяют перенести существенную часть функционала Web-приложения на клиентскую сторону;

- трафик между клиентской и серверной стороной Web-приложения становится избыточным – вместо готового html-представления с его тегами и стилями сервер может возвращать лишь данные, на основе которых клиент сам может сформировать внешний вид Web-страницы;

- каждый запрос к серверу приводит к перерисовке страницы в браузере, что выглядит не очень эстетично – во многих случаях по результатам обработки запроса достаточно изменить (перерисовать) лишь небольшой фрагмент страницы;

- стандартными Web-запросами невозможно реализовать некоторые, уже ставшие привычными функции Web-приложений, как, например, автозаполнение полей форм при вводе данных.

6.1. Обмен данными по технологии Ajax

Решением всех этих проблем стал асинхронный обмен данными между клиентской и серверной стороной. Технология, позволяющая осуществлять подобный обмен, получила название **AJAX** (**A**synchronous **J**avascript **A**nd **X**ml») – загрузки данных с сервера без перезагрузки. Сам термин AJAX вошел в обиход Web-разработчиков в 2005-м году, хотя инструментарий, позволявший создавать асинхронные Web-приложения, были добавлены в браузере еще в середине 90-х годов прошлого века.

AJAX не является технологией в полном смысле этого слова, а объединяет в себе целый ряд методологий [10]:

- стандартизированное представление Web-страниц с помощью языков HTML и CSS;
- динамическое отображение и взаимодействие с пользователем с помощью DOM API и DHTML;
- обмен и обработка данных в виде XML и XSLT или JSON;
- асинхронные запросы с помощью объекта XMLHttpRequest.

Основу функциональности технологии AJAX обеспечивает объект XMLHttpRequest. Его назначение – передавать серверу HTTP- или HTTPS-запросы и обеспечивать прием и обработку ответов сервера.

Функциональность этого объекта доступна благодаря набору свойств и методов. Свойства объекта XMLHttpRequest:

`onreadystatechange`	Событие изменения состояния готовности сервера.
`readyState`	Позволяет узнать состояние готовности сервера.
`responseText`	Хранит ответ сервера как строку символов.
`responseXML`	Хранит ответ сервера как XML файл.
`status`	Хранит код ответа сервера.

Методы объекта XMLHttpRequest:

`open(тип_запроса, адрес, асинх [,польз, пароль])`	Создает запрос, определяет его параметры *тип_запроса*: GET или POST *адрес* – адрес ресурса *асинх :* true - запрос асинхронный, false - синхронный
`send(запрос)`	Позволяет передать запрос на сервер
`setRequestHeader( заголовок,значение)`	Позволяет добавить HTTP заголовок к запросу
`abort()`	Отменяет запрос
`getResponseHeader(headerName)`	Получить значение указанного поля ответа сервера

Отправка AJAX-запроса клиентской частью приложения осуществляется в следующей последовательности:

1. Создается экземпляр объекта XMLHttpRequest;
2. Для этого объекта определяется функция-обработчик события *onreadystatechange*. Это событие наступает при каждой смене состояния объекта XMLHttpRequest. Данная функция занимает одно из центральных мест в процессе обмена, поскольку именно в ней осуществляется обработка ответа сервера;
3. Открывается соединение с сервером, которое параметризуется указанием типа запроса (GET или POST), URL серверной части, флага асинхронного режима и имени и пароля пользователя (если необходимо);
4. Запрос отправляется серверу.

В коде на языке javascript эта последовательность выглядит следующим образом:

```
var xhttp;
//создаем объект
 if (window.XMLHttpRequest)
   { xhttp=new XMLHttpRequest(); }
else {//IE6
   xhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
  Xhttp.timeout=5000;
  xhttp.open('GET', 'ajaxService.php', false);
  xhttp.send();
//Точка останова клиентского скрипта
```

Приведенный пример демонстрирует синхронный вариант AJAX-запроса (третий параметр метода open равен false), когда метод send останавливает выполнение клиентского скрипта до момента возврата ответа сервера от сервера. Ответ сервера может быть получен из свойств объекта XMLHttpRequest *status, respnseText, responseXML.*

```
if(xhttp.status==200)
  document.getElementById("some").
            innerHTML=xhttp.responseText;
```

Свойство *status* хранит статус завершения из заголовка HTTP-ответа сервера. После того, как сервер пришлет ответ, его можно прочитать в свойстве *responseText* или *responseXML* (в зависимости от формата возвращаемых данных). В рассмотренном выше фрагменте этот ответ напрямую заносится в тег со стилевым идентификатором *some*.

При асинхронном запросе клиент не останавливает работу после направления запроса сервера. Момент прихода ответа сервера фиксируется в функции-обработчике события *onreadystatechange*:

```
<html>
<body>
<div id="some"><h2>Изменяемый блок</h2></div>
<button type="button" onClick="ajaxReq()">Загрузить данные</button>
</body>
<script type="text/javascript">
function ajaxReq()
{var xhttp = new XMLHttpRequest();
xhttp.open('GET', 'ajaxService.php', true);
xhttp.send();
xhttp.onreadystatechange = function() {
 if (xhttp.readyState != 4) return;
    if (xhttp.status != 200)
     document.getElementById("some").innerHTML=
          xhttp.status+ ': ' + xhttp.statusText;
else document.getElementById("some").innerHTML=
                              xhttp.responseText;
                                      }
}
</script></html>
```

Событие *onreadystatechange* активизируется (а, соответственно, и вызывается функция-обработчик этого события) неоднократно при обработке AJAX запроса. В связи с чем вызван обработчик этого события можно узнать из свойства *readyState.* Это свойство может принимать значения:

UNSENT (0)	Начальное состояние, объект создан

OPENED (1)	Вызван метод open
HEADERS_RECEIVED (2)	Получен заголовок HTTP-ответа
LOADING (3)	Загружается тело ответа (пришел очередной пакет)
DONE (4)	Ответ полностью получен или зарегистрирована ошибка

Таким образом, функция-обработчик события *onreadystatechange* вызывается как на стадии инициализации запроса, так и при его отправке, получении заголовка, всего ответа, возникновении ошибки при обработке запроса, что дает возможность гибко конфигурировать приложение, реагировать на различные стадии и статусы его обработки. Использование этого свойства для современных реализаций можно заменить на обработку специфичных событий. Если в ранних версиях реализации объекта XMLHttpRequest было определено лишь одно событие для контроля и обработки различных стадий обработки запроса (*onreadystatechange*), то в современной спецификации таких событий определено несколько, и они стали специфичны тому или иному этапу обработки запроса:

`loadstart` запрос начат.

`progress` браузер получил очередной пакет данных, можно прочитать текущие полученные данные в responseText.

`abort` запрос был отменён вызовом abort().

`error` произошла ошибка.

`load` запрос был успешно (без ошибок) завершён.

`timeout`	запрос был прекращён по таймауту.
`loadend`	запрос был завершён (успешно или неуспешно)

Шаблон использования объекта XMLHttpRequest в этом случае будет таким:

```
function ajaxReq()
{
  var xhttp = new XMLHttpRequest();
  xhttp.open('GET', 'ajaxService.php', true);
  xhttp.setRequestHeader('Content-Type', 'application/html;
  charset=utf-8');
  xhttp.timeout=10000;
  xhttp.onloadstart = function() {…}
  xhttp.onerror = function() {…}
  xhttp.onload = function() {…}
  xhttp.ontimeout = function() {…}
  xhttp.send();
}
```

Если с запросом необходимо передать параметры, то в случае GET-запроса они указываются в методе open:

```
xhttp.open('GET', 'ajaxSer-
vice.php?param1=value1&param2=value2', true);
```

Параметры POST-запроса необходимо передавать в теле запроса, поэтому они указываются в параметрах метода send:

```
xhttp.open('POST', 'ajaxService.php', true);
xhttp.send(“param1=value1&param2=value2”);
```

Обрабатывая запрос клиента, сервер формирует ответ, который может быть отправлен в различных форматах: html, json, xml. Серверный

скрипт, возвращающий ответ в html-формате может выглядеть следующим образом:

```
<?php
 require_once("db_func.php");
 $response = array('status'=>'no', 'message'=>'');
 header('Content-Type: application/html; charset=utf8');
 $db=connect_db();
 if($resStr!="ok")
    print(”Нет соединения с базой данных”);
 else
     {
       $res=getData($_GET[“param1”], $_GET[“param2”]);
       if($res==null)
           print("Не удалось извлечь данные");
       else
           {
print(“<h1>Данные о товаре</h1>”);
print(“<p>Наименование:
<span>{$res[“goodName”]}</span></p>”);
print(“<p>Стоимость:
<span>{$res[“goodPrice”]}</span></p>”);
           }
     }
?>
```

6.2. Форматы данных при асинхронном обмене

В приведенном примере серверный скрипт запрашивает данные о товаре, формирует для них html-представление и возвращает его клиенту, который может напрямую добавить их в некоторый элемент разметки. Однако современные тенденции разработки Web-приложений отводят серверной части лишь реализацию части бизнес-логики, которая извлекает и об-

рабатывает данные, а затем возвращает их клиенту. Формирование представления на основе результатов обработки запроса ложится на клиентский скрипт. Формат html при таком подходе не совсем подходит, так как с его помощью формируется готовое представление. Для передачи непосредственно данных можно использовать форматы json или xml.

Формат JSON (JavaScript Object Notation) позволяет легко сериализовать (преобразовать в строку) и десериализовать программные объекты для их передачи в запросах и ответах в виде текстовых строк.

Например, объект:

```
var subject = {
  fio: "Иванов И.И.", age: 30,
  address:
   { city: "Волжский", street:"Мира", house: 22}
};
```

методом JSON. stringify(subject) сериализуется в строку:

```
'{"fio":"Иванов И.И.", "age":30, "address":
{"city":«Волжский","street":"Мира","house":22}}'
```

из которой впоследствии легко десериализуется. Причем оба преобразования (объект->строки и строка->объект) легко реализуются как средствами javascript, так и языками серверных приложений (php, perl, c#, тот же javascript).

Если серверный скрипт реализован на языке php, для сериализации можно использовать метод json_encode:

```
<?
  $obj=getSubjectInfo(); //формируем объект
  echo json_encode($obj); //сериализуем его в строку
?>
```

Тогда на клиенте в обработчике события *onload* свойство *resnonseText* будет содержать сериализованный объект, на основе значений полей которого клиент может изменять представление фрагмента страницы:

```
var xhttp = new XMLHttpRequest();
xhttp.open('GET', 'ajaxService.php', true);
xhttp.setRequestHeader('Content-Type', 'application/json;
                                        charset=utf-8');
xhttp.timeout=10000;
xhttp.onload = function() {
   if (xhttp.status == 200)
      { var obj=JSON.parse(xhttp.responseText);
        var el=document.createElement("div");
        hEl=document.createElement("h1")'
        hEl.innerHTML="Информация о клиенте";
        el.appendChild(hEl);
        pName=document.createElement("p")'
        pName.innerHTML="ФИО: ";
        spName= document.createElement("span");
        spName.innerHTML=obj.fio;
        spName.style.color="red";
        pName.appendElement(spName);
        el.appendChild(pName);
        pAge=document.createElement("p")'
        pAge.innerHTML="возраст: ";
        spAge= document.createElement("span");
        spAge.innerHTML=obj.age;
        spAge.style.color="blue";
        pAge.appendElement(spAge);
        el.appendChild(pAge);
        var userList=document.getElementById("userList");
```

```
            userList.appendChild(el);
        }
    }
    xhttp.send();
```

Как следует из приведенного примера, после десериализации объекта методом JSON.parse его можно использовать как обычный программный объект javascript.

JSON-формат можно использовать и для передачи данных от клиента серверу. В этом случае на стороне клиента программный объект необходимо сериализовать методом JSON.stringify, а на серверной стороне (в случае использования языка php) десериализовать методом *json_decode*:

На клиенте:

```
var xhttp = new XMLHttpRequest();
xhttp.open('POST', 'ajaxServiceJSON.php', true);
xhttp.setRequestHeader("ContentType",
                "application/x-www-form-urlencoded");
xhttp.send("obj="+JSON.stringify(subject))
```

Серверный скрипт:

```
<?php
if(isset($_POST[“obj”]))
  {
   $obj=json_decode($_POST["obj"]);
   updateInfo($obj->fio, $obj->age, $obj->address);
  }
?>
```

Альтернативным форматом для обмена данных при асинхронной передаче является xml. XML (eXtensible Markup Language) – расширенный язык разметки, ориентированный на хранение и передачу регулярных данных с ориентацией на сеть Интернет [12]. С использованием XML можно структурировать описание сущностей, валидировать корректность их опи-

сания с использованием DTD-описания или XML-схем, описывать визуальное представление сущностей с использованием XSLT. Вот как может выглядеть список субъектов, зарегистрированных в некоторой информационной системе, средствами языка XML:

```
<?xml version="1.0"?>
  <PersonList>
     <Person status=”служащий”>
        <FIO>Иванов Иван Иванович</FIO>
         <BD>
           <day>10</day>
           <month>12</month>
             <year>1990</year>
          </BD>
        <Address>ул.Мира, 22-181</Address>
     </Person>
     <Person status=”пенсионер”>…</Person>
     <Person status=”студент”>…</Person>
  </PersonList>
```

Инструментарий обработки данных в xml-формате поддерживается большинством используемых в Web-разработке языков программирования. Для разбора xml-ответа на клиентской стороне можно использовать XML DOM:

```
     xmlhttp=new XMLHttpRequest();
     xmlhttp.open("GET",”some.xml",false);
     xmlhttp.send();
     xmlDoc=xmlhttp.responseXML;
document.getElementById(“fio").innerHTML=
xmlDoc.getElementsByTagName(“PersonList")[0].childNodes[0].nodeV
alue;
```

Разбор XML-документа средствами DOM API может оказаться рутинной задачей. Упростить поиск, выборку, изменение данных в xml-

документах можно с использованием языка XPath. XPath — это язык запросов к элементам xml или xhtml документа. Синтаксически схож с SQL, представляет собой декларативный язык запросов. XPath рассматривает документ как иерархическую модель из узлов, атрибутов и текстовых элементов.

```
var xhr = new XMLHttpRequest();
var xmldoc;
xhr.open("get", "personlist2.xml", true);
xhr.setRequestHeader("Content-type", "text/xml")
xhr.onreadystatechange = function(){
   if (xhr.readyState == 4){
       if (xhr.status >= 200 && xhr.status < 300){
        xmldoc = xhr.responseXML;
 var personCount = document.evaluate( 'count(//Person)',
xmldoc, null, XPathResult.ANY_TYPE, null );
alert("В ответе "+PersonCount+" персоны");
       }
   }
    };
xhr.send(null);
```

XPath-запрос на клиентской стороне выполняется с использованием метода:

```
document.evaluate(xpathExpression, contextNode,
                          namespaceResolver, resultType, re-
                          sult);
```

где

xpathExpression – xpath выражение

contextNode – узел документа, относительно которого вычисляется выражение

namespaceResolver – функция для координации пространств имен выражения и документа

resultType – тип результата вычисления выражения

result – ссылка на уже существующий XPathResult

Перебор всех персон в вышеприведенном xml-файле с использованием xpath не вызывает трудностей:

```
str="";
var iterator = document.evaluate('//Person', xmldoc, null, XPathResult.UNORDERED_NODE_ITERATOR_TYPE, null);
var pNode = iterator.iterateNext();
while (pNode) {
    str+="Персона:"     +pNode.children[0].textContent+", Ид="+

    pNode.attributes["status"].textContent+"<br>";
    pNode = iterator.iterateNext();
  }
d=document.getElementById("d");
d.innerHTML=str;
```

6.3. Поддержка асинхронного обмена средствами библиотеки jQuery

Программная реализация AJAX-обмена на клиентской стороне предполагает написание стереотипного, повторяющегося для различных запросов кода: кроссбраузерное создание объекта XMLHttpRequest, его параметризация, объявление функций-обработчиков событий для данного объекта. Упростить код и сосредоточиться только на разработке кода логики приложения можно, используя объекты классов FrontEnd-фреймворков – таких, как jQuery, Prototype, VanillaJS и др. Например, в библиотеке JQuery определен метод ajax, инкапсулирующий AJAX-обмен на клиентской стороне:

```
jQuery.ajax( url [, settings] )
```

или

```
jQuery.ajax(settings )
```

где *url* - URL адрес, на который будет отправлен Ajax запрос;

settings - набор параметров вида ключ / значение, которые настраивают запрос Ajax

Параметр *settings* позволяет всесторонне параметризовать ajax-обмен. Объект, передаваемый в качестве этого параметра, поддерживает широкий набор полей, наиболее востребованные из которых перечислены в таблице 3.

Таблица 3. Поля и методы объекта настроек ajax-обмена в jQuery

Поле	Описание	Тип
async По умолчанию: true	Асинхронность ajax-запроса	логический
beforeSend(jqXHR jqXHR, объект settings)	Функция, которая будет вызвана непосредственно перед отправкой ajax-запроса на сервер	функция
complete(jqXHR jqXHR, строка textStatus)	Функция, которая будет вызвана после завершения ajax запроса (срабатывает после функций-обработчиков success и error)	функция или массив
contentType По умолчанию: 'application/x-www-form-urlencoded; charset=UTF-8'	Используемая при отправке запроса кодировка	строка
crossDomain	Является ли запрос кроссдоменным	логический
data	Данные, передаваемые на сервер	объект или строка
dataType	Тип ожидаемых от сервера данных (html, json, xml, script)	строка
error(jqXHR jqXHR, строка textStatus, строка errorThrown)	Функция, исполняемая в случае неудачного запроса	функция или массив
headers По умолчанию: {}	Дополнительные HTTP- заголовки	объект

	запроса	
password	Пароль для HTTP-аутентификации	строка
username	Логин для HTTP-аутентификации	строка
success(объект data, строка textStatus, объект jqXHR)	Функция, которая будет вызвана в случае успешного завершения запроса	функция или массив
timeout	Время ожидания ответа от сервера в миллисекундах	число
type По умолчанию: GET	Определяет тип запроса GET или POST(возможно также DELETE и PUT)	строка
url По умолчанию: текущая страница.	Страница, на которую будет отправлен запрос	строка
xhr По умолчанию ActiveXObject в IE, XMLHttpRequest в других браузерах.	Callback-функция, для создания объекта XMLHttpRequest. Создав свою функцию, вы берёте на себя всю ответственность за формирование объекта	function

Шаблон *ajax*-запроса, сформированного средствами jQuery, выглядит следующим образом:

```
$.ajax({
    url: '/ajax/service.php',
    dataType : "json",
    data: {param1:val1,
           param2;val2 },
```

```
success: function (data, textStatus) {
      /* ... */},
error:function(){
   /*…*/   }
   )};
```

Метод *ajax* является «швейцарским ножом» асинхронных запросов библиотеки jQuery: он очень универсален, имеет множество настроек и применим для любых типов запросов. Если учесть, что чаще всего необходимо подавать обычные *get*- или *post*-запросы, то более удобными и компактными инструментами могут стать методы *post*() и *get*(). Например, метод *get*() имеет прототип:

```
jQuery.get(url,[data],[callback],[dataType])
```

url — url-адрес, по которому будет отправлен запрос. *data* — данные, которые будут отправлены на сервер. Они должны быть представлены в форме объекта, в формате: `{fName1:value1, fName2:value2, ...}`. *callback*(*data, textStatus, jqXHR*) — пользовательская функция, которая будет вызвана после ответа сервера.

data — данные, присланные с сервера.

textStatus — статус того, как был выполнен запрос.

jqXHR — объект jqXHR (в версиях до jquery-1.5, вместо него использовался XMLHttpRequest)

dataType — ожидаемый тип данных, которые пришлет сервер в ответ на запрос

Пример вызова метода get:

```
$.get( "test.php", {“param1”: “value1”, “pa-
ram2”:”value2”}, function( data ) {
```

```
  alert( "Получены данные: " + data );
});
```

Фактически, вызов метода *get*() приводит к вызову метода ajax с параметрами:

```
$.ajax({
  url: url,
  type: "GET",
  data: data,
  success: callback,
  dataType: dataType
});
```

6.4. Асинхронный обмен с использованием fetch API

Современный инструментарий асинхронных обращений к серверу – *fetch API*. Современные браузеры предоставляют альтернативу ставшему уже классическим способу асинхронного обращения клиента к серверу с использованием объекта *XMLHttpRequest* – использование функции *fetch*. Использование этой функции предполагает знакомство с такими концепциями JavaScript, как промисы и *async/await* – функции. Обобщенное определение функции *fetch* выглядит следующим образом:

```
let promise = fetch(url, [options]),
```

где *url* – адрес, к которому осуществляется запрос, *options* – параметры, конфигурирующие запрос. Результатом запроса будет объект-промис, переход которого в resolve-состояние будет соответствовать моменту окончания асинхронной загрузки данных, и параметр callback-обработчика успешного завершения промиса будет содержать объект типа *Response*, из которого можно извлечь переданные сервером данные. Обобщенная схема формирования и обработки результатов запроса будет следующей:

```
fetch(‘/some’)
```

```
.then(response => if(response.ok)response.json())
.then(obj => console.log(obj.field))
.catch(error => console.log(error));
```

В данном случае вызов *fetch* порождает асинхронный запрос на сервер, в случае удачного завершения (если, например, не было сбоя сети), в обработчик успешного завершения промиса (первый then в примере выше) получит объект response, из которого можно извлечь статус HTTP-ответа сервера, получить доступ к телу ответа как объекту *ReadableStream*, а также вызвать методы, позволяющие получить ответ в одном из популярных форматов:

response.text() – ответ сервера возвращается в текстовом формате,

response.json() – ответ сервера возвращается в формате JSON,

response.formData() – ответ сервера возвращается как объект типа *FormData*,

response.blob() – ответ сервера возвращается как объект типа *Blob* (бинарные данные),

response.arrayBuffer() – ответ сервера возвращается объект типа *ArrayBuffer* (низкоуровневые бинарные данные).

В приведенном примере объект *response* проверяется на успешный статус завершения (от 200 до 299) и, если статус ответа в указанном диапазоне, вызывается метод *response.json*(), создающий промис для асинхронного преобразования ответа в *json*-формат. Второй *then*-обработчик получит уже программный объект, поля составляют информационный ответ сервера.

Более короткий вариант вызова *fetch* – использование *async/await* – функций. Тот же пример можно переписать следующим образом:

```
async function asyncRequest(){
try{
```

```
        let response = await fetch(‘/some’);
         if(response.ok)
            obj =  await response.json();
         console.log(obj.field)
           }
         catch(error => console.log(error));
}
```

Асинхронный запрос, отправляемый на сервер с помощью функции *fetch*, можно настраивать, используя второй параметр *options*. Он позволяет, в частности, задавать параметры заголовка HTTP-запроса, указать желательный метод запроса (вместо GET по умолчанию), настроить параметры кеширования, настройки безопасности и др. Вот как, например, отправить асинхронный fetch-запрос методом POST:

```
let news = {
  title: '…',
  content: '…'
};
let response = await fetch('/news/add', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json;charset=utf-8'
  },
  body: JSON.stringify(newsr)
});
let result = await response.json();
console.log(result.message);
```

Как видно из примера выше, поля структуры, передаваемой в параметр *options* задают метод запроса (*method*), заголовки запроса (*headers*), тело запроса (*body*). Подробнее о формировании и обработке результатов асинхронных запросов, использующих *fetch*, можно узнать, например, в [20].

Рассмотренные в настоящем пособии инструментальные средства и методы обработки данных и запросов дают общее представление о распределении функционала между клиентской и серверной сторонами в современных web-приложениях. Были затронуты такие аспекты, как синхронное и асинхронное взаимодействие клиента и сервера, взаимодействие с внешними хранилищами данных, организация сессионности работы web-приложений. Однако, переходя от простых скриптов обработки единичных запросов к комплексным многопользовательским приложениям, необходимо решать задачи проектирования архитектуры разрабатываемой системы, обеспечивающей масштабируемость, модульность, контролируемость процесса проектирования и разработки. Облегчить процесс может использование фреймворков, которые предоставляют программисту наборы классов для обработки запросов и формирования представлений, аутентификации и авторизации, тестирования и развертывания, балансировки нагрузки и профилирования кода. Разработчик может выбрать фреймворк под предпочитаемый язык программирования: PHP (Laravel, Yii2, Smfony, CodeIgniter), Python (Django), JavaScript (NodeJS Express), Java (Spring). Объем второй части пособия не позволяет рассмотреть принципы функционирования серверных фреймворков, подробнее с ними можно познакомиться в специальной литературе [16, 17, 18, 19]. В третьей части пособия, которая будет посвящена вопросам построения архитектур современных web-приложений, будет дан обзор возможностей одного из PHP-фреймворков серверной стороны.

Заключение

Во второй части пособия, посвященного разработке приложений для работы в среде WWW, авторы перенесли внимание читателя на серверную сторону. Новые главы посвящены знакомству с web-сервером, особенностями вызова скриптов для формирования динамического контента в ответ на приходящие запросы, протоколами и форматами симметричного и асимметричного обмена данными между клиентом и сервером в рамках web-приложения.

Тенденции развития современных сред разработки в этой области ведут к постепенному скрытию деталей реализаций протоколов обмена и внутренностей механизмов функционирования скриптов, реализующих некоторые типовые операции (например, аутентификация пользователей, валидация данных, выборка данных из базы). Инструменты становятся все более высокоуровневыми, зачастую просто декларативными, они заметно упрощают и автоматизируют рутинную работу разработчика. Но знакомство с базовыми инструментами разработки, предложенное авторами в настоящем пособии, позволяет глубже понимать происходящее в процессе функционирования web-приложения, решать нестандартные задачи, периодически возникающие перед программистом.

В основу пособия положен опыт преподавания авторами основ проектирования Web-приложений для студентов IT-направлений в Волжском политехническом институте (филиале) ВолгГТУ. Многие наши выпускники по окончании вуза выбрали сферой своих профессиональных интересов именно Web-разработку и считают свою работу творческой и интересной. Авторы желают читателям приобрести твердые знания и уверенные компетенции в сфере Web-технологий, чтобы быть востребованными на рынке труда и обрести возможность работать в перспективной и динамично развивающейся отрасли.

Список литературы

1. http://httpd.apache.org/ Официальный сайт сервера Apache
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. Д.Н. Лясин, О.Ф. Абрамова. Основы проектирования Web-приложений (часть 1): учебное пособие. – Волжский (филиал) ВолгГТУ, 2019г.
5. Администрирование Apache. М. Арнольд, Д.Д. Алмейда, М.: Лори, 2012. – 418с.
6. Online-справочник по языку HTML. — Режим доступа: http://htmlbook.ru/html
7. https://ospanel.io/http://www.denwer.ru/ Официальный сайт проекта OpenServer
8. http://php.net Портал о PHP
9. PHP 5. Д. Котеров, А. Костарев, Спб.: БХВ-Петербург, 2014г., 1104с.
10. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера. Н. Прохоренок, В. Дронов., СПб.: БХВ Петербург, 2019. – 912 с.
11. MySQL по максимуму. Б. Шварц, П. Зайцев, В. Ткаченко., . СПб.: Питер, 2018г., 864с.
12. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5. Р. Никсон. СПб.: Питер, 2015г., 688с.
13. Изучаем PHP7. Руководство по созданию интерактивных Интернет-сайтов. Д. Скляр., М.: Вильямс, 2017г., 769с.
14. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5. Н. Робин. СПб.: Питер, 2019г., 819с.

15.PHP 7. Котеров Д. В., Симдянов И. В., СПб.: БХВ Петербург, 2019. – 1088 с.
16.Laravel. Быстрая разработка современных динамических Web-сайтов на PHP, MySQL, HTML и CSS. Дронов В., СПб.: БХВ Петербург, 2018. – 768 с.
17.Разработка веб-приложений в Yii 2. М. Сафронов, М.: ДМК Пресс, 2015г., 392с.
18.Django 2.1. Практика создания веб-сайтов на Python. Дронов В.А., СПб.: БХВ Петербург, 2019. – 672 с.
19.Разработка веб-приложений с помощью Node.js, MongoDB и Angular. Исчерпывающее руководство по использованию стека MEAN. Д.Бред, Д. Брендан, М.: Вильямс, 2017г., 656с.
20.https://learn.javascript.ru/. Современный учебник JavaScript.

Электронное учебное издание

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

ОСНОВЫ ПРОЕКТИРОВАНИЯ WEB-ПРИЛОЖЕНИЙ
ЧАСТЬ 2

Учебное пособие

Электронное издание сетевого распространения

Редактор Матвеева Н.И.

Темплан 2020 г. Поз. № 5.
Подписано к использованию 27.05.2020. Формат 60x84 1/16.
Гарнитура Times. Усл. печ. л. 5,0.

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

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