

УДК 004.415.25  
ББК 32.973  
Р 17

**Разработка серверных частей интернет-ресурсов [Электронный ресурс]: Учебное пособие** / Волков М.Ю., Литвинов В.В., Лобанов А.А. и др. — М.: МИРЭА – Российский технологический университет, 2021. — 1 электрон. опт. диск (CD-ROM)

Основная цель данного учебного пособия – дать представление о различных технологиях для разработки серверных частей интернет-ресурсов, заинтересовать студентов в изучении этих технологий и оказать помощь в их понимании. Задача пособия – дать краткое изложение основных технологий и методик, необходимых для разработки серверных частей интернет-ресурсов. Учебное пособие предназначено для студентов бакалавриата, ранее не изучавших данные технологии, но также будет полезно более заинтересованным в разработке интернет-ресурсов студентам. Предназначено для освоения учебной программы по дисциплине «Разработка серверных частей интернет -ресурсов» и получения знаний в области разработки серверных частей интернет-ресурсов.

Учебное пособие предназначено для студентов, обучающихся по направлению подготовки 09.03.04 «Программная инженерия».

Учебное пособие издается в авторской редакции.

Авторский коллектив: Волков Михаил Юрьевич, Литвинов Владимир Владимирович, Лобанов Александр Анатольевич, Синицын Анатолий Васильевич.

Рецензенты:

Андреева Ольга Николаевна, д.т.н., доцент, начальник отдела научной работы АО «Концерн «Моринсис-Агат».

Майоров Андрей Александрович, д.т.н., профессор, заведующий кафедрой информационно-измерительных систем, МИИГАиК.

Системные требования:

Наличие операционной системы Windows, поддерживаемой производителем.

Наличие свободного места в оперативной памяти не менее 128 Мб.

Наличие свободного места в памяти постоянного хранения (на жестком диске) не менее 30 Мб.

Наличие интерфейса ввода информации.

Дополнительные программные средства: программа для чтения pdf-файлов (Adobe Reader).

Подписано к использованию по решению Редакционно-издательского совета

МИРЭА — Российский технологический университет.

Объем: 2.92 мб

Тираж: 10

## ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.....	7
СПИСОК ОСНОВНЫХ ТЕРМИНОВ И СОКРАЩЕНИЙ .....	8
1. АРХИТЕКТУРА СЕРВЕРНЫХ ЧАСТЕЙ ИНТЕРНЕТ-РЕСУРСОВ ....	11
1.1. Архитектура клиент-сервер .....	11
1.1.1. Основы архитектуры клиент-сервер .....	11
1.1.2. Виды сервисов .....	13
1.1.3. Понятие масштабируемости .....	14
1.1.4. Протоколы передачи данных .....	15
1.1.5. Понятие «толстого» и «тонкого» клиента .....	17
1.2. Принципы построения архитектуры веб-приложений.....	19
1.2.1. Общие принципы построения архитектуры приложений .....	19
1.2.2. Паттерны проектирования .....	19
2. ОСНОВЫ PHP .....	23
2.1. История PHP, структура PHP .....	23
2.1.1. История PHP .....	23
2.1.2. Конфигурационный файл php.ini.....	25
2.2. Базовый синтаксис PHP. Переменные. Типы.....	25
2.2.1. Первая программа на PHP .....	25
2.2.2. Переменные и типы .....	27
2.2.3. Выражения .....	31
2.2.4. Операторы и операции .....	32
2.2.5. Управляющие конструкции .....	38
2.2.6. Функции .....	45
2.3. Объектно-ориентированное программирование на PHP .....	47
2.3.1. Важное уточнение: ссылки .....	47
2.3.2. Общие принципы .....	48
2.3.3. Область видимости .....	50
2.3.4. Конструкторы и деструкторы .....	52
2.3.5. Разрешение области видимости.....	56
2.3.6. Магические методы .....	57

2.3.7. Наследование .....	63
2.3.8. Абстрактные классы .....	64
2.3.9. Интерфейсы .....	65
2.3.10. Трейты .....	67
2.3.11. Анонимные классы .....	69
2.3.12. Nullsafe и ООП в PHP .....	70
2.3.13. Автоматическая загрузка классов .....	71
2.3.14. Ключевое слово final.....	72
2.3.15. Позднее статическое связывание .....	72
2.3.16. Ковариантность и контравариантность .....	74
2.3.17. Правила совместимости сигнатуры .....	75
3. ОСНОВЫ ВЕБ-СЕРВЕРОВ .....	77
3.1. Ретроспектива развития веб-серверов .....	77
3.1.1. История развития интернета .....	77
3.1.2. Язык динамической сборки веб-страниц SSI.....	80
3.1.3. CGI – Интерфейс общего шлюза .....	81
3.1.4. FastCGI .....	83
3.1.5. PHP-FPM и spawn-fcgi .....	84
3.1.6. SCGI, PCGI, PSGI, WSGI и прочие. ....	86
3.2. Архитектура сервера Apache .....	87
3.2.1. Ядро веб-сервера Apache.....	88
3.2.2. Конфигурация HTTP сервера Apache .....	88
3.2.3. Сервер приложений как модуль Apache .....	88
3.2.4. Многопроцессорные модели (MPM).....	91
3.3. Архитектура сервера nginx.....	93
3.3.1. Основной функционал .....	93
3.3.2. Архитектура .....	94
3.4. Apache vs Nginx: практический взгляд .....	95
3.4.1. Apache.....	95
3.4.2. Nginx .....	96
3.5. Архитектура сервера IIS .....	96

3.5.1. Архитектура службы .....	97
3.5.2. Реализация веб-приложений для ИС .....	97
4. ПОСТРОЕНИЕ ВЗАИМОДЕЙСТВИЯ КЛИЕНТА И СЕРВЕРА .....	99
4.1. Протокол HTTP .....	99
4.1.1. Основные понятия и принципы HTTP-протокола.....	99
4.1.2. История протокола HTTP.....	100
4.1.3. Структура HTTP-протокола.....	101
4.1.4. Согласование контента.....	107
4.1.5. Обработка HTTP-запросов на PHP.....	108
4.1.6. Отправка HTTP-запросов на PHP.....	109
4.2. Обзор парадигм API и способы их реализации .....	110
4.2.1. REST .....	111
4.2.2. GraphQL .....	115
4.2.3. Протокол SOAP .....	125
5. РАБОТА С ДАННЫМИ В PHP .....	134
5.1. Сессии и cookies .....	134
5.1.1. Работа с сессиями и cookies в PHP .....	135
5.2. Работа с файлами разных форматов.....	143
5.2.1. Потоки .....	144
5.2.2. XML и DOM .....	146
5.2.3. JSON .....	148
5.3. Работа с базами данных.....	149
5.3.1. MySQL.....	149
5.3.2. MongoDB.....	150
5.4. Регулярные выражения.....	150
5.5. Валидация данных.....	154
6. СТАНДАРТНАЯ БИБЛИОТЕКА PHP .....	162
6.1. Модули и сторонние библиотеки PHP.....	162
6.1.1. Установка сторонних модулей на PHP .....	162
6.1.2. Обзор сторонних библиотек PHP .....	167
6.2. Стандартная библиотека PHP .....	179

6.2.1. Время и дата.....	179
6.2.2. Работа с изображениями.....	183
СПИСОК ИСТОЧНИКОВ.....	185

## **ВВЕДЕНИЕ**

Назначением данного учебного пособия является дать краткое изложение основных технологий и методик, необходимых для разработки серверных частей интернет-ресурсов. Дисциплина «Разработка серверных частей интернет-ресурсов» является одной из основных дисциплин для подготовки специалистов по специальностям «Разработка программных продуктов и проектирование информационных систем» и «Разработка и дизайн компьютерных игр и мультимедийных приложений» по направлению подготовки 09.03.04 «Программная инженерия». Данное учебное пособие решает задачи воспитания и обучения учащихся навыкам саморазвития в рамках получения знаний о технологиях и их применении в процессе разработки интернет-ресурсов, в частности серверной части. При изучении данной дисциплины рекомендуется использовать практический подход, основанный на практическом изучении технологий, то есть их непосредственном применении и тестировании. При изучении материалов данного пособия рекомендуется дополнительно использовать следующие средства: среду разработки с бесплатной студенческой лицензией PhpStorm, средство контейнеризации Docker, веб-сервера Apache и Nginx. Данная дисциплина тесно связана с такими дисциплинами, как «Разработка клиентских частей интернет-ресурсов», «Интерфейсы прикладного программирования», «Архитектура клиент-серверных приложений». Данное пособие также дает возможность студентам других специальностей в рамках индивидуальной траектории изучить методы, технологии и подходы к изучению данной дисциплины.

## СПИСОК ОСНОВНЫХ ТЕРМИНОВ И СОКРАЩЕНИЙ

**ООП** – объектно-ориентированное программирование.

**Сервер (программное обеспечение)** – программный компонент вычислительной системы, выполняющий сервисные (обслуживающие) функции по запросу клиента, предоставляя ему доступ к определённым ресурсам или услугам.

**Сервер (аппаратное обеспечение)** – выделенный или специализированный компьютер для выполнения сервисного программного обеспечения без непосредственного участия человека.

**Клиент** – это аппаратный или программный компонент вычислительной системы, посылающий запросы серверу.

Программа, являющаяся клиентом, взаимодействует с сервером, используя определённый протокол. Она может запрашивать с сервера какие-либо данные, манипулировать данными непосредственно на сервере, запускать на сервере новые процессы и т. п. Полученные от сервера данные клиентская программа может предоставлять пользователю или использовать как-либо иначе, в зависимости от назначения программы. Программа-клиент и программа-сервер могут работать как на одном и том же компьютере, так и на разных. Во втором случае для обмена информацией между ними используется сетевое соединение.

На одном и том же компьютере могут одновременно работать программы, выполняющие как клиентские, так и серверные функции. Например, веб-сервер может в качестве клиента получать данные для формирования страниц от SQL-сервера.

Сервер и клиент (рабочая станция) могут иметь одинаковую аппаратную конфигурацию, так как различаются лишь по участию в своей работе человека.

**База данных** – это информационная модель, позволяющая упорядоченно хранить данные об объекте или группе объектов, обладающих набором свойств, которые можно категорировать. Базы данных функционируют под управлением систем управления базами данных (сокращенно СУБД).

**API (Application Programming Interface)** – прикладной программный интерфейс) – набор функций и подпрограмм, обеспечивающий взаимодействие клиентов и серверов.

**API (в клиент-сервере)** – описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой.

**Сервис** – легко заменяемый компонент сервисно-ориентированной архитектуры со стандартизированными интерфейсами.

**Веб-сервис (веб-служба)** – идентифицируемая уникальным веб-адресом (URL-адресом) программная система со стандартизированными интерфейсами.

**Web API** – используется в веб-разработке, содержит, как правило, определённый набор HTTP-запросов, а также определение структуры HTTP-ответов, для выражения которых используют XML– или JSON–форматы.

Web API является практически синонимом для веб-службы, хотя в последнее время за счёт тенденции Web 2.0 осуществлен переход от SOAP к REST типу коммуникации. Веб-интерфейсы, обеспечивающие сочетание нескольких сервисов в новых приложениях, известны как гибридные.

**Веб-сервер** – сервер, который хранит, обрабатывает и доставляет веб-страницы для клиента. Веб-серверы доставляют клиенту только статический HTML-контент клиенту. Под статическим контентом будем понимать различного рода документы (вне зависимости от типа файлов), файлы изображений, видео файлы и т.д.

**Сервер приложения (application server)** – это программная платформа, предназначенная для эффективного выполнения бизнес-логики приложения. Сервер приложений действует как набор компонентов, доступных разработчику программного обеспечения через API (интерфейс прикладного программирования), определённый самой платформой. При использовании трехзвенной архитектуры сервер-приложения располагается в серверной части.

Сервер приложений действует как хост для бизнес-логики пользователя, он также обеспечивает доступ к бизнес-приложениям и задаёт их параметры для пользователя.

**SSI (Server Side Includes – включения на стороне сервера)** – язык для динамической «сборки» веб-страниц на сервере из отдельных составных частей и выдачи клиенту полученного HTML-документа.

**CGI (Common Gateway Interface – интерфейс общего шлюза)** – стандарт интерфейса, используемого внешней программой для связи с веб-сервером.

**URI (Uniform Resource Identifier)** – унифицированный (единообразный) идентификатор ресурса) – URI есть последовательность символов, идентифицирующая абстрактный или физический ресурс.

**URL (Uniform Resource Locator)** – универсальный указатель ресурса, который помимо сведений о ресурсе определяет также его физическое местоположение.

**HTTP (англ. HyperText Transfer Protocol)** – протокол прикладного уровня (верхний, 7 уровень, в модели OSI) передачи данных, используемый для пе-



редачи произвольных данных, относится к набору протоколов семейства TCP/IP.

Согласование контента (англ. content negotiation) — это механизм, используемый для отображения различных представлений ресурса по тому же URI, так чтобы клиент мог указать, что лучше подходит для пользователя (например, желаемый язык документа, формат изображения, или кодировку текста).

**CMS (Content Management System)** – система управления содержимым или иначе – система управления контентом. Системы управления контентом представляют собой самостоятельные информационные системы, которые используются для обеспечения и организации совместного процесса создания, редактирования и управления содержимым веб-ресурсов.

# 1. АРХИТЕКТУРА СЕРВЕРНЫХ ЧАСТЕЙ ИНТЕРНЕТ-РЕСУРСОВ

## 1.1. Архитектура клиент-сервер

### 1.1.1. Основы архитектуры клиент-сервер

Когда идет речь о вычислительных моделях, модель клиент-сервер занимает прочное место среди методов распределенных вычислений. Часто данную модель называют просто архитектура клиент-сервер. Данная модель – это идея разделения системы или приложения на отдельные задачи, размещаемые на различных платформах для большей эффективности. Применение данной идеи лежит в основе архитектуры клиент-сервер, распределенных вычислений, архитектуры приложений и т. д. На рис. 1.1 представлена общая структура архитектуры клиент-сервер:

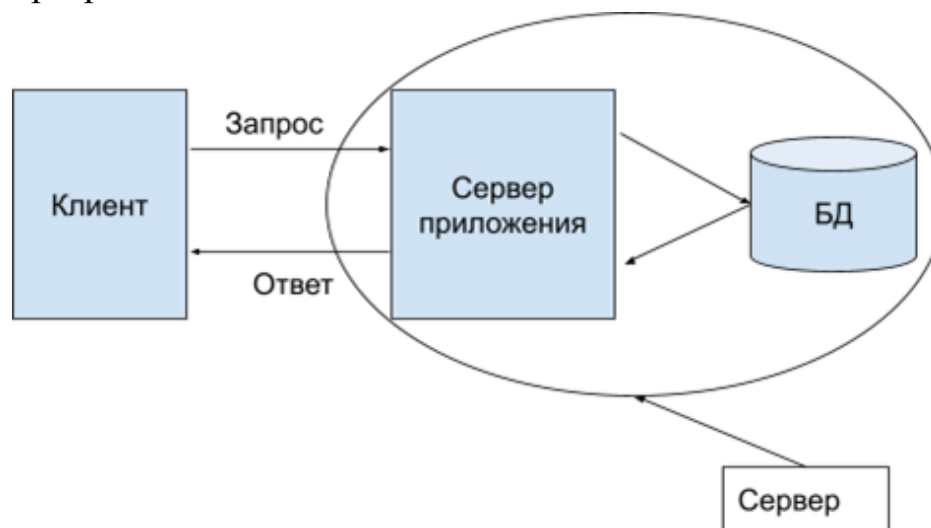


Рисунок 1.1. Общая структура архитектуры клиент-сервер

Одна из частей модели – клиент, имеющий дружественный интерфейс. Клиент представляет собой программу представления данных, которая для их получения посылает запросы серверу, который в свою очередь может делать запрос к базе данных, обрабатывать данные и возвращать их клиенту. Возможны случаи разделения обработки данных, когда часть работы сервера в виде обработки данных выполняет клиент. Но нужно понимать, что в этом случае очень важно разделение обязанностей и уровней доступа к данным на стороне клиента.

Если рассматривать в общем случае как модель, архитектура клиент-сервер — это сетевое окружение, в котором управление данными осуществляется на серверном узле, а другим узлам (клиентам) предоставляется доступ к данным.

Клиент-серверная архитектура имеет следующие достоинства:

- 1) основная нагрузка ложится на сервер, в системе имеется одна (или несколько) мощная серверная конфигурация часто сложной конструкции и множество “слабых” машин клиентов, что снижает общую стоимость всей системы;
- 2) данные находятся в безопасности, так как сервер дает клиенту только требуемую выборку данных для клиента, основываясь на его уровне доступа;
- 3) разграничение полномочий между клиентом и сервером;
- 4) кроссплатформенность – реализаций клиента может быть сколько угодно: от веб-браузера до приложений мобильных платформ;
- 5) нет дублирования сервисов на каждом клиенте, основная обработка данных лежит на сервере.

Нужно понимать, что наряду с плюсами существуют некоторые минусы:

- 1) неработоспособность сервера может сделать неработоспособной всю информационную систему. Неработоспособным сервером следует считать сервер, производительности которого не хватает на обслуживание всех клиентов, а также сервер, находящийся на ремонте, профилактике и т. п., в случае отсутствия горизонтальной масштабируемости;
- 2) поддержка работы данной системы требует отдельного специалиста;
- 3) высокая стоимость серверного оборудования.

С рассмотрения модели клиент-сервер спустимся на уровень ниже в конкретику интернет-ресурсов и введем следующее определение:

Веб-приложение — это клиент-серверное приложение, в котором клиентом выступает браузер, а сервером – веб-сервер (в широком смысле).

Четырьмя основными компонентами архитектуры веб-приложений являются:

1. Клиент – в общем случае это браузер, но в ситуации взаимодействия двух серверов один является сервером, а другой клиентом.
2. Веб-сервер – это сервер, принимающий запросы от клиентов и выдающий им ответы. Веб-сервером называют как программное обеспечение, выполняющее функции веб-сервера, так и непосредственно компьютер, на котором это программное обеспечение работает.
3. База данных.
4. API.

Самая простая клиент-серверная конфигурация, изображенная на рис. 1.1,

не является единственно возможной. Существуют более сложные конфигурации, состоящие из множества серверов, так называемые трехзвенные и при их расширении N-звенные архитектуры. Рассмотрим эти основные архитектуры:

Двухзвенная архитектура (изображена на рис. 1.1). Состоит из 2 базовых компонентов. Сервер напрямую отвечает на запросы клиентов.

Трехзвенная архитектура (изображена на рис. 1.2). Сервер состоит более, чем из двух самостоятельных частей, которые взаимодействуют друг с другом.

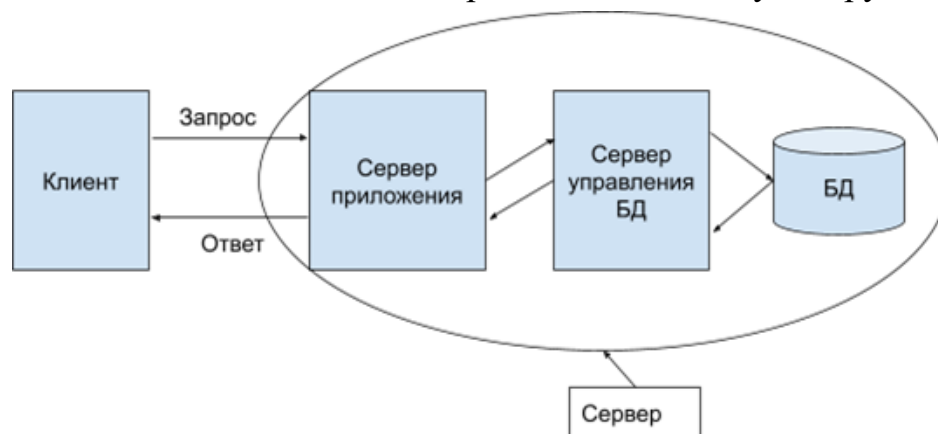


Рисунок 1.2. Трехзвенная архитектура

Плюсами трехзвенной архитектуры дополнительно являются:

- 1) высокая степень гибкости и масштабируемости;
- 2) высокая степень безопасности (т.к. защиту можно определить для каждого сервиса или уровня);
- 3) высокая производительность (т.к. задачи распределены между серверами).

### 1.1.2. Виды сервисов

Существует великое множество возможных сервисов, как самостоятельных, так и в составе приложений. Рассмотрим некоторые из них:

1. **Серверы приложений.** Сервер приложений (англ. application server) – это программная платформа (фреймворк), предназначенная для эффективного исполнения процедур (программ, скриптов), на которых построены приложения.
2. **Веб-серверы.** Являются подвидом серверов приложений. Изначально предоставляли доступ к гипертекстовым документам по протоколу HTTP. Сейчас поддерживают расширенные возможности, в частности, передачу произвольных данных.
3. **Серверы баз данных.** Серверы баз данных используются для обработки запросов. На сервере находится СУБД для управления БД и ответов на запросы.

4. **Файл-серверы.** Файл-сервер хранит информацию в виде файлов и предоставляет пользователям доступ к ней. Как правило, файл-сервер обеспечивает и определенный уровень защиты от несанкционированного доступа.
5. **Прокси-сервер.** Прокси-сервер (от англ. proxy – представитель, уполномоченный; часто просто прокси, сервер-посредник) – промежуточный сервер (комплекс программ) в компьютерных сетях, выполняющий роль посредника. Существует несколько видов прокси-серверов. Веб-прокси – широкий класс прокси-серверов, служащий для кэширования данных. Обратный прокси – прокси-сервер, который, в отличие от веб-прокси, ретранслирует запросы клиентов из внешней сети на один или несколько серверов, логически расположенных во внутренней сети. Часто используется для балансировки сетевой нагрузки между несколькими веб-серверами и повышения их безопасности, играя при этом роль межсетевого экрана на прикладном уровне.
6. **Файрволы (брандмауэры).** Межсетевые экраны, анализирующие и фильтрующие проходящий сетевой трафик, с целью обеспечения безопасности сети.
7. **Почтовые серверы.** Предоставляют услуги по отправке и получению электронных почтовых сообщений.

### ***1.1.3. Понятие масштабируемости***

Вернемся к одному из плюсов трехуровневой архитектуры, а точнее к понятию масштабируемости. Масштабируемость – способность работать с увеличенной нагрузкой путем наращивания ресурсов без фундаментальной перестройки архитектуры и/или модели реализации при добавлении ресурсов. Система называется масштабируемой, если она способна увеличивать производительность пропорционально дополнительным ресурсам. Основными являются горизонтальная и вертикальная масштабируемость.

Вертикальная масштабируемость (англ. scaling up) представляет собой увеличение производительности компонентов серверной системы в интересах повышения производительности всей системы. Следует понимать, что система ограничена объемом памяти, скоростными характеристиками и т.д. Данный метод не снимает нагрузку на всю систему, но является самым простым. Ярким примером является увеличение оперативной памяти, установка более мощного процессора.

Горизонтальная масштабируемость (англ. scaling out) представляет собой как разбиение системы на более мелкие структурные компоненты и разнесение их, так и увеличение количества компонентов, параллельно выполняющих одну и ту же функцию. Данный метод часто требует изменение программного обеспечения для налаживания взаимодействия между новыми компонентами. Частым примером является добавление еще одного сервера тех же характеристик к существующему.

#### ***1.1.4. Протоколы передачи данных***

Протокол передачи данных – набор определенных правил или соглашений интерфейса логического уровня, который определяет обмен данными между различными программами. Эти правила задают единообразный способ передачи сообщений и обработки ошибок.

Сетевой протокол – набор правил, действий и их очередности, позволяющий осуществлять соединение и обмен данными между двумя и более включенными в сеть устройствами. Сетевые протоколы предписывают правила работы компьютерам, которые подключены к сети. Они строятся по многоуровневому принципу. Протокол некоторого уровня определяет одно из технических правил связи.

В общей классификации протоколы делятся на низкоуровневые, протоколы верхнего уровня и протоколы промежуточного уровня. К промежуточному уровню относятся коммуникационные протоколы и протоколы аутентификации. Протоколами верхнего уровня являются прикладные, сеансовые протоколы и протоколы представления. Физический, канальный, сетевой и транспортный протоколы относят к низкоуровневым протоколам.

Протоколы определяются следующими организациями: IETF (Internet Engineering Task Force), IEEE (Institute of Electrical and Electronics Engineers), ISO (International Organization for Standardization), ITU-T (International Telecommunication Union – Telecommunication sector).

В настоящее время для сетевых протоколов используется две основных сетевых модели: сетевая модель OSI (Open System Interconnection – взаимодействие открытых систем) и стек протоколов TCP/IP.

Вот как традиционно протоколы TCP/IP вписываются в модель OSI:

Таблица 1.1. Распределение протоколов по уровням модели OSI

	Уровень TCP/IP	Уровень OSI	
7	Прикладной	Прикладной	HTTP, SMTP, SNMP, FTP, Telnet, SSH, SCP, SMB, NFS, RTSP, BGP
6		Представления	TLS, SSL
5		Сеансовый	RPC, NetBIOS, PPTP, L2TP
4	Транспортный	Транспортный	TCP, UDP, GRE
3	Сетевой	Сетевой	IP, ICMP, IGMP, OSPF, RIP, IPX
2	Канальный	Канальный	Ethernet, Token ring, HDLC, PPP, X.25, Frame relay, ISDN
1		Физический	электрические провода, радиосвязь, волоконно-оптические провода, инфракрасное излучение

В рамках данного курса мы ограничимся рассмотрением 7 и 6 уровней модели OSI или 4 уровня модели IP.

Модель OSI. 6 и 7 уровни логической модели работы сети:

- 1) 6-ой уровень, уровень представления служит для преобразования данных из внутреннего формата компьютера в формат передачи;
- 2) 7-ой уровень, прикладной уровень является пограничным между прикладной программой и другими уровнями, обеспечивая удобный интерфейс связи для сетевых программ пользователя.

Аналогично рассмотренному, стек протоколов TCP/IP содержит 4-ый – прикладной уровень (application layer) для решения тех же задач.

Наиболее известные прикладные протоколы, используемые в сети Интернет:

1. Протокол RTP (Real-time Transport Protocol), протокол работает на прикладном уровне (OSI – 7) и используется при передаче трафика реального времени.
2. HTTP (Hyper Text Transfer Protocol) – это протокол передачи гипертекста.
3. FTP (File Transfer Protocol) – это протокол передачи файлов со специального файлового сервера на компьютер пользователя.

4. POP3 (Post Office Protocol) – это стандартный протокол почтового соединения.
5. SMTP (Simple Mail Transfer Protocol) – протокол, который задает набор правил для передачи почты.
6. TELNET – это протокол удаленного доступа.

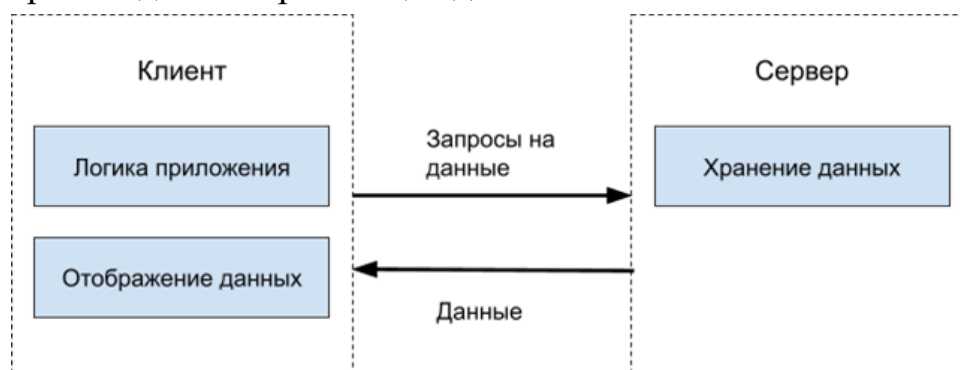
В массе своей эти протоколы работают поверх TCP или UDP и привязаны к определённому порту, порты определены Агентством по выделению имен и уникальных параметров протоколов (IANA), например:

- 1) HTTP на TCP-порт 80 или 8080;
- 2) FTP на TCP-порт 20 (для передачи данных) и 21 (для управляющих команд);
- 3) SSH на TCP-порт 22;
- 4) запросы DNS на порт UDP (реже TCP) 53;
- 5) обновление маршрутов по протоколу RIP на UDP-порт 520;

#### **1.1.5. Понятие «толстого» и «тонкого» клиента**

При классификации компонентов архитектуры клиент-сервер существует понятия “толстый” и “тонкий” клиент.

При применении толстого клиента полная функциональность приложения обеспечивается вне зависимости от сервера. Схематично это изображено на рис. 1.3. В данном случае сервер чаще всего выступает в роли хранилища информации, а вся логика приложения, как и механизм отображения данных располагаются и выполняются на клиенте. Даже при отсутствии соединения с сервером работа ведется с локальными копиями данных, а при возобновлении соединения происходит синхронизация данных.

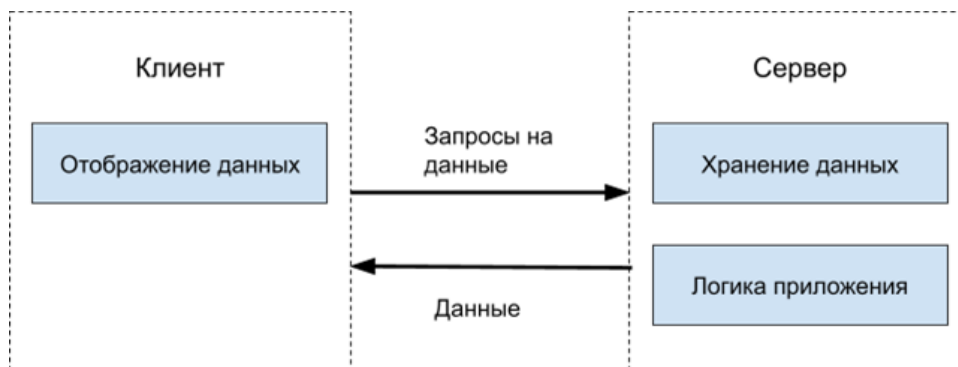


*Рисунок 1.3. Толстый клиент*

Тонким клиентом называют компьютеры и программы, функционирующие в терминальной или серверной сети. Множество задач по обработке данных осуществляются на главных компьютерах, к которым присоединено приложение и компьютер. Тонкий клиент же в отличие от толстого только отображает



данные, принятые от сервера. Вся логика приложения выполняется на более производительном сервере, что не требует клиентских мощностей, кроме хорошего и стабильного канала связи. К сожалению, любой сбой на сервере и в канале связи влечет “падение” всего приложения. Данная технология изображена на рис. 1.4.



*Рисунок 1.4. Тонкий клиент*

Рассматривая два этих подхода в совокупности, следует выделить существенные различия:

1. Различные требования к каналу связи, для тонкого клиента это существенно важно, т.к. потеря соединения сильно влияет на работоспособность и даже полностью тормозит работу приложения.
2. Обширные функции продукта при использовании толстого клиента не совместимы с политикой безопасности, при использовании же тонкого клиента обработка данных идет на уровне сервера, что безопаснее.
3. При использовании толстого клиента значительно усложняется совместная работа с данными, их синхронизация и обновление, хотя данную проблему решает методика тонкого клиента, но все же данную проблему в большей или меньшей степени можно отнести к обоим методикам.
4. При использовании толстого клиента требуются значительные мощности на пользовательском компьютере для обработки данных, что повышает требования к пользовательским машинам.
5. Технология тонкого клиента упрощает работу с исправлением ошибок в программном обеспечении, его обновлении и т.д., за счет того, что для исправления логики приложения достаточно внести изменения один раз на сервере.

## **1.2. Принципы построения архитектуры веб-приложений**

### **1.2.1. Общие принципы построения архитектуры приложений**

Для понимания различий между архитектурой системы и архитектурой программного обеспечения в нашем случае веб-приложения обратимся к определению архитектуры программного обеспечения. Архитектура программного обеспечения (англ. software architecture) – совокупность важнейших решений организации программной системы. Архитектура включает:

- 1) выбор структурных элементов и их интерфейсов, с помощью которых составлена система, а также их поведения в рамках сотрудничества структурных элементов;
- 2) соединение выбранных элементов структуры и поведения в более крупные системы;
- 3) архитектурный стиль, который направляет всю организацию – все элементы, их интерфейсы, их сотрудничество и их соединение.

Разработка архитектуры очень важный и сложный процесс. От полученного результата зависит дальнейшее развитие программного продукта. Возможно наложение ненужных слоев и идей, перерастающее в огромный ком со сложными взаимоотношениями внутри, который без хорошо написанной документации будет сложно содержать и поддерживать, в случае выбора неподходящей архитектуры. Или архитектура изначально будет выбрана соответственно проекту и требованиям, что в результате даст расширяемый и модернизируемый продукт.

### **1.2.2. Паттерны проектирования**

Для решения проблем проектирования были созданы архитектурные конструкции, называемые паттернами проектирования или часто коротко паттернами. Существует фундаментальный паттерн (шаблон), нашедший свое применение повсеместно, не только в веб-разработке. Название паттерну дают первые буквы его основных компонентов: Model View Controller.

Первая часть данного паттерна – это модель (Model). Это представление содержания функциональной бизнес-логики приложения. Модель, как и любой компонент архитектуры под управлением данного паттерна, не зависит от остальных частей продукта. То есть слой, содержащий модель, ничего не знает об элементах дизайна и любом другом отображении пользовательского интерфейса. Обобщая, модель обладает следующими признаками:

- 1) модель – это бизнес-логика приложения;
- 2) модель обладает знаниями о себе самой и не знает о контроллерах и

представлениях;

- 3) для некоторых проектов модель – это просто слой данных (DAO, база данных, XML-файл);
- 4) для других проектов модель – это менеджер базы данных, набор объектов или просто логика приложения.

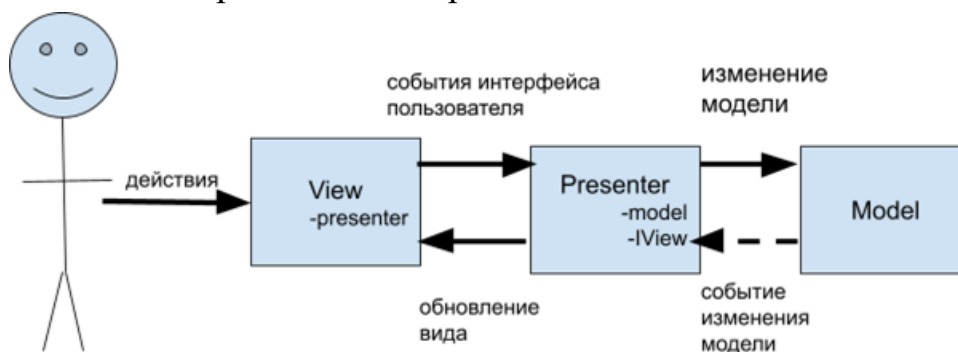
Представление (View) – это отображение данных, получаемых от модели. Никакого влияния на модель представление оказать не может. Данное разграничение является разделением компетенций компонентов приложения и влияет на безопасность данных. Если рассматривать интернет-ресурсы, представлением является html-страница.

Третьим компонентом системы является контроллер (Controller). Данный компонент является неким буфером между моделью и представлением. Обобщенно он управляет представлением на основе изменения модели.

Существуют разные виды MVC-паттерна, различающихся 3 компонентом системы. Наиболее распространенными видами являются:

1. Model-View-Presenter;
2. Model-View-View Model;
3. Model-View-Controller.

Особенностью паттерна Model-View-Presenter является то, что он позволяет создавать абстракцию представления. Для реализации данного метода выделяется интерфейс представления, а презентер получает ссылку на реализацию интерфейса, подписывается на события представления и по запросу меняет модель. Данный подход представлен на рис. 1.5.



*Рисунок 1.5. Паттерн Model-View-Presenter*

Признаки подхода с использованием презентера:

- 1) двусторонняя коммуникация с представлением;
- 2) представление взаимодействует напрямую с презентером путем вызова соответствующих функций или событий экземпляра презентера;
- 3) презентер взаимодействует с представлением путем использования специального интерфейса, реализованного представлением;

4) одному презентеру соответствует одно отображение.

Особенностью паттерна Model-View-View Model является связывание элементов представления со свойствами и событиями View-модели. Схематично данная идея изображена на рис. 1.6.

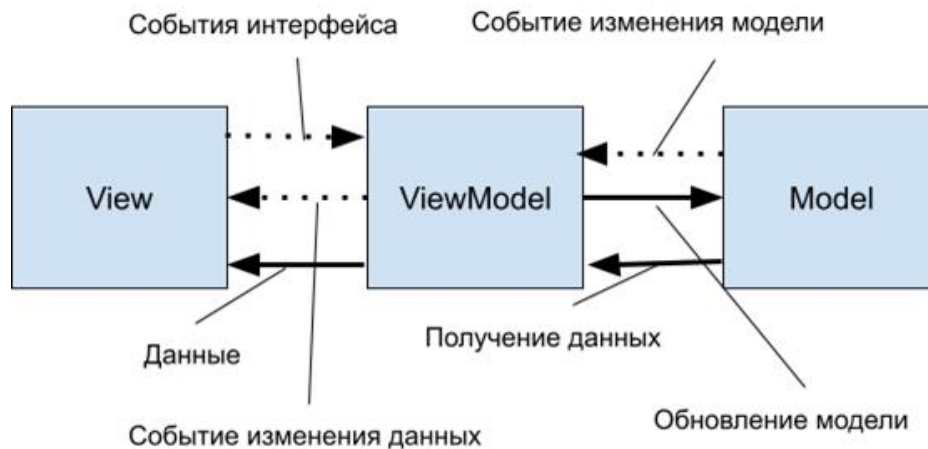


Рисунок 1.6. Паттерн Model-View-View Model

Признаками данного подхода являются:

1. Двусторонняя коммуникация с представлением.
2. View-модель — это абстракция представления. Означает, что свойства представления совпадают со свойствами View-модели или модели.
3. View-модель не имеет ссылки на интерфейс представления (IView). Изменение состояния View-модели автоматически изменяет представление и наоборот, поскольку используется механизм связывания данных (Bindings).
4. Одному экземпляру View-модели соответствует одно отображение.

Особенностью паттерна Model-View-Controller является то, что контроллер и представление зависят от модели, но при этом сама модель не зависит от двух других компонентов. Представление данного паттерна представлено на рис. 1.7.

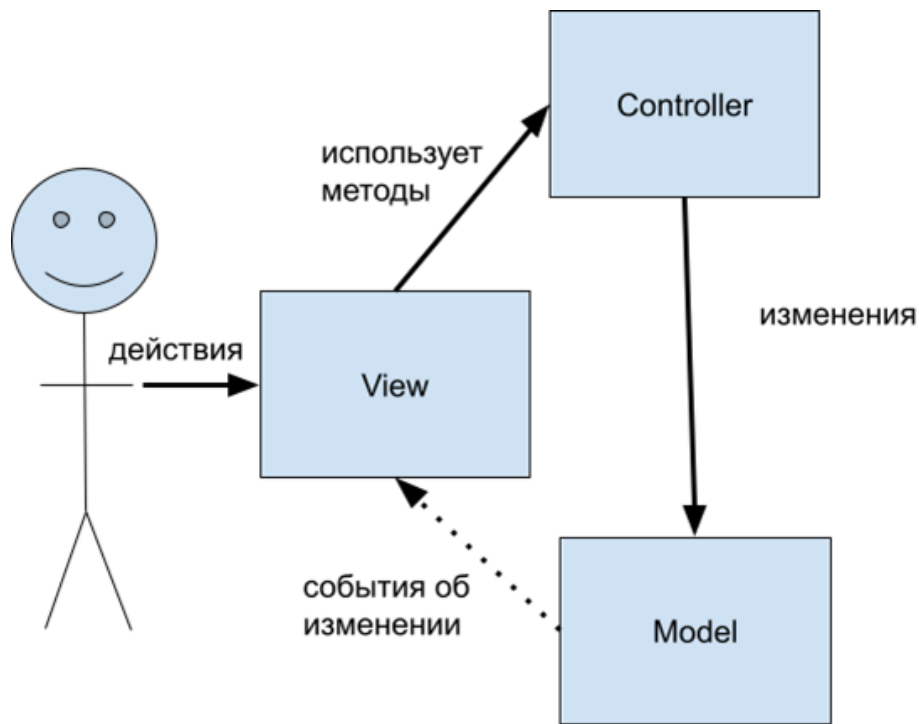


Рисунок 1.7. Паттерн Model-View-Controller

Признаками данного подхода являются:

1. Контроллер определяет, какое представление должно быть отображено в требуемый момент. Если рассматривать применение для разработки веб-приложений, то контроллер управляет запросами пользователя. Его основная функция – вызывать и координировать действие необходимых ресурсов и объектов, нужных для выполнения действий, задаваемых пользователем. Обычно контроллер вызывает соответствующую модель для задачи и выбирает подходящий вид.
2. События представления могут повлиять только на контроллер. Контроллер может повлиять на модель и определить другое представление.
3. Возможно несколько представлений только для одного контроллера. Данный вариант чаще всего используется при разработке веб-приложений.

## 2. ОСНОВЫ PHP

### 2.1. История PHP, структура PHP

#### 2.1.1. История PHP

Конец 90-х – пора бурного развития интернета, поэтому остро вставал вопрос о том, на чем писать веб-приложения. В то время разработчики располагали тремя основными языками: C, Java и Perl. Разработка сайтов на C была очень трудоемкой, так как язык является низкоуровневым. Такое средство, как Java, было дорогим, тяжеловесным ну и, конечно же, ресурсоемким. Единственным решением было использование скриптового языка Perl, именно на нем были написаны первые CGI-сценарии. Для справки CGI-сценарии – это приложения разработанные в соответствии со спецификацией CGI (Common Gateway Interface). CGI (от англ. Common Gateway Interface – «интерфейс общего шлюза») – стандарт интерфейса, используемого внешней программой для связи с веб-сервером.

В 1994 году датский программист Расмус Лердорф (Rasmus Lerdorf) создал набор CGI-скриптов на языке программирования C для вывода и учёта посетителей его онлайн-резюме, обрабатывающий шаблоны HTML-документов. Лердорф назвал набор Personal Home Page Tools (Инструменты Личной Домашней Страницы), но более часто упоминалось название “PHP Tools”. Вскоре функциональности перестало хватать, и Расмус переписал PHP Tools, создав реализацию с расширенным функционалом.

В июне 1995 года Расмус открыл исходный код PHP Tools общественности, что позволило разработчикам использовать его по своему усмотрению. Это также дало возможность пользователям исправлять ошибки в коде и улучшать его.

В 1997 году после длительного бета-тестирования вышла вторая версия обработчика, написанного на C – PHP/FI 2.0. Её использовали около 1 % (приблизительно 50 тысяч) всех интернет-доменов мира. Он совершенно не был похож на современный PHP, а представлял собой некий шаблонный язык.

Третья версия была разработана также на языке C израильскими программистами Энди Гутмансом и Зеевом Сураски. Она построена на модульной основе, что позволяло сторонним разработчикам создавать расширения языка. Данная разработка велась в сотрудничестве с Расмусом Лердорфом. Язык был назван просто “PHP” – аббревиатура, означающая рекурсивный акроним – Hypertext Preprocessor.

Уже в 1999 году был выпущен новый движок Zend Engine. А в 2000 году

вышла обновленная версия PHP 4.0. Данный движок используется в своей обновленной версии по сей день. Через 6 лет в мир врывается 5 версия языка на второй версии движка с введением концепции ООП. Шестая версия так и не была выпущена, но ее наработки были выпущены версией 5.3. Это были важные изменения в интерпретаторе, появление пространства имен, а также лямбда функций и замыканий.

PHP набирал популярность за счет своей ориентированности на веб-приложения и такие крупные и значимые продукты, как Facebook и Wikipedia, были созданы именно на нем. С появлением новых мощных и удобных фреймворков в других языках программирования PHP начал уходить на задворки веб-разработки, все чаще при разработке новых продуктов использовались C#, .NET, Ruby on Rails, Django, Go, Node.js и другие. В то время как другие языки поддерживали MVC-фреймворки для веб-разработки, PHP фактически не поддерживал ООП. Тем, что было, невозможно было пользоваться. Переход небольших сайтов и проектов на новый фреймворк не составляет большого труда, но переход такого гиганта как Facebook – это страшная катастрофа, поэтому данная компания занялась модификацией языка, на котором написан их главный продукт.

Далее с 2010 года идет поэтапная модификация языка компанией. Был создан компилятор из PHP в C++, а потом и виртуальной машины. Следует напомнить, что PHP это язык с открытым исходным кодом и его развивали крупные корпорации параллельно, часто никак между собой не согласовываясь.

В 2015 году вышел PHP 7 на движке Zend Engine 3. Успехом разработчиков стало ускорение движка в два раза на фоне снижения потребления оперативной памяти и добавление многих удобных и распространенных инструментов.

В 2020 году вышла 8 версия PHP, были исправлены и доработаны многие спорные нюансы, появившиеся после выхода 7 версии. Главным достижением стала компиляция Just-In-Time, которая позволяет создать даже ИИ.

В заключении данного раздела можно сказать, что в развитие языка было вложено много средств как независимыми командами разработчиков, так и крупными корпорациями. В результате имеется актуальный язык, ориентированный на веб-разработку, качественно решающий поставленные перед ним задачи. Далее будут рассмотрены конфигурирование и базовый синтаксис языка PHP.

### 2.1.2 Конфигурационный файл *php.ini*

В конфигурационном файле сосредоточены настройки интерпретатора PHP и его многочисленных расширений. На начальном этапе изучения PHP использование данного файла будет сведено к минимуму, но для более тонкой профессиональной настройки он необходим. Содержимое данного файла представляет собой секции и директивы. Секции заключаются в квадратные скобки, после них идут директивы формата “directive = value”. Комментариями являются строки, начинающиеся с точки запятой. Возможности настройки данного файла обширны: языковые опции, ограничение ресурсов, настройка производительности, настройка обработки данных, настройка путей и директорий загрузки файлов и тому подобное. Рассмотрим простой пример в виде следующего листинга 2.1:

Листинг 2.1. Отрывок файла *php.ini*

```
[MySQLi]
mysqli.max_persistent = -1
mysqli.allow_persistent = On
mysqli.max_links = -1
mysqli.default_port = 3306
```

В примере в листинге 2.1 рассматривается отрывок конфигурационного файла настроек *php.ini* для использования модуля для взаимодействия с СУБД MySQL. Директива *mysqli.max\_persistent* обозначает максимальное количество постоянных ссылок для подключения к СУБД. Значение -1 снимает ограничения на количество ссылок. Директива *mysqli.allow\_persistent* отвечает за запрет или его отсутствие на постоянные ссылки. Директива *mysqli.max\_links* обозначает максимальное количество ссылок. Значение -1 так же, как и для директивы *mysqli.max\_persistent*, снимает ограничение. Директива *mysqli.default\_port* указывает порт для подключения к базе данных по умолчанию.

## 2.2. Базовый синтаксис PHP. Переменные. Типы

### 2.2.1. Первая программа на PHP

Рассмотрим простейшую программу на PHP, представленную в листинге 2.2. Весь код обернут в тег `<?php?>`, а сам код представляет собой оператор вывода `echo` и строку “Hello World”, которую данный оператор собственно выводит на экран. Важно отметить, что как самостоятельный код или как вставка в `html`-код скрипт PHP всегда оборачивается в тег `<?php?>`. В первом случае конец тега можно опустить.



## Листинг 2.2. Простая программа на PHP

```
<?php
    echo "Hello World";
?>
```

Рассмотрим второй вариант простейшей программы на PHP, представленной в листинге 2.3. Главной особенностью языка является поддержка обработки HTML, потому пример программы с простым статическим текстом может выглядеть и так. Использование оператора вывода `echo` здесь необязательно, как и именно PHP-кода. Как уже было сказано ранее, PHP с легкостью встраивается в HTML-код.

## Листинг 2.3. Второй вариант простой программы на PHP

```
<body>
Hello World
</body>
```

Рассмотрим пример такой простой комбинации, представленной в листинге 2.4. Если поверхностно рассматривать данный пример, то все, что находится внутри тега, интерпретируется как программа, а все остальное интерпретируется как HTML-документ. Создание списка происходит программно, то есть теги начала и конца списка находятся и интерпретируются в формате HTML-документа, а уже его элементы являются результатом выполнения кода. В данном примере также представлено три способа написания комментариев в коде: с помощью двойного слэша, символа решетка и многострочного комментария в теге слэш со звездочкой. Нужно понимать, что данный синтаксис работает только внутри тега `<?php?>`. Вне его работают правила HTML.

Также следует упомянуть еще раз, что, если файл содержит только код PHP, предпочтительно опустить закрывающий тег в конце файла. Это помогает избежать добавления случайных символов пробела или перевода строки после закрывающего тега PHP, которые могут послужить причиной нежелательных эффектов, так как PHP начинает выводить данные в буфер при отсутствии намерения у программиста выводить какие-либо данные в этой точке скрипта.

## Листинг 2.4. Пример комбинирования PHP и HTML

```
<html>
<head>
<title>Пример комбинации PHP и HTML</title>
</head>
<body>
<h1>Пример простого списка</h1>
<!-- Далее следует список. Это комментарий-->
<ul>
<?php
```

```

echo "<li> Первый элемент списка. <br>";
echo "Продолжение первого элемента списка</li><br>";
// Это однострочный комментарий
# Это тоже однострочный комментарий
echo "<li> <i>Второй элемент списка</i></li>";
/*
А это многострочный комментарий
*/
?>
</ul>
</body>
</html>

```

### 2.2.2. Переменные и типы

В PHP как и во многих других языках существует такое понятие, как переменная. Правилом хорошего тона данного языка является не скупиться на количество переменных в коде, т.к. интерпретатору создать новый идентификатор не затратно. Основные правила, связанные с переменными в языке PHP:

1. Объявление переменной начинается со знака \$. Данная особенность языка облегчает интерпретатору выделение переменных в тексте.
2. Имена переменных должны состоять из латинских букв, цифр и нижнего подчеркивания. Хотя данное правило не является обязательным, рекомендуется его соблюдать.
3. Имена переменных чувствительны к регистру.

PHP поддерживает десять основных типов переменных:

- 1) **integer**: целое число со знаком, размер зависит от разрядности системы;
- 2) **double**: вещественное число, число с плавающей точкой, отводится 8 байт. Для обозначения бесконечности используется INF, а для обозначения несуществующего числа NAN;
- 3) **boolean**: логическая переменная с двумя возможными состояниями true и false;
- 4) **string**: набор символов;
- 5) **array**: ассоциативный массив. В PHP нет обычного классического массива, как в других языках программирования, здесь массив представляет набор элементов, представляющих пары ключ => значение. Ключами могут быть как целые числа, так и строки;
- 6) **object**: ссылка на объект, который реализует несколько принципов ООП;
- 7) **resource**: некоторый ресурс, обрабатываемый языком особым образом;

- 8) **null**: специальное значение;
- 9) **callable**: нововведение версии 5.4. Этим типом является функция обратного вызова. Данный подход используется, когда в функцию нужно передать другую функцию, которая в этом случае и называется функцией обратного вызова;
- 10) **iterable**: псевдотип, введенный в PHP 7.1. Он принимает любой массив или объект, реализующий интерфейс Traversable. Используется как тип параметра для указания, что функция принимает набор значений, но ей не важна форма этого набора, пока он будет использоваться с `foreach`.

Далее приведен листинг 2.5 создания переменных разных типов.

Листинг 2.5. Пример создания различных переменных

```
<?php
$a_bool = true;
$a_bool = TRUE;    // логический
$a_str  = «foo»;   // строковый
$a_str2 = 'foo';   // строковый, следует обратить внимание, что
можно использовать два типа кавычек
$a_int  = 12;      // целочисленный
$a_double = 0.012; // эквивалентно записи 1.2e-2
$a_array = array(
    "foo" => "bar",
    «bar» => «foo»,
    1=>10;
); // массив
?>
```

Создаются две переменные логического типа `a_bool`. Это две разные переменные, т.к. имена переменных чувствительны к регистру, также здесь идет присвоение значения константы ИСТИНА, представленной в двух разных видах. Потом создаются две строковые переменные. На первый взгляд они ничем не отличаются, но следует обратить внимание, что их значения задаются разными типами кавычек. В этом случае это роли не играет, но запомните эту особенность, ее смысл будет рассмотрен чуть позже. Далее создается целочисленная переменная `a_int` и переменная, представляющая число с плавающей точкой. Для типа `double` возможно применение и эквивалентной экспоненциальной записи. Последней в данном примере создается переменная типа массив. Существует несколько типов создания массивов, здесь рассмотрен один из самых тривиальных. Можно заметить, что ключами массива являются как целочисленные значения, так и строки.

В PHP существует множество функций обработки переменных на уровне типов. Рассмотрим основные из них в следующем примере, представленном в

## листинге 2.6.

Листинг 2.6. Пример функций для работы на уровне переменных

```
<?php
$a_bool = TRUE;    // логический
$a_str  = "foo";   // строковый
$a_int  = 12;      // целочисленный
$a_null = null;

// функция gettype выводит тип переменной
echo gettype($a_bool); // выводит: boolean
echo gettype($a_str);  // выводит: string

// функция is_type выводит является ли переменная требуемым
типом
echo is_int($a_bool); // выводит FALSE
echo is_string($a_str); // выводит TRUE

/* функция isset определяет, была ли установлена переменная
значением, отличным от null
*/
echo isset($a_int); // TRUE
echo isset($a_null); // FALSE, переменная существует, но ее
значение не задано
echo isset($a_none); // FALSE, переменной не существует

// функция settype для задания типа переменной
$foo = "5bar"; // строка
$bar = true;   // булево значение
settype($foo, "integer"); // $foo теперь 5    (целое)
settype($bar, "string");  // $bar теперь "1" (строка)

// функция unset удаляет переменную
$per = 1;
unset($per); // удаляет переменную per
echo isset($per); // FALSE

// функция var_dump выводит информацию о переменной
$b = 3.1;
$c = true;
var_dump($b, $c);
/*
float(3.1)
bool(true)
*/
?>
```

В примере создается несколько переменных разных типов. Первой важной функцией является `gettype`. Эта функция возвращает тип, переданной ей переменной. Так как в PHP нет жесткого задания типа переменной, а также применяется динамическая типизация, обозначающая связывание типа переменной с

ней при присвоении ей значения, она важна. Второй необходимой функцией является `is_type`. Она возвращает ИСТИНУ или ЛОЖЬ в зависимости от того, принадлежит переменная типу `type` или нет. Третьей важной функцией является `isset`. Она возвращает ИСТИНУ в случае, если переменная задана каким-либо значением. Чаще всего данная функция используется для проверки существования ключа в массиве. Следующая необходимой функцией является функция `settype`, которая отвечает за преобразование типов. На вход ей подаются ссылка на переменную и тип данных к которой эту переменную нужно преобразовать. В примере строка «5foo» успешно преобразуется в число 5, что по логике не является тривиальным поведением для любого другого языка программирования на уровне базовой работы с типами. Еще одним важным инструментом при написании РНР-скриптов является функция `unset`. Данная функция удаляет переменную. Последняя в данном примере функция это `var_dump`. Результатом данной функции является удобно воспринимаемое человеком представление того, что ей передано с указанием типа и значения. Данная функция полезна в процессе отладки РНР-скриптов. Рекомендуется обратить внимание на то, что в РНР нет четких правил по именованию функций и из-за разработки этих функций разными разработчиками их логика именования отличается.

Возвращаясь к теме переменных в РНР, следует упомянуть существование предопределенных переменных. Любому запускаемому скрипту РНР предоставляет большое количество предопределенных переменных. Однако многие из этих переменных не могут быть полностью задокументированы, поскольку они зависят от запускающего скрипта сервера, его версии и настроек, а также других факторов. Некоторые из этих переменных недоступны, когда РНР запущен из командной строки. РНР предоставляет дополнительный набор предопределенных массивов, содержащих переменные сервера, если они доступны, окружения и пользовательского ввода. Эти массивы являются особыми, поскольку они становятся глобальными автоматически, то есть автоматически доступны в любой области видимости. По этой причине они также известны как “автоглобальные” или “суперглобальные” переменные.

Любопытным инструментом разработки РНР являются переменные переменных. То есть имя переменной, которое может быть определено и изменено динамически. Подробный пример представлен в листинге 2.7.

Листинг 2.7. Пример переменной переменных

```
<?php
// определим обычную переменную
$a = 'hello';
// переменная переменной берет значение переменной и рассмат-
ривает его как имя переменной.
$$a = 'world'; // значение переменной $hello теперь такое
```

```
echo "$a ${$a}"; // hello world
echo "$a $hello"; //hello world
?>
```

В примере создается строковая переменная \$a. Далее создается переменная переменной. То есть объявляется переменная с динамическим именем, хранимым в переменной \$a. Переменная переменной берет значение переменной и рассматривает его как имя переменной. При просмотре данного скрипта можно заметить, что используются фигурные скобки. В данном примере они не являются обязательными, но они важны и уместны при решении проблемы двусмысленности. Наиболее ярким примером является использование переменных при работе с массивами. То есть, если существует запись \$\$a[1], обработчику необходимо знать, использовать ли \$a[1] в качестве переменной, либо использовать как переменную \$\$a, а затем ее индекс [1]. Синтаксис для разрешения этой двусмысленности таков: \${\$a[1]} для первого случая и \${\$a}[1] для второго.

### 2.2.3. Выражения

Выражения – это самые важные строительные элементы PHP. Почти всё, что пишется на PHP, является выражением. Самое простое и точное определение выражения – "все, что угодно, имеющее значение". Самым простым примером выражения является переменная или константа в операторе присваивания. Рассмотрим следующий любопытный пример, представленный в листинге 2.8.

Листинг 2.8. Пример переменной переменных

```
<?php
$a = 5; /* константа 5 является выражением, так как имеет значение 5, и после действия оператора присваивания в переменной $a будет записано значение выражения 5*/

$b = $a; // следует понимать, что следующая запись также является выражением, и это дает право на запись далее

$a = $b = 10; // переменным $a и $b присваивается значение константы 10
$a = 12;
echo "$a $b"; // вывод: 12 10, то есть переменным присвоены константы

$a = 3 * sin($b = $c + 10) + $d; // данная запись также возможна, ниже приведен ее эквивалент

$b = $c + 10; // эквивалент записи выше
$a = 3 * sin($c + 10) + $d;
?>
```

В примере создается переменная \$a. Константа 5 является выражением, так как имеет значение 5, и после действия оператора присваивания в переменную \$a будет записано значение выражения 5. Следует обратить внимание, что запись  $a=b$  не является выражением, поэтому имеет право быть следующая запись:  $a = b = 10$ . В результате переменным  $a$  и  $b$  присваивается значение константы 10. Теперь, когда происходит изменение значения переменной,  $a$  на 12, значение переменной  $b$  останется неизменным: 10. Эта функциональность предполагает следующие действия, представленные далее в примере. Составляется сложное выражение из различных операций и выражений. Эквивалент данного выражения представлен ниже в примере. Составление такого сложно-сочиненного выражения имеет место быть при сокращении кода при необходимости. Когда такие выражения слишком сложные это снижает их дальнейшую читабельность и необходимую проверку.

#### **2.2.4. Операторы и операции**

Оператором называется нечто, принимающее одно или более значений или выражений, если говорить на жаргоне программирования, и вычисляющее новое значение, таким образом вся конструкция может рассматриваться как выражение. Классификация операторов возможна по количеству принимаемых значений (выражений).

Тема приоритетности операторов может быть изучена в официальной документации. Единственной и основной рекомендацией является использование скобок при написании операторов и определении приоритета там, где это необходимо.

##### *Арифметические операторы*

Всего существует 8 основных арифметических операторов, записываемых с помощью 5 символов. Примеры использования операторов представлены в листинге 2.9. Первый оператор – это оператор идентичности, он отвечает за конвертацию в целочисленное или в значение с плавающей точкой в зависимости от того, что больше подходит. Вторым оператором является оператор смены знака. Как видно в примере он работает очень просто и понятно. Следует заметить, что возможны арифметические операции с числами в строковом формате, если возможно их преобразование. Операторы сложения, вычитания и умножения работают по правилам математики, отсюда же идут правила их приоритета. Операция деления работает в зависимости от результата этого деления. Если делится нацело, то возвращается целое число, иначе число с плавающей точкой. Про оператор вычисления остатка от деления нужно помнить, то что он

преобразует операнды к целым числам путем отбрасывания дробной части. Также наряду с классическими арифметическими операторами в РНР присутствует оператор возведения в степень, который работает, как с целыми, так и с числами с плавающей точкой.

#### Листинг 2.9. Примеры арифметических операторов

```
<?php
// оператор идентичности конвертация в int или float, что бо-
лее подходит
$a = "1.1";
$a = +$a;
echo gettype($a); // double

// оператор отрицания, делает смену знака
$a = "1.1";
$a = -$a;
echo gettype($a). " ".$a; // double -1.1

/*
Следует заметить, что возможны арифметические операции с чис-
лами в строковом формате, если возможно их преобразование
*/

// Сложение вычитание умножение - выполнение идет по математи-
ческому приоритету операторов
$a = 3+ 6 * 8 - 10; // 42

// Деление
$a = 10 / 5; // 2 возвращаемое значение является целым, если
делимое и делитель целые и делится все нацело
$b = 10 / 3; // 3.3333 в остальных случаях получаем число с
плавающей точкой

// Целочисленный остаток от деления
$a = 10.9 % 3; // 1 следует отметить, что операнды преобразу-
ются в целые числа путем удаления дробной части

// Возведение в степень
$a = 2.1 ** 0.5; // 1.4491... допускается использование как це-
лых так и дробных выражений
?>
```

#### *Битовые операции*

Вторыми базовыми операциями являются битовые операции, представленные широким спектром в РНР. Побитовые операторы позволяют считывать и устанавливать конкретные биты целых чисел. Классические битовые операторы И, ИЛИ, ИСКЛЮЧАЮЩЕЕ ИЛИ, ОТРИЦАНИЕ и конечно же БИТОВЫЙ СДВИГ. Примеры битовых операций представлены в примере в листинге 2.10.



## Листинг 2.10. Пример побитовых операций

```
<?php
// И - результат число с установленными битами, которые вы-
ставлены в обоих операндах одновременно
$a = 1 & 2; // 0

// ИЛИ - результат число с установленными битами, которые вы-
ставлены хотя бы в 1 операнде
$b = 12 || 2; // 14

// ИСКЛЮЧАЮЩЕЕ ИЛИ или СЛОЖЕНИЕ ПО МОДУЛЮ 2 - результат число
с установленными битами, которые выставлены либо в 1 операнде
либо во 2
$a = 15 ^ 3; // 14

// ОТРИЦАНИЕ - результат число с инвертированными битами
$a = 10;
$a = ~$a;

// БИТОВЫЙ СДВИГ - результат число, полученное поразрядным
сдвигом битов вправо или влево
$a = 2<<2; // 8
$b = 2>>2; // 0
?>
```

### *Строковые операции*

В PHP существует один оператор для работы именно со строками. Это оператор конкатенации строк “.”. Пример представлен в листинге 2.11. Сложение строк также работает, но рекомендуется использовать конкатенацию, как специальный оператор объединения строк. Так как при сложении строк, содержащих числа произойдет преобразование их к числовым типам и арифметическое сложение.

## Листинг 2.11. Пример строкового оператора

```
<?php
$a = "Привет, ";
$b = $a . "Мир!"; // $b теперь содержит строку "Привет, Мир!"
?>
```

### *Оператор присваивания*

Отдельным разделом рассматриваются возможные операции с оператором присваивания. Основным является оператор простого присваивания “=”. Оператор присваивания означает, что левый операнд получает значение правого выражения, то есть устанавливается значением. Результатом выполнения оператора присваивания является само присвоенное значение. Таким образом, результат выполнения “\$a = 3” будет равен 3. В дополнение к базовому оператору присваивания имеются “комбинированные операторы” для всех бинарных

арифметических операций, операций объединения массивов и строковых операций, которые позволяют использовать некоторое значение в выражении, а затем установить его как результат данного выражения. Различные возможные действия с оператором присваивания представлены в листинге 2.12.

Листинг 2.12. Примеры с оператором присваивания

```
<?php
$a = ($b = 4) + 5; // $a теперь равно 9, а $b было присвоено
4.

$a = 3;
$a += 5; // устанавливает $a в 8, как если бы мы написали: $a
= $a + 5;
$b = "Привет";
$b .= "-привет!"; // устанавливает $b в "Привет-привет!", как
и $b = $b . "-привет!";
?>
```

### *Операторы сравнения*

Операторы сравнения представляют собой уникальные операции, независимо от типа операндов, возвращающие одно из 2 значений: true или false. Полный список данных операторов представлен в следующей табл. 2.1. Далее будут даны комментарии по данной таблице. Оператор равно расширяется оператором тождественно равно, отличаются они тем, что последний учитывает также и тип. Например, при сравнении одного и того же числа в строковом и числовом представлении, оператор равно выдаст ИСТИНУ, когда оператор тождественно равно выдаст ЛОЖЬ. Аналогично работают операторы не равно и тождественно не равно. Любопытной функциональностью обладает оператор «Космический корабль»: число типа int меньше, больше или равно нулю, когда а соответственно меньше, больше или равно b. В случае, если оба операнда являются строками, содержащими числа или один операнд является числом, а другой - строкой, содержащей числа, то сравнение выполняется численно. Преобразование типа не происходит при сравнении === или !==, поскольку функциональность включает сравнение типа операндов, а также значения. Два разных по обозначению оператора не равно различаются приоритетом.

Таблица 2.1. Операторы сравнения

Название	Обозначение	Описание
Равно	<code>\$a == \$b</code>	true если \$a равно \$b после преобразования типов.

### Окончание табл. 2.1.

Тождественно равно	<code>\$a === \$b</code>	true если \$a равно \$b и имеет тот же тип.
Не равно	<code>\$a != \$b</code>	true если \$a не равно \$b после преобразования типов.
Не равно	<code>\$a &lt;&gt; \$b</code>	true если \$a не равно \$b после преобразования типов.
Тождественно не равно	<code>\$a !== \$b</code>	true если \$a не равно \$b, или они разных типов.
Меньше	<code>\$a &lt; \$b</code>	true если \$a строго меньше \$b.
Больше	<code>\$a &gt; \$b</code>	true если \$a строго больше \$b.
Меньше или равно	<code>\$a &lt;= \$b</code>	true если \$a меньше или равно \$b.
Больше или равно	<code>\$a &gt;= \$b</code>	true если \$a больше или равно \$b.
Космический корабль (spaceship)	<code>\$a &lt;=&gt; \$b</code>	Число типа int меньше, больше или равное нулю, когда \$a соответственно меньше, больше или равно \$b.

### Операторы инкремента и декремента

Отдельно следует упомянуть, что PHP поддерживает префиксные и постфиксные операторы инкремента и декремента в стиле языка программирования C. Пример использования данных операторов представлен в листинге 2.13. При использовании постфиксного инкремента и декремента сначала идет работа со значением переменной, а потом операция сложения или вычитания единицы. При использовании префиксного инкремента и декремента сначала идет операция, а потом уже работа со значением переменной.

Листинг 2.13. Пример префиксных и постфиксных операторов

```
<?php
// Постфиксный инкремент и декремент
$a = 5;
echo $a++; // 5 сначала идет вывод, потом уже операция прибавления 1
echo $a--; // 6 сначала идет вывод, потом уже операция вычитания 1

// Префиксный инкремент и декремент
$a = 5;
echo ++$a; // 6 сначала идет операция прибавления 1, потом вы-
```

```
вод
echo --$a; // 5 сначала идет операция вычитания 1, потом вывод
?>
```

### *Логические операторы*

В PHP также поддерживаются логические операторы. Это операции предназначенные для работы с логическими выражениями и возвращают булево значение: true или false. На основе табл. 2.2 видно, что существует не только тривиальные операторы, но и по два оператора для представления операций И и ИЛИ. Особенностью выбора одного или другого является различие их приоритетов.

Таблица 2.2. Логические операторы

Название	Обозначение	Описание
И	\$a and \$b	true, если и \$a, и \$b true.
И	\$a && \$b	true, если и \$a, и \$b true.
ИЛИ	\$a or \$b	true, если или \$a, или \$b true.
ИЛИ	\$a    \$b	true, если или \$a, или \$b true.
Отрицание	! \$a	true, если \$a не true.
Исключающее ИЛИ	\$a xor \$b	true, если \$a, или \$b true, но не оба.

### *Операторы, работающие с массивами*

Операторы сравнения массивов и другие возможности работы с ними часто в языке полностью отсутствуют или являются только дополнительными функциями. Так как в PHP массив является стандартным типом, то он имеет операторы для работы с ним. Данные операторы представлены в следующей табл. 2.3.

Таблица 2.3. Операторы работы с массивами

Название	Обозначение	Описание
Объединение	\$a + \$b	Объединение массива \$a и массива \$b.
Равно	\$a == \$b	true в случае, если \$a и \$b содержат одни и те же пары ключ/значение.

Окончание табл. 2.3.

Тождественно равно	<code>\$a === \$b</code>	true в случае, если \$a и \$b содержат одни и те же пары ключ/значение в том же самом порядке и того же типа.
Не равно	<code>\$a != \$b</code>	true, если массив \$a не равен массиву \$b.
Не равно	<code>\$a &lt;&gt; \$b</code>	true, если массив \$a не равен массиву \$b.
Тождественно не равно	<code>\$a !== \$b</code>	true, если массив \$a не равен тождественно массиву \$b.

Из табл. 2.3 видно, что, как и в случае с логическими операторами, существует два представления оператора “Не равно”. Они различаются также приоритетом.

### 2.2.5. Управляющие конструкции

Любой сценарий PHP состоит из последовательности инструкций. Инструкцией может быть присваивание, вызов функции, повтор кода (цикл), сравнение, или даже инструкция, которая ничего не делает (пустой оператор). После инструкции обычно ставится точка с запятой. Кроме того, инструкции могут быть объединены в блоки заключением их в фигурные скобки. Блок инструкций также сам по себе является инструкцией. Далее рассматриваются различные типы инструкций.

#### *Условная конструкция*

В листинге 2.14 представлен пример условной конструкции. Данная конструкция предоставляет возможность условного выполнения фрагментов кода. Структура if реализована в PHP по аналогии с языком C. Нужно понимать, что выражение else выполняется только, если выражение if вычисляется как false, и если нет других любых выражений elseif, или если они все равны false. Конструкция elseif, есть сочетание конструкций if и else. Аналогично else, она расширяет оператор if для выполнения различных выражений в случае, когда условие начального оператора if эквивалентно false. Однако, в отличие от else, выполнение альтернативного выражения произойдет только тогда, когда условие оператора elseif будет являться равным true.

Листинг 2.14. Пример условного оператора

```
<?php
if ($a > $b) {
```

```

        echo "a больше, чем b";
    } elseif ($a == $b) {
        echo "a равен b";
    } else {
        echo "a меньше, чем b";
    }
?>

```

### *Циклы*

Для общего понимания введем следующее определение: Цикл – разновидность управляющей конструкции в высокоуровневых языках программирования, предназначенная для организации многократного исполнения набора инструкций.

#### *While*

Циклы `while` являются простейшим видом циклов в РНР. Они ведут себя так же, как и в языке С. Данная конструкция предполагает циклическое выполнение какого-то участка кода с предварительной проверкой условия на необходимость выполнения. Простой пример вывода всех чисел от 1 до 10 с использованием цикла `while` в листинге 2.15.

Листинг 2.15. Пример цикла `while`

```

<?php
$i = 1;
while ($i <= 10) {
    echo $i++;
    /* выводиться будет значение переменной $i перед её увеличени-
    ем (post-increment) */
}
?>

```

#### *Do-while*

Цикл `do-while` очень похож на цикл `while`, с тем отличием, что истинность выражения проверяется в конце итерации, а не в начале. Главное отличие от обычного цикла `while` в том, что первая итерация цикла `do-while` гарантированно выполнится (истинность выражения проверяется в конце итерации), тогда как она может не выполниться в обычном цикле `while` (истинность выражения которого проверяется в начале выполнения каждой итерации, и если изначально имеет значение `false`, то выполнение цикла будет прервано сразу). Пример данной конструкции представлен в листинге 2.16.

Листинг 2.16. Пример цикла `do-while`

```

<?php
// вывод чисел от 0 до 10
$i = 0;
do {
    echo $i;
} while ($i < 10);

```

```
} while ($i < 10);  
?>
```

### Универсальный цикл *for*

Вспоминая уже не в первый раз такой язык программирования как Си, то данный тип цикла чаще всего используется для перебора значений счетчика и каких-либо действий с использованием данного счетчика. Данный же цикл назван универсальным, т.к. предполагает создавать не такие тривиальные действия. Общий вид данной конструкции таков:

for (инициализирующие команды; условие цикла; команды после прохода)  
    тело цикла;

Пример использования данной конструкции представлен в листинге 2.17.

Листинг 2.17. Пример универсального цикла *for*

```
<?php  
for($i = 0, $j = 0, $k = "Points"; $i<100; $j++, $i += $j) $k  
= $k.".";   
// первыми выполняются единожды инициализирующие команды  
// потом проверяется условие цикла, в данном примере оно про-  
стое  
// если условие истинно, то выполняется итерация цикла  
// после выполнения итерации цикла выполняются команды после  
прохода  
// в данном примере нет тела цикла, т.к. в теле цикла всего 1  
команда  
  
// пример вывода чисел от 0 до 10 с использованием цикла for  
for($i = 0; $i < 10; $i++){  
    echo $i;  
}  
?>
```

### Инструкции *break* и *continue*

Инструкции *break* и *continue* знакомы по практическому любому другому языку программирования, используемые для более тонкого управления циклами. Инструкция *break* используется, чтобы выйти из циклов или структур: *for*, *foreach*, *while*, *do-while*, *switch*. Но конструкция *break* не так тривиальна, как в других языках. В PHP расширена ее функциональность за счет работы с уровнем вложения.

Инструкция *continue* работает только с циклическими конструкциями для пропуска оставшейся части итерации цикла и, при соблюдении условий, начала следующей итерации. Как и *break*, *continue* принимает необязательный числовой аргумент, который указывает на скольких уровнях вложенных циклов будет пропущена оставшаяся часть итерации. Значением по умолчанию является 1, при которой пропускается оставшаяся часть текущего цикла. Пример исполь-

зования данных конструкций представлен в листинге 2.18.

Листинг 2.18. Пример с использованием инструкции break и continue

```
<?php
    $arr = array('один', 'два', 'три', 'четыре', 'стоп',
'пять');
    for ($i = 0; $i < count($arr); $i++) {
        if ($arr[$i] == 'стоп') {
            break;        // эквивалентно записи break 1; или
break(1);
        }
        echo "{$arr[$i]}<br />\n";
    }

    // инструкция break принимает необязательный числовой па-
раметр - из какого вложенного цикла должен быть произведен вы-
ход

    $matrix = array (
        array (1, 2, 3, 6, 8, 9),
        array (4, 7, 3, 3, 1, 5)
    );
    for ($i = 0; $i < count($matrix); $i++){
        for ($j = 0; $j < count($matrix[$i]); $j++){
            if ($matrix[$i][$j] == 0) break(2); // будет про-
изведен выход из всего вложенного цикла полностью, если бы па-
раметр не был бы подан, то выход был бы совершен только из
внутреннего цикла for
        }
    }
    $i = 0;
    while ($i++ < 5) {
        echo "Снаружи<br />\n";
        while (1) {
            echo "В середине<br />\n";
            while (1) {
                echo "Внутри<br />\n";
                continue 3;
            }
            echo "Это никогда не будет выведено.<br />\n";
        }
        echo "Это тоже.<br />\n";
    }
?>
```

### *Foreach*

В PHP существует и особый тип цикла для перебора элементов массива и итерируемых объектов. Следует напомнить, что массив есть встроенный тип для PHP и представляет во всех случаях набор пар ключ-значение. Существует два синтаксиса данной конструкции в зависимости от желания игнорировать ключ элемента в массиве или нет. Важным уточнением является то, что кон-



структура оперирует копией массива, поэтому изменение возможно, но только при использовании ссылок. Синтаксис и применение данной конструкции представлен в следующем листинге 2.19.

Листинг 2.19. Пример цикла foreach

```
<?php
// вывод всех переменных окружения
foreach($_SERVER as $key => $value){
    echo "<b>$k</b> => <tt>$value</tt><br />\n";
}

// вывод всех элементов списка
// под списком понимается массив, где ключами являются последовательные
// числовые индексы
$array = array("foo", "bar", "hallo", "world");
foreach($array as $value){
    echo "$value<br />\n";
}
?>
```

### *Switch*

В ситуации сложных и многоуровневых условных конструкций на помощь приходит конструкция switch-case. Данная конструкция используется, когда требуется сравнивать одну и ту же переменную (или выражение) с множеством различных значений и выполнять различные участки кода в зависимости от того, какое значение принимает эта переменная (или выражение). Использование if+ifelse+else создаст излишнюю громоздкость данного участка кода. Пример использования данной конструкции представлен в листинге 2.20.

Листинг 2.20. Пример использования конструкции switch-case

```
<?php
// подается на вход выражение, которое вычисляется 1 раз
// далее идет сравнение с каждым возможным оператором case
// если сравнение истинно, то данному оператору передается управление
switch ($i) {
    case 0:
        echo "i равно 0";
        break;
    case 1:
        echo "i равно 1";
        break;
    case 2:
        echo "i равно 2";
        break;
    default:
        echo "i не равно 0, 1 и 2";
}
}
```

```
// использование default возможно, когда существуют ситуации,
// когда
// управление может не перейти ни одному из операторов case,
// тогда
// управление переходит оператору default
?>
```

Следует дополнительно отметить, что сравнение в switch конструкции идет нестрогое, а также возможно, как выражение использовать строку.

### *Match*

В версии PHP 8.0.0 появилась новая конструкция для ветвления потока исполнения на основании проверки совпадения значения с заданным условием. Конструкция аналогична switch, но имеет некоторые существенные различия, которые способствуют применению данной конструкции. Главным отличием от switch является возвращение результата, а также привязка сравнения также к типам. Подробный пример данной конструкции представлен в листинге 2.21.

Листинг 2.21. Пример использования конструкции match

```
<?php
$expressionResult = match ($condition) {
    1, 2 => foo(),
    3, 4 => bar(),
    5 => 5,
    default => baz(),
}

// удобное использование для вхождения в диапазоны чисел
$result = match (true) {
    $age >= 65 => 'elderly',
    $age >= 25 => 'adult',
    $age >= 18 => 'full age',
    default => 'child',
};
?>
```

### *Include и Require*

Идея разделения программы на несколько файлов очень важна для переиспользования кода, создания шаблонов и т.д. Первое, о чем нужно сказать в данном разделе это задание путей поиска файлов для PHP. Это задается директивой `include_path` в файле `php.ini`, указывается список директорий, в которых функции `require`, `include`, `fopen()`, `file()`, `readfile()` и `file_get_contents()` ищут файлы. Формат соответствует формату системной переменной окружения `PATH`: список директорий, разделенных двоеточием в Unix или точкой с запятой в Windows. Возможно добавление в ходе выполнения сценария еще какой-либо директории с помощью функции `set_include_path()`. Подробнее можно узнать в официальной документации.

Итак, выражение `include` включает и выполняет указанный файл. Файлы включаются исходя из пути указанного файла, или, если путь не указан, используется путь, указанный в директиве `include_path`. Если файл не найден в `include_path`, `include` попытается проверить директорию, в которой находится текущий включающий скрипт и текущую рабочую директорию перед тем, как выдать ошибку. Конструкция `include` выдаст `E_WARNING`, если не сможет найти файл.

В отличие от выражения `include` выражение `require` при невозможности добавления файла останавливает выполнение скрипта и выдает фатальную ошибку уровня `E_COMPILE_ERROR`.

Использование выражений `require` и `include` оправдывает себя при формировании HTML-шаблонов. Когда повторное добавление одного и того же файла точно не вызовет фатальных ошибок. Для того, чтобы не было ошибок существуют выражения `require_once` и `include_once`. Эти инструкции второй раз не срабатывают в одном файле или при более сложной иерархии. Например, есть файл `func.php` в нем определена функция `my_func`. Чтобы ее использовать в файле `file.php` нужно включить этот с помощью инструкции `require_once('func.php')`. Существует еще один файл, где будет использоваться `my_func` из файла `func.php` и функция `find_file` из файла `file.php`. Может быть неизвестно, что файл `func.php` уже включен в файл `file.php`. Поэтому, чтобы обезопасить код и не определять функцию `my_func` второй раз используется инструкция `require_once`.

#### *Альтернативный синтаксис управляющих структур*

Возможно использование альтернативного синтаксиса управляющих структур для ситуации вставки HTML-кода в тело сценария. Исчерпывающие примеры некоторых доступных альтернативных конструкций представлены в листинге 2.22.

Листинг 2.22. Примеры использования альтернативного синтаксиса

```
<?php if (condition): ?>

// HTML-код для ситуации, когда выражение condition истинно

<?php else: ?>

// HTML-код для ситуации, когда выражение condition ложно

<?php endif ;?>

<?php while(condition):?>

// HTML-код для ситуации, когда выражение condition истинно

<?php endwhile;?>
```

### 2.2.6. Функции

Функции являются неотъемлемой частью любого языка программирования, и PHP не исключение. Для понимания синтаксиса описания функций и их использования нужно понимать, что:

- 1) у функции может быть переменное количество параметров и параметры по умолчанию;
- 2) функция имеет собственную область видимости. Это нужно учитывать при работе с глобальными переменными программы и с локальными переменными функции, которые уничтожаются при окончании ее работы;
- 3) в PHP тип возвращаемого значения может быть любым и тип возвращаемого значения не регламентируется. То есть функция может возвращать и число, и строку одновременно;
- 4) допускается создание и использование при необходимости анонимных функций;
- 5) в пределах одного сценария не должно быть определений двух функций с одинаковыми именами.

Общий вид синтаксиса функций, а также простой пример представлен в листинге 2.23.

Листинг 2.23. Пример определения функции

```
<?php
/*
function nameOfFunction($arg1[=val1], $arg2[=val2], ...,
$argN[=valN]){
    function body operators;
}
*/
// пример простой функции
function plus($a, $b){
    return $a + $b;
}
// все будет корректно работать если вы будете подавать кор-
ректные значения
?>
```

Так как синтаксис PHP не требует обозначения типа параметров и возвращаемого значения, то могут возникать различные неточности и даже ошибки при работе функций. Но данный функционал предусмотрен и его можно использовать при необходимости. Возможная модернизация с использованием данного функционала листинга 2.23 представлена в листинг 2.24.

#### Листинг 2.24. Пример типизации аргументов и возвращаемого значения

```
<?php
function plus(int $a, int $b) : int
{
    return $a + $b;
}
?>
```

Теперь при подаче в функцию не целочисленного значения до 7 версии возникает фатальная ошибка, которая полностью прекращает работу скрипта, а после 7 версии возникает ошибка типа, которую можно перехватить.

Возвращаясь к вопросу об области видимости при использовании функций в РНР. В данном языке программирования предусмотрено три типа переменных: локальные, глобальные и как особый тип статические. Поведение локальных переменных понятно и они в совокупности называются контекстом функции или областью видимости внутри функции. Пример использования глобальных и статических переменных представлен в листинге 2.25.

#### Листинг 2.25. Пример использования глобальной и статической переменных

```
<?php
$numbers = [
    0 => "zero",
    1 => "one",
    2 => "two",
    // ..
    9 => "nine"
];
// возвращает строковое представление цифры
function getNameOfNum($n) {
    global numbers;
    return $numbers[$n];
}

// функция подсчета кол-ва ее вызовов
function getNumCount() {
    static $count = 0;
    $count++;
    return $count;
}
?>
```

Вспоминая особенности синтаксиса функций в РНР, обратим внимание на такой инструмент, как анонимная функция. Рассмотрим пример в листинге 2.26.

#### Листинг 2.26. Пример использования анонимной функции

```
<?php
$greet = function($name)
{
```

```

    printf("Привет, %s\r\n", $name);
};

$greet('Мир');
$greet('PHP');
?>

```

## 2.3. Объектно-ориентированное программирование на PHP

### 2.3.1. Важное уточнение: ссылки

Каждый язык программирования обладает своими особенностями, ознакомление с которыми помогает в понимании идей и алгоритмов работы языка программирования. Одним из важных для понимания инструментов в PHP являются ссылки. При желании возможно проведение аналогии с указателями в языке программирования Си, но данная мысль неверна из-за разной реализации и разной логики данных технологий. Реализация ссылок в PHP является средством доступа к содержимому одной переменной под разными именами. Ссылки в PHP - аналог жёстких ссылок в файловых системах Unix. В следующем листинге 2.27. представлены основные операции, связанные с использованием ссылок.

Листинг 2.27. Основные операции и использованием ссылок

```

<?php
// первая операция
$b = 5;
$a =& $b;

// вторая операция
function example(&$var) {
    $var++;
}

$d = 5;
example($d);

// третья операция
class example {
    public $val = 42;

    public function &getVal() {
        return $this->val;
    }
}

$params = new example;
$myValue = &$params->getVal();
$params->val = 2; // $myValue = val = 2

```

Первой операцией, представленной в листинге 2.27, является операция присвоение по ссылке. Переменная `b` имеет значение 5. Далее переменной `a` присваивается ссылка на значение переменной `b`. Теперь переменные `$a` и `$b` указывают на одно и то же значение и изменение одной из переменных не приведет к изменению другой. Пример в листинге 2.27. недостаточно показателен по той причине, что изменение константы 5 невозможно. В дальнейшем при рассмотрении основ объектно-ориентированного программирования будет рассмотрен более подробный пример. А другие особенности, связанные с присвоением по ссылке других встроенных типов, рекомендуется рассмотреть в официальной документации языка программирования PHP.

Второй операцией, представленной в листинге 2.27, является передача параметров по ссылке. Данная функциональность используется в функциях. В функцию `example` передается ссылка на целочисленное значение. Происходит изменение данного целочисленного значения. В итоге значение переменной в глобальной области также изменилось, так как локальная переменная для функции `example` и глобальная `$d` ссылаются на одно и то же значение.

Третьей операцией, представленной в листинге 2.27, является возврат по ссылке. Перед разъяснениями данной функциональности следует упомянуть, что данная операция со ссылками часто используется для оптимизации, но в PHP оптимизация происходит на уровне ядра, и такая искусственная оптимизация никогда не приводит к удачному результату. Возврат по ссылке предполагает связывание функции для выбора переменной, с которой должна быть связана ссылка. В примере в листинге 2.27 представлен класс `example`, имеющий одно публичное свойство `$val`, равное по умолчанию 42. Также данный класс имеет метод. для получения своего единственного публичного свойства. Далее создается новый экземпляр класса `example` и переменная `$myValue` теперь ссылается на свойство объекта, возвращаемого функцией его получения, а не его копии. Теперь при изменении значения свойства объекта изменится, как и результат значения переменной `$myValue`.

### **2.3.2. Общие принципы**

При изучении реализации ООП на PHP часто возникает много вопросов, особенно при сравнении объектно-ориентированного ядра PHP, например с объектно-ориентированным ядром такого известного объектно-ориентированного языка программирования, как Java. Следует понимать, что PHP изначально создавался как скриптовый язык для динамического создания веб-страниц. Разработчиками было реализовано простое объектно-

ориентированное ядро, которое поддерживало самые простые основы ООП с некоторыми уместными искажениями (на современный лад) для авторов РНР.

Примером такого искажения (на современный лад) является то, что в первых версиях была передача объектов не по ссылке, а по значению. То есть, при использовании тривиальной записи `$b = $c` создавалась новая копия объекта, содержащегося в переменной `$c`. В последующих версиях, а точнее в релизе РНР 5 данная функциональность была изменена.

В итоге каждая новая версия РНР содержит какие-либо исправления, связанные с реализацией ООП в РНР, что подключает к реализации объектов не всегда тривиальную функциональность, которая будет рассмотрена в данном разделе позднее.

В настоящий момент (последним релизом является РНР 8) работа с объектами идет так же, как и со ссылками. Что означает, что каждая переменная содержит ссылку на объект. Такая переменная содержит только идентификатор объекта, который позволяет найти конкретный объект при обращении к нему. Когда объект передается как аргумент функции, возвращается или присваивается другой переменной. Они содержат копию идентификатора, который указывает на один и тот же объект.

Рассмотрим далее определение класса на основе парадигмы ООП в РНР. В следующем листинге 2.28 представлено определение простого класса.

Листинг 2.28. Пример определения класса на РНР

```
<?php
class Example{
    public $property = "simple property";

    public function exampleMethod(){
        echo "Hello, world!";
    }

    public function setProperty($property){
        $this->property = $property;
    }
}
```

В примере определяется класс `Example`. Правила именования классов в РНР совпадают с правилами именования переменных: имя не входит в список зарезервированных слов и состоит из букв, цифр и символов нижнего подчеркивания. Класс `Example` содержит одно свойство `$property` типа строка и два метода: `exampleMethod` и `setProperty`. Первый метод содержит вывод текстового сообщения "Hello, world!". Второй метод задает новое значение свойства класса `$property`. `$this` является псевдопеременной, данная переменная доступна толь-



ко в контексте метода класса, вызванного от объекта (подробнее об этом будет расписано далее). `$this` - значение вызывающего объекта. После определения класса предполагается создание его экземпляра, данный этап описан в листинге 2.29:

Листинг 2.29. Пример создания экземпляра класса

```
<?php
$instance = new Example; // эквивалентно new Example();
$instance->exampleMethod();
echo $instance->property; // "simple property"
$instance->setProperty("new string");
echo $instance->property; // "new string"
```

В данном примере создается экземпляр класса `Example`. Главной особенностью является то, что если конструктор не принимает параметров, то круглые скобки можно опустить. Для того, чтобы вызвать методы класса нужно прописать название переменной, содержащей ссылку на экземпляр класса, знак `"->"` и далее вызываемый метод, также можно получить доступ к свойствам класса. Таким образом вызывается метод `exampleMethod`, выводящий информационное сообщение, а также метод, изменяющий значение свойства класса `property`. Важно упомянуть, что имена свойств класса и методов находятся в различных пространствах имен класса, поэтому возможно иметь метод и свойство класса с одинаковыми именами. Учитывая, что PHP иногда обладает довольно необычной функциональностью, то создание нового экземпляра класса можно реализовать образом, представленным в листинге 2.30.

Листинг 2.30. Пример создания класса на основе имени класса

```
<?php
$classname = "Example";
$instance = new $classname(); // эквивалентно new Exam-
ple();
```

В примере, представленном в листинге 2.30 разбирается создание нового экземпляра класса `Example` с помощью переменной, которая хранит имя класса в виде строки.

### 2.3.3. Область видимости

Как и в любом языке программирования, реализующим парадигму ООП, существует понятие области видимости для элементов класса. Область видимости для всех элементов класса: констант, свойств и методов, – определяется с помощью ключевых слов `public`, `protected` или `private`. Данная функциональность введена только с версии PHP 7.1.0. Логика данных модификаторов доступа повторяет логику реализации в других языках программирования: публич-

ные (public) свойства и методы класса доступны отовсюду, где есть упоминание данного класса, защищенные (protected) методы и свойства класса доступны внутри класса, родителя или наследника, закрытые (private) доступны только в контексте класса. Важно упомянуть, что объявление области видимости свойств обязательно, а вот для методов необязательно, отсутствие модификатора доступа будет означать публичный вариант доступа. В сравнении с языком Java, где отсутствие модификатора означает пакетный вариант доступа для элемента. Пример с различными уровнями области видимости представлен в листинге 2.31.

Листинг 2.31. Пример использования модификаторов доступа

```
<?php
class Woman{
    public string $name;
    private string $birthday;

    protected function countAge(): int|string
    {
        $birthday_timestamp = strtotime($this->birthday);
        $age = date('Y') - date('Y', $birthday_timestamp);
        if (date('md', $birthday_timestamp) > date('md')) {
            $age--;
        }
        return $age;
    }

    public function sayAge(){
        echo $this->countAge();
    }

    public function setBirthDay($birthday){
        $this->birthday = $birthday;
    }
}

$alice = new Woman;
$alice->setBirthDay("1995-04-01");
$alice->sayAge();
```

В примере в листинге 2.31 представлен пример объявления класса Woman. Данный класс содержит два основных свойства открытое свойства имя и закрытое свойство дата рождения. Также данный класс содержит защищенный метод подсчета возраста на основе свойства 'дата рождения'. Данный метод закрытый и доступен только внутри класса - его использует метод sayAge для вывода возраста. Также следует заметить, что при объявлении свойств класса помимо модификаторов доступа используется жесткое задание типа

### 2.3.4. Конструкторы и деструкторы

Работа с объектами классов предполагает некий жизненный цикл, состоящий из инициализации, функционирования и разрушения. Возвращаясь к листингу 2.31 можно заметить, что инициализация значения свойства, отвечающего за дату рождения, производится с помощью специального публичного метода, что в данном случае избыточно по причине неизменности данного параметра в процессе существования экземпляра класса. Логичнее было бы инициализировать свойства в конструкторе и PHP предоставляет в рамках реализации парадигмы ООП данную функциональность. Измененный вариант класса Woman представлен в листинге 2.32.

Листинг 2.32. Измененный класс Woman

```
<?php
class Woman{
    public string $name;
    private string $birthday;

    protected function countAge(): int|string{...}
    public function sayAge(){...}

    public function __construct($name, $birthDay){
        $this->birthday = $birthDay;
        $this->name=$name;
    }
}

$alice = new Woman("Alice", "1995-04-01");
$alice->sayAge();
```

В примере представлен модифицированный класс Woman с конструктором, которой создается за счет определения функции `__construct`. Все специальные методы для класса начинаются с двух символов нижнего подчеркивания, данные методы принято называть магическими. Они будут рассмотрены далее в этой главе. Похожая функциональность с аналогичным названием представлена при реализации объектно-ориентированной парадигмы в языке программирования Python. Важно помнить о том, что в PHP нет возможности определения нескольких методов с разными параметрами для одного класса, как в языке программирования Java. Поэтому невозможно определение множественных конструкторов с переменным количеством параметров. Частичной заменой данной функциональности является присвоение значения свойствам при их объявлении (см. пример в листинге 2.28) и применение значений параметров по умолчанию в конструкторе. Но, к сожалению, данные методы не решают полностью проблему необходимости множественных конструкторов.

Следует обратить внимание на то, что у конструктора измененного класса Woman обозначен модификатор доступа public. У конструктора может быть любой из возможных модификаторов доступа, если этого требует реализуемая функциональность. Пример представлен в листинге 2.33.

Листинг 2.33. Пример класса с приватным конструктором

```
<?php
class Woman{
    public static $women = [];
    public string $description;

    private function __construct($desc){
        $this->description = $desc;
    }

    public static function create($desc){
        if (isset(self::$women[$desc])) return
        self::$women[$desc]);
        return self::$women[$desc] = new self($desc);
    }
}
```

В примере представлен вновь модифицированный класс Woman, содержащий конструктор с закрытым модификатором. Данное явление называется статическим методом создания объектов. Кроме закрытого конструктора класс имеет открытый статический метод create, который отвечает в свою очередь за создание нового экземпляра класса только в случае уникального описания. Логика применения статических свойств и методов класса аналогична другим языкам программирования, реализующим парадигму ООП. Существуют лишь особенности, связанные с использованием оператора разрешения видимости — ::, который будет рассмотрен уже в следующем разделе. Более подробный и показательный пример использования статического метода создания объектов представлен в листинге 2.34.

Листинг 2.34. Использование статического метода создания объектов

```
<?php
class Person {

    private int $id;
    private string $name;

    private function __construct(int $id = 0, string $name =
    "") {
        $this->id = $id;
        $this->name = $name;
    }
}
```

```

        public static function fromBasicData(int $id, string
$name): static {
            return new static($id, $name);
        }

        public static function fromJson(string $json): static {
            $data = json_decode($json);
            return new static($data['id'], $data['name']);
        }
    }

    $p1 = Product::fromBasicData(5, 'Alex');
    $p2 = Product::fromJson($some_json_string);

```

В листинге представлен закрытый конструктор и два статических метода создания объектов: один на основе подачи двух параметров и второй на основе строки в формате json. Ключевое слово `static`, используемое в паре с оператором `new` транслируется в имя класса, в котором этот код вызывается. В примере конструктор защищен от прямого вызова во избежание возможных ошибок и создания использования множественных конструкторов (как часто используется в языке программирования Java). Статические методы класса полноценные и имеют доступ ко всем методам и свойствам класса при необходимости и их использование опционально потребностям.

Новой функциональностью, доступной с PHP 8.0.0. является определение свойств объекта в конструкторе. Теперь параметры конструктора можно использовать для задания соответствующих свойств объекта. Когда в конструкторе содержатся только операции присвоения свойствам переданных параметров как в листинге 2.31, получается некий громоздкий код с тривиальными операциями. Новый вариант класса `Woman` из листинга 2.31. представлен в листинге 2.35.

Листинг 2.35. Модифицированный класс `Woman` из листинга 2.31

```

<?php
class Woman{
    protected function countAge(): int|string{...}
    public function sayAge(){...}

    public function __construct(public string $name, private
string $birthDay){}
}

```

Когда в месте указания параметров конструктора идет не только сочетание имени и типа, но и указывается модификатор типа, то данный параметр интерпретируется как новое свойство класса. При этом не обязательно при использовании данной функциональности объявлять все свойства класса как параметры и делать конструктор пустым. Код конструктора выполнится после того, как

все аргументы присвоятся всем соответствующим свойствам.

В отличие от других методов, `__construct()` освобождается от обычных правил совместимости сигнатуры при наследовании. Данные правила будут рассмотрены далее.

Самым частым примером при упоминании использования деструктора класса для уничтожения подключения к базе данных для классов управления доступа к базам данных. Вопрос освобождения ресурсов очень важен при использовании языка программирования PHP, так как у него нет прямого управления памятью как у C++. Главная задача деструктора – освобождение ресурсов и эту задачу он выполняет при удалении всех ссылок на объект. Пример деструктора представлен в листинге 2.36.

Листинг 2.36. Пример класса с деструктором

```
<?php
class FileWriter{
    private $file;
    private $filepath;

    public function __construct($path){
        $this->filepath=$path;
        $this->file=fopen($path, "a+");
    }

    public function write($string){
        fputs($this->file, $string);
    }

    public function __destruct(){
        fclose($this->file);
    }
}
```

В примере создается класс для записи данных в файл. Параметром конструктора является путь к файлу, функциональность конструктора предполагает открытие файла на запись в конец и сохранение ресурса файла в свойстве класса. При удалении всех ссылок на созданный экземпляр класса вызывается деструктор, который закрывает открытый файл. По идее можно не закрывать файл и при удалении всех ссылок на экземпляр класса, в какой-то момент сборщик мусора закроет файл, но, когда это произойдет невозможно загадать, да и при желании использовать файл далее в программе не представится возможным, если он не закрыт при окончании использования экземпляром класса `FileWriter`. Деструктор будет вызываться даже в том случае, если скрипт был остановлен с помощью функции `exit()`. Вызов `exit()` в деструкторе предотвратит запуск всех последующих функций завершения.

### 2.3.5. Разрешение области видимости

Перед толкованием вопросов разрешения видимости введем понятие важного ключевого слова `self`. `Self` используется к обращению к внутреннему содержимому класса. Важно не путать и различать использование псевдопеременной `$this` для обращения к содержимому экземпляра класса. `Self` используется без знака `$` для обращения, но и внутри класса для обращения к статическим свойствам и методам.

Оператор разрешения области видимости, представляющий собой «двойное двоеточие», — это лексема, позволяющая обращаться к статическим свойствам, константам и переопределенным свойствам или методам класса. Далее рассматривается листинг 2.37.

Листинг 2.37. Использование статических переменных в Java

```
public class Woman {
    private String name;
    private static int count = 0;

    public Woman(String name) {
        this.name = name;
        count++;
    }

    public static int getCount() {
        return Woman.count;
    }
}
...
public class Main {

    public static void main(String[] args) {
        Woman alice = new Woman("Alice");
        System.out.println(alice.getCount()); // Woman.getCount()
    }
}
```

В листинге 2.37 представлена реализация класса `Woman` со статической переменной `count` для подсчета количества экземпляров класса. Внутри класса доступ к данной переменной возможен напрямую или через название класса в случаях предотвращения конфликта имен. В основной программе доступ возможен от экземпляра класса или от класса. Первый способ использовать не рекомендуется, но он возможен. Теперь в листинге 2.38 для сравнения приводится аналогичная реализация на языке программирования PHP:

### Листинг 2.38. Пример использования оператора разрешения видимости

```
<?php
class Woman{
    private string $name;
    private static int $count = 0;

    public function __construct($name){
        $this->name = $name;
        self::$count++;
    }

    public static function getCount(){
        return self::$count;
    }
}

$alice = new Woman("Alice");
echo Woman::getCount();
```

В примере в листинге 2.38 приведен пример реализации класса `Woman`, аналогичного классу в листинге 2.37. Важным отличием является доступ к статическим методам и свойствам класса. Внутри класса доступ осуществляется через ключевое слово `self` и оператора разрешения видимости. Аналогичная ситуация образуется при доступе к статическим элементам. Доступ осуществляется через имя класса и оператора разрешения области видимости. Доступ от экземпляра класса невозможен, что является важным и существенным отличием.

#### 2.3.6. Магические методы

Магическими называются методы, которые переопределяются действия производимые над объектами. Для переопределения доступен лишь ограниченный набор действия, например арифметические действия над объектами как в языке программирования Python, недоступны и невозможны над объектами в принципе. Всего существует 17 магических методов: `__construct()`, `__destruct()`, `__call()`, `__callStatic()`, `__get()`, `__set()`, `__isset()`, `__unset()`, `__sleep()`, `__wakeup()`, `__serialize()`, `__unserialize()`, `__toString()`, `__invoke()`, `__set_state()`, `__clone()` и `__debugInfo()`. Важным уточнением является то, что модификатор доступа для всех магических методов, кроме конструктора, деструктора и клонирования, должен быть публичным. Конструкторы и деструкторы были рассмотрены ранее и в данном разделе рассматриваться не будут. Важно упомянуть, что при использовании функциональности без определения магических методов будет вызываться фатальная ошибка, но при их определении будет выполняться их непосредственное выполнение без каких-либо предупреждений.



`__call()` и `__callStatic()` вызываются при вызове несуществующего метода класса, в первом случае простого метода, а во втором методы статического контекста. Предположим осуществляется вызов недоступной функции `count` с аргументами 6, 7, 8. Так как такой функции не существует для класса `Counter`, то вызывается метод `__call`. Пример представлен в листинге 2.39.

Листинг 2.39. Пример реализации метода `__call`

```
<?php
class Counter{
    public function __call($name, $args){
        echo "calling the method $name with arguments ";
        var_dump($args);
        return "undefined method";
    }
}

$result = (new Counter())->count(6, 7, 8);
```

В методе `__call` идет базовая обработка вызова: метод получает в параметрах название и массив аргументов. Метод `__callStatic` работает аналогично, но для статических вызовов.

Перегрузка в РНР означает возможность динамически создавать свойства и методы. Эти динамические сущности обрабатываются с помощью магических методов, которые можно создать в классе для различных видов действий. Этими методами являются: `__get()`, `__set()`, `__isset()`, `__unset()`. Метод `__get` вызывается при попытке чтения данных закрытых, или защищенных, или несуществующих. Метод `__set` аналогично вызывается в ситуации попытки записи в несуществующие или закрытие (или защищенные) свойства класса. Метод `__isset` выполняется при вызове методов `isset` и `empty` для закрытых (или защищенных) или несуществующих свойств класса. Метод `__unset` вызывается при вызове `unset` для несуществующих или закрытых (или защищенных) свойств. Параметрами данных методов являются имена свойств, над которыми производятся названные операции. Важным уточнением является то, что данные методы не работают в статическом контексте вызова.

Листинг 2.40. Пример переопределения методов

```
<?php
class DataStorage{
    private array $data = array();

    public function __set($name, $value){
        $this->data[$name] = $value;
    }

    public function __get($name){
```

```

        if(isset($this->data[$name])) return $this->data[$name];
        return null;
    }
}

$datastorage = new DataStorage();
$datastorage->lilo = "lila";
echo $datastorage->lilo;

```

В листинг 2.40 представлен пример определения методов `__set` и `__get`. В примере создается класс для хранения неких данных. Разработчику не важна внутреннее содержимое и внутренняя логика класса. Определение методов `__set` и `__get` дает возможность работать с определенными разработчиками свойствами.

Методы `__sleep()`, `__wakeup()` используются в процессах сериализации и десериализации объектов. Что такое сериализация и десериализация объектов и зачем она нужна? На этот вопрос имеется простой и неоригинальный ответ. Большая часть веб-приложения работает на основе протокола HTTP. Данный протокол работает без сохранения промежуточного состояния и поэтому разработчику нужно сохранять где-то это состояние сессии клиента и передача объектов сессии. Для это существуют процессы сериализации(превращения объекта в некую хранимую строку) и обратный процесс десериализации(превращение из строки обратно в объект). Для сериализации используется функция `serialize()`, которая проверяет, присутствует ли в классе метод с так называемым магическим именем `__sleep()`. Если это так, то этот метод выполняется до любой операции сериализации. Он может очистить объект и должен возвращать массив с именами всех переменных этого объекта, которые должны быть сериализованы. Логика использования `__sleep` заключается в сокрытии части свойств при сериализации, закрытии соединений и ресурсов для освобождения памяти. С другой стороны, функция `unserialize()` проверяет наличие метода с магическим именем `__wakeup()`. Если она имеется, эта функция может восстанавливать любые ресурсы, которые может иметь объект. Функциональность `__wakeup` предполагает восстановление всех свойств объекта, восстановление и возобновление всех требуемых соединений и ресурсов. Пример применения так называемых магических методов представлен в листинге 2.41.

Листинг 2.41. Пример использования магических методов `__sleep` и `__wakeup`

```

<?php
class FileWriter{
    ...
    public function __sleep(){
        return array('filepath');
    }
}

```

```

    }

    public function __wakeup() {
        $this->file=fopen($this->filepath, "a+");
    }
}

```

В примере представлено определение двух магических методов `__sleep` и `__wakeup`. Для сериализации данного класса нужен только путь к файлу, в который записываются данные, поэтому метод `__sleep` возвращает массив из одного элемента 'filepath'. После процесса десериализации нужно снова открыть файл для записи, что и происходит в методе `__wakeup`.

Методы `__serialize()` и `__unserialize()` используются соответственно для сериализации и десериализации объектов. `serialize()` проверяет, есть ли в классе функция с магическим именем `__serialize()`. Если да, функция выполняется перед любой сериализацией. Она должна создать и вернуть ассоциативный массив пар ключ/значение, которые представляют сериализованную форму объекта. Важно упомянуть, что если и `__serialize()` и `__sleep()` определены в одном и том же объекте, будет вызван только метод `__serialize()`. `__sleep()` будет игнорироваться. Магический метод `__serialize()` предполагается к использованию для формирования произвольного типа для сериализации объекта, который не обязательно может соответствовать свойствам класса. И наоборот, `unserialize()` проверяет наличие магической функции `__unserialize()`. Если функция присутствует, ей будет передан восстановленный массив, который был возвращён из `__serialize()`. Затем он может восстановить свойства объекта из этого массива соответствующим образом. Важно также упомянуть, что если и `__unserialize()` и `__wakeup()` определены в одном и том же объекте, будет вызван только метод `__unserialize()`. `__wakeup()` будет игнорироваться. Пример использования данных магических методов представлен в листинге 2.42.

Листинг 2.42. Магические методы `__serialize()` и `__unserialize()`

```

<?php
class FileWriter{
    private $file;
    private $filename;
    private $dir;
    ...
    public function __serialize(){
        fclose($this->file);
        return ["filepath" => $this->dir.$this->filename];
    }

    public function __deserialize(array $data){
        $arr = explode("/", $data["filepath"]);
    }
}

```

```

        $this->filename = $arr[count($arr)-1];
        $this->dir=implode("/", array_splice($arr, 0,
count($arr)-1));
        $this->file=fopen($this->dir.$this->filename, "a+");
    }
}

```

В примере приводится некоторая усовершенствованная реализация класса `FileWriter` с магическими методами для сериализации и десериализации. Для примера хранимый путь к файлу был разделен на 2 части: путь к папке, хранящейся файл и название самого файла. В методе для сериализации, во-первых, закрывается ресурс файла, что освобождает память и возвращается массив с полным значением пути к файлу. В методе десериализации из массива данных, который был возвращен методом `__serialize()`, производится восстановление объекта, а также открытие файла на запись.

Метод `__toString()` аналогичен переопределяемому методу `toString` в языке программирования Java для получения пользовательского варианта строкового представления объекта. Использование данного магического представлено в листинге 2.43.

Листинг 2.43. Создание класса с магическим методом `__toString`

```

<?php
class Cat{
    private string $name;

    public function __construct(string $name){
        $this->name=$name;
    }

    public function __toString(){
        return "/\._.\_/'\ named {$this->name}";
    }
}

$ball = new Cat("Ball");
echo $ball;

```

В пример приведено описание класса `Cat`, представляющего описание кота, со свойством имя. Для создания милого текстового представления экземпляра данного класса был определен магический метод `__toString`.

Когда любой язык программирования, особенно языки, реализующие объектно-ориентированную парадигму, запрещает исполнение и интерпретацию объекта как функцию, то в реализации объектно-ориентированной парадигмы в РНР данная реализация представлено магическим методом `__invoke`. Пример использования данного магического метода представлен в листинге 2.44.

Листинг 2.44. Пример использования магического метода `__invoke`

```
<?php
class Counter{
    private int $x = 0;
    public function __invoke(int $y){
        $this->x+=$y;
    }
}

$counter = new Counter;
$counter(5);
```

В примере описывается класс `Counter` с единственным свойством-счетчиком. Определение магического метода `__invoke` дает возможность вызывать экземпляр класса `Counter` как функцию, которая увеличивает счетчик на заданное количество.

Статический метод `__set_state` вызывается для тех классов, которые экспортируются функцией `var_export()`. `var_export()` возвращает структурированную информацию о данной переменной. Функция аналогична `var_dump()` за одним исключением: возвращаемое представление является полноценным PHP-кодом. Единственным параметром `__set_state` является массив, содержащий экспортируемые свойства в виде `['property' => value, ...]`. Пример использования данного магического метода представлен в листинге 2.45.

Листинг 2.45. Пример использования магического метода `__set_state`

```
<?php
class Counter{
    ...
    public static function __set_state($array){
        $c = new Counter;
        $c->x = $array["x"];
        return $c;
    }
}

$counter = new Counter;
$counter(5);
var_export($counter);
```

В примере определяется функция `__set_state` для класса `Counter` который экспортируется функцией `var_export`.

Метод `__debugInfo()` используется функцией `var_dump()` для получения списка свойств и их значений экземпляра класса. Этот метод нужен для сокрытия части свойств от данной функции, при не реализации будут выведены все свойства в независимости от идентификатора доступа. Пример использования данного магического метода представлен в листинге 2.46.

Листинг 2.46. Пример использования магического метода `__debugInfo()`

```
<?php
class Woman{
    public string $name;
    private string $birthday;

    public function __construct($name, $birthDay){
        $this->birthday = $birthDay;
        $this->name=$name;
    }
    ...
    public function __debugInfo(){
        return ["name"=>$this->name];
    }
}

$alice = new Woman("Alice", "1995-04-01");
var_dump($alice);
```

В примере представлена частичная дополненная реализация класса `Woman` с определением магического метода `__debugInfo()`. За счет этого идет сокрытие свойства дата рождения.

### 2.3.7. Наследование

Наследование является одним из признаков объектно-ориентированной парадигмы. Данный принцип связывает классами связью `parent-child`, которая широко используется при использовании ООП парадигмы. При расширении класса дочерний класс наследует все общедоступные и защищенные методы, свойства и константы родительского класса. До тех пор, пока эти методы не будут переопределены, они будут сохранять свою исходную функциональность. Закрытые методы родительского класса недоступны для дочернего класса. В результате дочерние классы могут повторно реализовать закрытый метод без учёта обычных правил наследования. Однако до версии PHP 8.0.0 к закрытым методам применялись ограничения `final` и `static`. Начиная с версии PHP 8.0.0, единственное ограничение закрытого метода, которое применяется – это конструкторы `private final`, поскольку это обычный способ "отключить" конструктор при использовании вместо него статичных фабричных методов. Пример наследования приведен в листинге 2.47.

Листинг 2.47. Пример наследования классов

```
<?php
class Animal{
    protected string $name;
    public function __construct(string $name){
        $this->name=$name;
    }
}
```

```

    }
    public function say(){
        echo "I am an Animal";
    }
}
class Cat extends Animal {

    public function __toString(){
        return "/\._.^\\ named {$this->name}";
    }

    public function say()
    {
        echo "May may";
    }
}

```

В примере определяется класс `Animal` со защищенным свойством `name`, конструктором и методом `say`. Класс `Cat`, который наследует класс `Animal` с помощью ключевого слова `extends`. Так как свойство родительского класса `name` защищенное, значит оно доступно и в дочернем классе.

Существует вопрос доступа к методам родительского класса, например в дочернем классе определяется конструктор, как и в родительском классе. Для того, чтобы воспользоваться функциональностью родительского класса, для этого в дочернем конструкторе возможен вызов родительского конструктора с помощью `parent::__construct()`. В данной сигнатуре `parent` обозначает родительский класс. Данную сигнатуру можно использовать к любому открытому или защищенному методу или константе родительского класса.

Важно упомянуть, что один класс может наследовать только один базовый класс.

### 2.3.8. Абстрактные классы

Абстрактные классы также являются неотъемлемой частью реализации ООП парадигмы. Поэтому в РНР возможно определение не только абстрактных классов, но и методов. Нельзя создать экземпляр абстрактного класса. Любой класс, содержащий хотя бы один абстрактный метод, должен быть определен как абстрактный. Абстрактный метод несет в себе только описательную функцию его наличия, реализация предполагается в дочерних классах. Пример абстрактного класса представлен в листинге 2.48.

Листинг 2.48. Пример создания абстрактного класса

```

<?php
abstract class Animal{
    protected string $name;
}

```

```

        public function __construct(strung $name){
            $this->name=$name;
        }
        public abstract function say();
    }
class Cat extends Animal {

    public function __toString(){
        return "/\._.'\ named {$this->name}";
    }

    public function say()
    {
        echo "May may";
    }
}

```

В примере приведена модификация классов из листинга 2.47. Так не может существовать по идее экземпляра класса животное и поэтому данный класс сделан абстрактным с абстрактным методом say, который реализуется в дочернем классе Cat.

### 2.3.9. Интерфейсы

Интерфейсы объектов — это важная функциональность, которая аналогична функциональности интерфейсов в языке программирования Java. Данная идея важна тем, что множественное наследование как в C++ в реализации парадигмы РНР невозможно, что с одной стороны решает проблему ромбовидного наследования, а с другой сильно ограничивает разработчика в его действиях. Интерфейсы расширяют функциональные возможности реализации парадигмы ООП за счет того, что:

- 1) возможна имплементация множественных интерфейсов;
- 2) возможно работать с объектами разных классов, имплементирующих один интерфейс, как с взаимозаменяемыми;
- 3) возможно оперировать параметром, который соответствует интерфейсу, не заботясь о том, что ещё может делать объект или как он реализован.

Интерфейсы могут определять магические методы, требуя от реализующих классов реализации этих методов. Но заявлять в интерфейсах реализацию конструктора не рекомендуется. В общем случае интерфейсы содержат методы без определения и константы классов. Имплементация интерфейса происходит с помощью ключевого слова implements. В следующем листинге 2.49 представлен пример реализации интерфейсов и его простого использования:



#### Листинг 2.49. Пример создания и использования интерфейсов

```
<?php
interface Sayable{
    public function say();
}

abstract class Animal implements Sayable {...}

class Cat extends Animal {
    ...
    public function say()
    {
        echo "May may";
    }
}

class Woman implements Sayable {
    ...
    public function say()
    {
        echo "I am a human!!";
    }
}

$arr = [new Woman("Alice", "1995-04-01"), new Cat("Ball")];
foreach ($arr as $value){
    $value.say();
}
```

В примере создается интерфейс `Sayable` с единственным методом `say`, который представляет собой вывод типового сообщения от объекта. Возвращаясь и модифицируя предыдущие примеры из листингов 2.46 и 2.48. получаем следующие модифицированные классы: `Animal` имплементирующий созданный интерфейс и не реализующий его по причине того, что абстрактные классы не обязательно должны реализовывать методы, объявленные в интерфейсах, `Cat`, который в свою очередь наследует абстрактный класс `Animal` и реализующий метод `say` интерфейса `Sayable`, а также класс `Woman`, который также имплементирует интерфейс `Sayable` и не является в данном случае потомком класса `Animal`. Далее для иллюстрирования 3 пункта списка возможностей интерфейсов создается массив из 2 элементов: экземпляра класса `Woman` и экземпляра класса `Cat`. При прохождении по массиву идет вызов метода `say`, здесь неважно к какому классу принадлежат экземпляры, главное, что классы реализуют интерфейс `Sayable`.

Уникальной функциональной возможностью является возможность наследования одного интерфейса от другого. Также возможно множественное наследование интерфейсов, что недоступно при наследовании классов. Пример

наследования интерфейсов представлен в листинге 2.50.

Листинг 2.50. Пример наследования интерфейсов

```
<?php
interface Able
{
    public function toBeAble();
}

interface Bable extends Able
{
    public function tobeBable(Bablez $baz);
}

class Test implements Bable
{
    public function toBeAble(){...}

    public function tobeBable(Bablez $baz){...}
}
```

В примере создается интерфейс `Able` с единственным методом `toBeAble()` и интерфейс `Bable`, который с помощью ключевого слова `extends` наследуется от интерфейса `Able`. с единственным методом `tobeBable()`. Класс `Test` имплементирует интерфейс `Bable` и это обязывает этот класс реализовывать метод `tobeBable` интерфейса `Bable`, а также метод `toBeAble` интерфейса `Able`.

### 2.3.10. Трейты

Идея повторного использования кода возникла уже при зарождении основ программирования. Повторное использование кода — это не только функции, циклы. В РНР существует отдельная функциональность для повторного использования кода под названием трейт. Трейт очень похож на класс, но предназначен для группирования функционала хорошо структурированным и последовательным образом. Невозможно создать самостоятельный экземпляр трейта. Это дополнение к обычному наследованию и позволяет сделать горизонтальную композицию поведения, то есть применение членов класса без необходимости наследования. Трейт содержит фрагмент класса. Зачем использовать трейты, если есть абстрактные классы, в которые можно вставить реализацию методов, реализация которых не меняется или частично меняется в дочерних классах, и есть интерфейсы, которые позволяют связать независимые классы. Но существует ситуация, когда есть множество классов, связанных интерфейсом. И реализация метода, объявленного интерфейсом во многих классах полностью идентична, что увеличивает размер и громоздкость кода. Именно в этой ситуации применяются трейты, содержащие отрывки классов с реализацией методов.

Пример создания трейта представлен в листинге 2.51.

Листинг 2.51. Пример создания трейта

```
<?php
trait Dbtrait{
    private $user;
    private $password;
    private $dbname;

    public function getUser($username){
        return "SELECT * FROM users WHERE name=$username";
    }
}
class UserStorage{
    use Dbtrait;
}
class AdminStorage{
    use Dbtrait;
}
```

В примере с помощью ключевого слова `trait`, определяется трейт для получения информации о пользователях из базы данных и других методов, связанных с доступом к данным. Для использования содержимого трейта нужно воспользоваться внутри класса ключевым словом `use` и далее название трейта.

При использовании трейтов важно понимать несколько важных аспектов. Во-первых, наследуемый член из базового класса переопределяется членом, находящимся в трейте. Порядок приоритета следующий: члены из текущего класса переопределяют методы в трейте, которые в свою очередь переопределяют унаследованные методы. Во-вторых, в класс можно добавить несколько трейтов, перечислив их в директиве `use` через запятую. Это аналогично имплементации нескольких интерфейсов классом. В-третьих, если два трейта добавляют метод с одним и тем же наименованием, это приводит к фатальной ошибке в случае, если конфликт не разрешён явно до этого. Для разрешения конфликтов именования между трейтами, используемыми в одном и том же классе, необходимо использовать оператор `insteadof` для того, чтобы точно выбрать один из конфликтующих методов. Пример разрешения конфликтов представлен в листинге 2.52.

Листинг 2.52. Множественное использования трейтов и разрешения конфликтов

```
<?php
trait Atalk{
    public function talk(){
        echo "A";
    }
}
trait Btalk{
```

```

    public function talk(){
        echo "B";
    }
    public function easytalk(){
        echo "Easy";
    }
}
class EasyTalker{
    use Atalk, Btalk{
        Atalk::talk insteadof Btalk;
        Btalk::easytalk as reallyeasytalk;
    }
}

```

В примере класс EasyTalker использует наполнение двух трейтов: Atalk и Btalk. Но при использовании данных трейтов получается конфликт для функции talk, которая определена в каждом трейте. Для разрешения конфликта четко указывается какой метод talk из двух возможных использовать с помощью ключевого слова `insteadof`. Также возможно переименование метода при его использовании с помощью ключевого слова `as`. А если после `as` поставить измененный модификатор доступа, то можно его изменить для соответствующего метода, если это требуется.

Продолжая список важных аспектов о трейтах. В-четвертых, трейты могут использоваться как в классах, так и в других трейтах. Используя один или более трейтов в определении другого трейта, он может частично или полностью состоять из членов, определенных в этих трейтах. Это применяется для случаев необходимости группировки функциональности. В-пятых, трейты поддерживают использование абстрактных методов для того, чтобы установить требования к используемому классу. До РНР версии 8.0.0 поддерживались открытые и защищенные абстрактные методы, после поддерживаются все доступные модификаторы доступа. В-шестых, трейты поддерживают все возможные статическое наполнение класса: переменные, методы и свойства. В-седьмых, как уже упоминалось ранее, трейты могут объявлять в себе свойства и даже инициализировать их.

### **2.3.11. Анонимные классы**

Начиная с РНР версии 7, началась поддержка анонимных классов. Функциональность анонимных классов РНР практически аналогична анонимным классам в языке программирования Java. Анонимные классы нужны в случае, когда нужна некоторая функциональность в одном конкретном месте. Пример создания и использования анонимных классов представлен в листинге 2.53.

Листинг 2.53. Пример создания и использования анонимного класса

```
<?php
class Storage{
    private string $title = "Class Storage";
    protected int $id_storage = 1;
    public function getAnonClass(){
        return new class($this->title) extends Storage{
            private string $name;
            public function __construct($title){
                $this->name=$title;
            }
            public function myecho(){
                echo "{$this->name} ({ $this->id_storage})";
            }
        };
    }
}
(new Storage())->getAnonClass()->myecho();
```

В примере представлено не совсем тривиальное использование анонимного класса, чаще всего анонимный класс подается какой-либо функции или какому-либо методу как параметр, а в случае примера анонимный класс возвращается из метода другого класса. При этом анонимный класс является его потомком и здесь работают все правила наследования: закрытое свойство `title` недоступно анонимному классу и его приходится передавать в конструктор при его инициализации и объявлении одновременно, а вот доступ к защищенному методу есть.

Анонимные классы используются для передачи аргументов в конструкторы, расширения других классов, реализации интерфейсов и использования трейтов как обычных классов. Также следует обратить внимание на то, что анонимным классам присваиваются имена движком PHP. Это имя следует рассматривать как особенность реализации, на которую не следует полагаться.

### 2.3.12. *Nullsafe и ООП в PHP*

Нововведением PHP версии 8.0.0 является оператор `nullsafe`. Оператор `nullsafe` очень важен и сокращает огромные куски кода с множественными проверками на `null`. Для понимания оператора и его работы с ООП в PHP приводится пример в листинге 2.54.

Листинг 2.54. Пример использования оператора `nullsafe`

```
<?php
$storage = new UserStorage();
$result = $storage?->getUser(5)?->name;

if (is_null($storage)) {
```

```

        $result = null;
    } else {
        $user = $storage->getUser(5);
        if (is_null($user)) {
            $result = null;
        } else {
            $result = $user->name;
        }
    }
}

```

В примере использование оператора `nullsafe` уберегает разработчика от использования сложного вложенного условного оператора, что в разы сокращает код и простоту его понимания и оберегает от выброса исключений. Оператор `nullsafe` работает так же, как доступ к свойству или методу, за исключением того, что если разыменованное выражение выдает `null`, то будет возвращен `null`, а не выброшено исключение. Если разыменованное выражение является частью цепочки, остальная часть цепочки пропускается. Данный оператор следует использовать только в том случае, если `null` является ожидаемым ответом, иначе нужно выбрасывать исключение.

### 2.3.13. Автоматическая загрузка классов

Существует вопрос использования написанных разработчиком классов. Как и для языка программирования Java существует правило хранения один файл – один класс. Но для использования множества классов в одном скрипте требуется прописать использование всех классов вручную, что неудобно и громоздко в плане кода. В PHP существует решение данной проблемы в виде автоматической загрузки классов. Существует 2 основные функции для использование данной функциональности: `spl_autoload_register()` и `__autoload()`. Но последняя признана устаревшей в релизе PHP 7.2.0 и удалена в версии 8.0. Первая функция `spl_autoload_register()` позволяет зарегистрировать необходимое количество автозагрузчиков для автоматической загрузки классов и интерфейсов, если они в настоящее время не определены. Регистрируя автозагрузчики, PHP получает последний шанс для интерпретатора загрузить класс прежде, чем он закончит выполнение скрипта с ошибкой. Пример автоматической загрузки классов представлен в листинге 2.55.

Листинг 2.55. Пример использования автоматической загрузки классов

```

<?php
spl_autoload_register(function ($class_name) {
    require_once $class_name . '.php';
});

$alice = new Woman();

```

Что произойдет при выполнении кода из листинга 2.55? Будет произведен анализ какие классы используются в данном скрипте и потом на основе данного списка будет вызываться функция, переданная функции `spl_autoload_register()`, для автоматического подключения нужного класса. В нашем случае будет выполнена строка `require_once 'Woman.php'`. Поэтому важно использовать идею из языка программирования Java: один класс — один файл и название файла полностью соответствует имени класса.

#### **2.3.14. Ключевое слово *final***

Запрет переопределения в наследовании или запрет самого наследования является важной частью реализации парадигмы ООП. Для реализации данной функциональности применяется ключевое слово `final`, для применения которого нужно его разместить первым перед определением метода или класса. Важно понимать, что запрет переопределения или наследования доступен только для класса или метода, но не для константы или свойства класса. Пример недоступного для наследования класса представлен в листинге 2.56.

Листинг 2.56. Пример `final`-класса

```
<?php
final class Unextendable{
    private string $name;

    public function getName(){
        return $this->name;
    }
}
```

В примере представлен класс недоступный для наследования класс `Unextendable`. Попытка создать класс на основе данного выдает фатальную ошибку. Применение ключевого слова `final` для методов данного класса избыточно и не является необходимым.

#### **2.3.15. Позднее статическое связывание**

Позднее статическое связывание — это получение ссылки на вызываемый класс в контексте статического наследования. Более точно, это сохранение имени класса указанного в последнем "не перенаправленном вызове". В случае статических вызовов это явно указанный класс (обычно слева от оператора `::`); в случае не статических вызовов это класс объекта. Перенаправленный вызов - это статический вызов, начинающийся с `self::`, `parent::`, `static::`, или, если двигаться вверх по иерархии классов, `forward_static_call()`. Функция `get_called_class()` может быть использована для получения строки с именем вы-

званного класса, а `static::` представляет её область действия.

"Позднее связывание" отражает тот факт, что обращения через `static::` не будут вычисляться по отношению к классу, в котором вызываемый метод определён, а будут вычисляться на основе информации в ходе исполнения. Также эта особенность была названа "статическое связывание" потому, что она может быть использована (но не обязательно) в статических методах.

Важным является ограничение `self` и `__CLASS__`. Рассмотрим пример, представленный в листинге 2.57.

Листинг 2.57. Пример ограничения `self` и `__CLASS__`

```
<?php
class A{
    public static function who() {
        echo __CLASS__;
    }
    public static function test() {
        self::who();
    }
}
class B extends A{
    public static function who() {
        echo __CLASS__;
    }
}
B::test();
```

Результатом выполнения данного будет "А", так как вызов данного метода будет в контексте соответствующего класса. По идее разработки хочется видеть в результате "В".

Для иллюстрирования использования позднего статического связывания немного модифицируем пример из листинга 2.57. Модифицированный вариант приведен в листинге 2.58.

Листинг 2.58. Пример использования позднего статического связывания

```
<?php
class A {
    public static function who() {
        echo __CLASS__;
    }
    public static function test() {
        static::who();    }
}

class B extends A {
    public static function who() {
        echo __CLASS__;
    }
}

B::test();
```



В примере ключевое `self` в методе `test` было изменено на ключевое слово `static`, которое указывает на класс, от которого был вызван метод и теперь разработчик получает желаемый результат.

### 2.3.16. Ковариантность и контравариантность

В PHP 7.4.0. добавлена полная поддержка ковариантности и контравариантности. Что значат два этих важных понятия будет разобрано далее.

Ковариантность позволяет дочернему методу возвращать более конкретный тип, чем тип возвращаемого значения его родительского метода. Для иллюстрации данного понятия обратимся к примеру в листинге 2.59.

Листинг 2.59. Иллюстрация ковариантности

```
<?php
abstract class Animal{
    protected string $name;
    public function __construct(string $name){
        $this->name=$name;
    }
    public abstract function say();
}
class Cat extends Animal {

    public function say()
    {
        echo "May may";
    }
}

class Dog extends Animal{

    public function say()
    {
        echo "Woof woof";
    }
}

interface AnimalShelter
{
    public function adopt(string $name): Animal;
}

class CatShelter implements AnimalShelter
{
    public function adopt(string $name): Cat    {
        return new Cat($name);
    }
}

class DogShelter implements AnimalShelter
```

```

{
    public function adopt(string $name): Dog    {
        return new Dog($name);
    }
}

```

В примере идет модификация уже разобранный примера, в точности добавление еще одного класса потомка `Animal` – `Dog`. Далее создаются конкретные фабрики для создания конкретных потомков класса `Animal`. Ковариантность заключается в том, что методы фабрик возвращают конкретные типы объектов потомков класса `Animal`.

В то время как контравариантность позволяет типу параметра в дочернем методе быть менее специфичным, чем в родительском. Для иллюстрирования понятия контравариантность модифицируем пример из листинга 2.59 в листинге 2.60.

Листинг 2.60. Пример контравариантности

```

<?php

class Food {}

class AnimalFood extends Food {}

abstract class Animal{
    ...
    public function eat(AnimalFood $food)
    {
        echo $this->name . " ест " . get_class($food);
    }
}

class Dog extends Animal{
    ...
    public function eat(Food $food) {
        echo $this->name . " ест " . get_class($food);
    }
}

```

В примере идет добавление двух классов `Food` и его потомка `AnimalFood`. А также нового метода `eat`, который в классе `Animal` принимает в качестве параметра объект типа `AnimalFood`, а в переопределении в классе `Dog` объекта класса `Food`. Метод в классе `Cat` соответствует методу в классе `Animal`. Уже переопределение в классе `Dog` метода с параметром `Food` родителем `AnimalFood`.

### 2.3.17. Правила совместимости сигнатуры

Важными правилами являются правила совместимости сигнатур при переопределении метода родительского класса при наследовании. В PHP 8.0 выда-

ется фатальная ошибка при несовместимости, а до версии 8 ошибка уровня E\_WARNING. Сигнатура является совместимой, если она соответствует правилам контравариантности, делает обязательный параметр необязательным и если какие-либо новые параметры являются необязательными. Это известно как принцип подстановки Барбары Лисков или сокращённо LSP. Правила совместимости не распространяются на конструктор и сигнатуру private методов, они не будут выдавать фатальную ошибку в случае несоответствия сигнатуры. Пример совместимости дочерних методов представлен в листинге 2.61.

Листинг 2.61. Пример совместимости сигнатуры

```
<?php
class Base{
    public function baz(int $a) {
        echo "Допустимо\n";
    }
}

class Extendclass1 extends Base{
    function baz(int $a = 5)
    {
        parent::baz($a);
    }
}

class Extendclass2 extends Base{
    function baz(int $a, $b = 5)
    {
        parent::baz($a);
    }
}
```

В примере объявляется базовый класс Base с единственным публичным методом baz. Класс Extendclass1 является потомком класса Base и переопределяет метод baz, делая для параметра \$a значение по умолчанию 5. Класс Extendclass2 является потомком класса Base и переопределяет метод baz, делая для нового параметра \$b значение по умолчанию 5. Правила совместимости сигнатур соблюдены.

## 3. ОСНОВЫ ВЕБ-СЕРВЕРОВ

### 3.1. Ретроспектива развития веб-серверов

#### 3.1.1. История развития интернета

По сути, термин «веб-сервер» – некое подобие черного ящика. Веб-сервер в процессе своей работы обрабатывает запросы браузера и выдает в ответ веб-страницы. Работа веб-сервера сводится к приему HTTP-запросы от веб-браузеров, выступающих в качестве и выдачи им в ответ соответствующих ответов (как правило это HTTP-ответы, включая HTML-страницы) а также файлы изображений, медиа-поток, файлы других типов и различные данные. Важно заметить, что, по сути, веб-сервер — это информационная система, так как его функционирование невозможно как без специального программного обеспечения, так и соответствующей аппаратной части, т.е. компьютера. Не смотря на бурное развитие веб-разработки, многие из используемых в настоящее время решений, используемых в данной области, имеют в основе технологии и принципы, которые остаются неизменными. Такие принципы, и будут рассмотрены в данном разделе пособия.

Для того, чтобы лучше понимать базовые технологии и пути их развития, следует совершить экскурс в историю становления и совершенствования компьютерных технологий.

Традиционно принято считать, что современный интернет начал развиваться в конце 80-ых и начале 90-ых годов. В то время интернет-сайты были просто хранилищем документов, которые имели специальную разметку. В частности, использовался язык разметки SGML. SGML (Standard Generalized Markup Language) — это стандартный обобщенный язык разметки. Помимо размеченных документов сайты сохраняли некоторых связанных с документом данные, например, файлы, изображения и т.д. Для создания ссылок между документами и ссылок между документом и связанными файлами, был предложен специальный язык гипертекстовой разметки – HTML (HyperText Markup Language). Для доступа к таким документам посредством сети – протокол HTTP (HyperText Transfer Protocol). Не смотря на развитие интернета и модификацию, многие принципы, заложенные в эти языки существенно, не изменились до настоящего времени. Только в 1999 году при смене протокола HTTP/1.1 на более совершенный HTTP/2 произошли некоторые кардинальные изменения, которые были необходимы для развития сети. Протокол HTTP реализован по клиент-серверной технологии и работает по принципу запрос-ответ без сохранения состояния. Целью запроса служит некий ресурс, который опре-

деляется единым идентификатором ресурса – URI (Uniform Resource Identifier), HTTP использует одну из разновидностей URI – URL (Uniform Resource Locator) – универсальный указатель ресурса, который помимо сведений о ресурсе определяет также его физическое местоположение.

В процессе работы HTTP-сервера получает запрос клиента, обрабатывает его и либо выдает ему запрашиваемый ресурс, либо сообщает, что это сделать невозможно. Работа HTTP-сервер показана на рис. 3.1. Рассмотрим работу сервера более подробно.



*Рисунок 3.1. Схема взаимодействия клиента и HTTP-сервера*

Пользователь посредством HTTP-клиента (чаще всего это браузер) запрашивает у HTTP-сервера некий URL, сервер проверяет и отдает соответствующий этому URL-файл, обычно это HTML-страница. Запрашиваемый документ (в нашем примере это HTML-страница) может, в свою очередь, ссылаться на связанные ресурсы, например, изображения, файлы данных, документы. Для отображения таких связанных ресурсов на странице, клиентом (браузером) последовательно запрашиваются у сервера все связанные документы. Помимо изображений клиент может запрашивать таблицы стилей, скрипты, исполняемые на стороне клиента и т.д. После получения всех запрашиваемых ресурсов клиент (браузер) обработает их в соответствии с кодом HTML-документа и выдаст пользователю готовую страницу.

На рис. 3.1 обозначен HTTP-сервер, однако в настоящее время он может также обозначаться как веб-сервер. Основные цели и задачи такого веб-сервера сводятся к обработке HTTP-запросов клиента и возвращении пользователю результатов этой обработки. Заметим, что Веб-сервер не способен к самостоятельной генерации контента, он может обрабатывать только статическое содержимое. Несмотря на большие возможности современных серверов, они по-прежнему не способны к самостоятельной генерации контента.

В течение продолжительного времени веб-сервер оставался самодостаточным инструментом реализации полноценного сайта. По мере роста сети Интернет, возможности статического HTML перестали удовлетворять интернет-

сообщество. В качестве примера можно привести ситуацию, когда добавляются новые страницы. Как известно, каждая статическая интернет-страница уже является самодостаточной и содержит ссылки на все связанные с ней ресурсы. В случае же добавления новых страниц необходимо будет добавить ссылки и на них в уже существующие страницы, в противном случае никто не сможет попасть на вновь созданные страницы. Заметим, что это надо будет сделать вручную. Задача выглядит трудоемкой, однако при большом числе добавляемых страниц она становится невыполнимой.

Таким образом даже сайты, созданные в 90-ых годах были ощутимо менее удобными чем современные. Так в качестве примера можно привести сайт компании Rambler образца 1997 года (см. рис. 3.2).

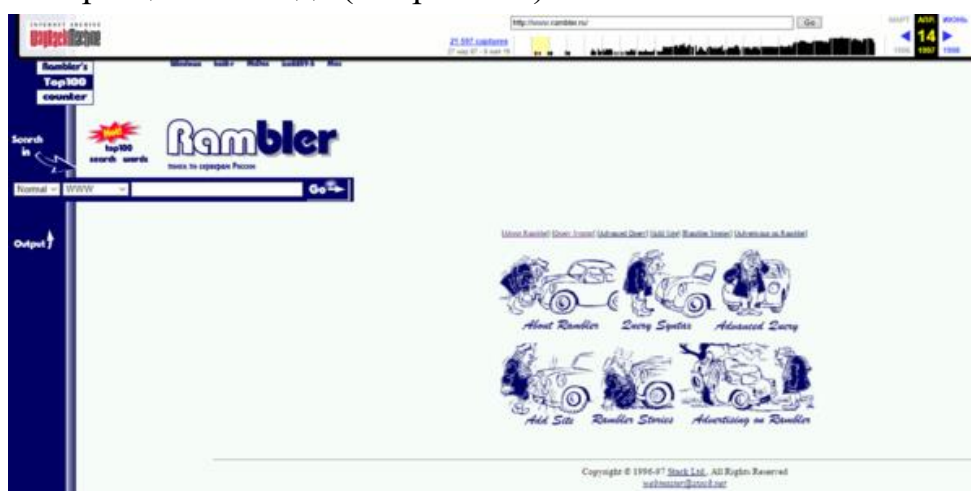


Рисунок 3.2. Сайт компании Rambler 1997 года

На представленной странице возможен только односторонний переход. Иначе говоря, после того, как пользователь перешел по ссылке на странице вернуться обратно он может только используя специальную кнопку-стрелку в браузере. Стандартные на данный момент средства перехода тогда еще не существовали.

Создание сайтов похожих на современные по внешнему виду и удобству использования представляло собой весьма сложную работу, в процессе которой задачи нарастали как снежный ком. Причем необходимо было вносить соответствующие изменения во все существующие страницы, связанные с вновь создаваемой. Даже изменение логотипа в шапке, например, на стартовой странице приводило к необходимости изменения соответствующих данных во всех существующих к тому моменту страницах. При изменении пути к одной из страниц сайта или её удаления, необходимо было отыскать все существующие ссылки на эту страницу, а потом изменить их или удалить.

### 3.1.2. Язык динамической сборки веб-страниц SSI

Такая трудоемкость создания веб-страниц не могла устраивать разработчиков, поэтому был сделан следующий шаг в развитии веб-серверов – был создан язык SSI. SSI (от английского Server Side Includes – включение на стороне сервера) – это, как уже было сказано ранее, язык, разработанный для динамического создания и «сборки» веб-страниц на сервере из отдельных составных частей и выдачи клиенту полученного HTML-документа. Использование языка SSI дает возможность в код страницы инкапсулировать содержимое различных файлов. В этом случае, внесение дополнительных повторяющихся элементов, такие, например, как шапка (header), подвал (footer), меню и другие осуществляется в специальные обособленные файлы. Затем созданные таким образом файлы просто подключаются при окончательной сборке страницы. Описанная схема показана на рис. 3.3.

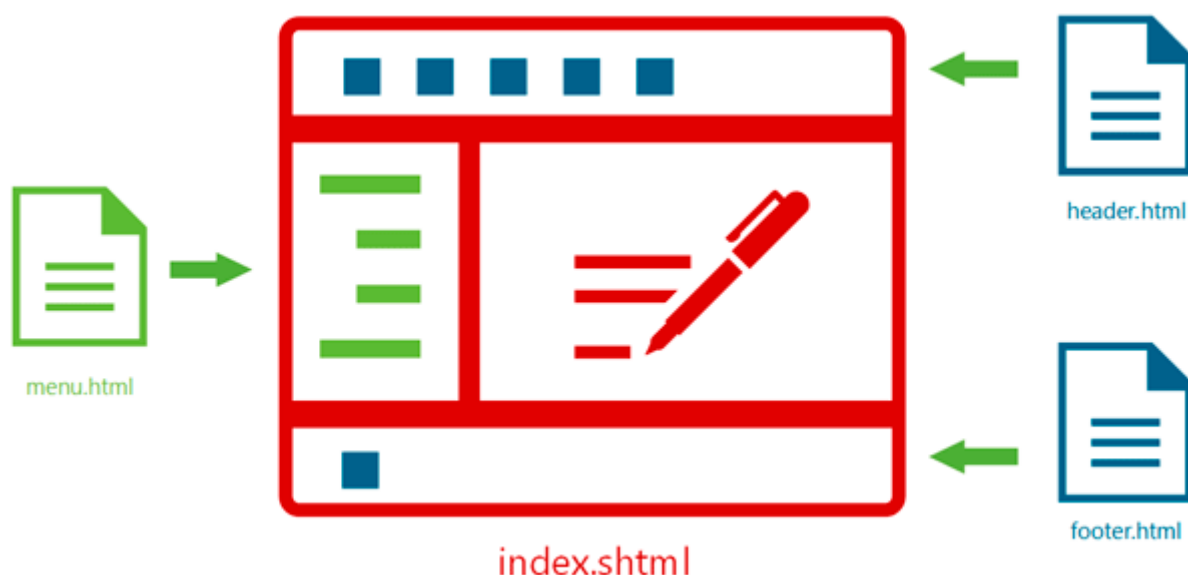


Рисунок 3.3. Схема технологии SSI

При таком подходе для изменения логотипа или пункта меню изменения надо было вносить только в один файл, что, согласитесь, значительно проще, а главное гораздо менее трудоемко, чем правка всех существующих страниц. Учитывая описанный прогресс можно называть SSI не только языком, но и технологией. Помимо описанных достоинств, технология SSI дает возможность добавления на веб-страницы дополнительное динамическое содержимое, такое, например, как актуальная дата на странице. Технология дополнительно дает возможность выполнять несложные условия и работать с переменными. Создание SSI явилось важной вехой в развитии интернет-технологий. Использование

SSI позволило значительно облегчить работу вебмастеров и повысило удобство использования сайтов. Однако реализовать по-настоящему динамический сайт данная технология все еще не позволяет.

Несмотря на ограничения SSI активно применяют и в настоящее время, в первую очередь из-за простоты и низкой ресурсоемкости создаваемых страниц. SSI, как правило, применяют там, где предполагается вставка статического контента в код.

### **3.1.3. CGI – Интерфейс общего шлюза**

CGI (от английского Common Gateway Interface) или интерфейс общего шлюза, представляет собой стандарт интерфейса, используемого внешней программой для связи с веб-сервером.

Бурное развитие интернет-технологий и связанные с ним запросы потребителей требовали новых решений. Обработка запросов пользователей на стороне сервера стала возможно с появлением специального класса программ – СКРИПТОВ. Такие программы, как правило, создаются с использованием специальных скриптовых языков, первоначально в качестве такого языка использовался Perl, поскольку он создавался лингвистом Ларри Уолли и обладает богатыми возможностями для работы с текстом. Затем был разработан PHP, на который Perl, кстати, оказал существенное влияние. Дальнейшее увеличение объемов работ и требований к оперативности их выполнения привели к созданию особого класса программ, можно даже сказать – систем. Такие системы в настоящее время прочно заняли свое место в арсенале веб-мастера и называются CMS системы. CMS (от английского Content management system) – системы управления контентом, в виде полноценного веб-приложения. CMS предназначены для обеспечения динамической обработки запросов пользователя.

Описанные ранее языки и технологии работают с классическими веб-серверами, которые не выполняют скрипты. Задача таких серверов состоит в том, чтобы отдавать статический материал, накапливаемый в базах данных (БД) и обрабатываемый системой управления базой данных (БД). Такая организация продиктована необходимостью доступа к большому количеству взаимосвязанной информации.

Далее мы переходим к рассмотрению нового типа сущности, а именно – СЕРВЕРА ПРИЛОЖЕНИЙ. Сервер приложений является интерпретатором скриптовых языков, с помощью которых, работают написанные на них веб-приложения.

Следует особо подчеркнуть, что сервер приложений не работает с прото-



колом HTTP и не обрабатывает пользовательские запросы. Все эти действия по-прежнему выполняет веб-сервер. Для корректного взаимодействия был создан так называемый «общий интерфейс шлюза» – CGI (от английского Common Gateway Interface), который схематически представлен на рис. 3.4.

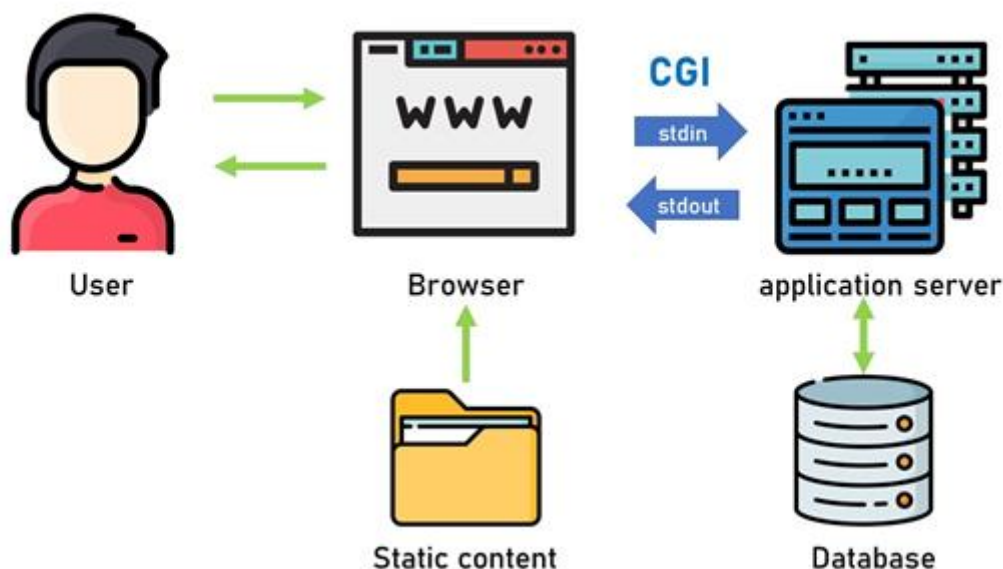


Рисунок 3.4. Общий интерфейс шлюза – CGI

Особо отметим, что CGI не является ни программой, ни протоколом, это именно интерфейс. Под интерфейсом мы понимаем некую определённую совокупность способов взаимодействия между приложениями. Существует также другой термин CGI-приложение / CGI-скрипт. Этим термином называют специальную программу (скрипт), которая поддерживает работу через интерфейс CGI. Эти термины ни в коме случае нельзя путать.

Для передачи данных используются стандартные потоки ввода-вывода, от веб-сервера к CGI-приложению данные передаются через поток STDIN, принимаются обратно через поток STDOUT, а для передачи сообщений об ошибках используется поток STDERR.

Рассмотрим процесс работы такой системы подробнее. Получив запрос от браузера пользователя, веб-сервер определяет, что запрошено динамическое содержимое и формирует специальный запрос, который через интерфейс CGI передается веб-приложению. При его получении приложение запускается и выполняет запрос, результатом которого служит HTML-код динамически сформированной страницы, который передается назад веб-серверу, после чего приложение завершает свою работу.

Еще одно важное отличие динамического сайта – его страницы физически не существуют в том виде, который отдается пользователю. Фактически имеет-

ся веб-приложение, т.е. набор скриптов, шаблонов и база данных, которая хранит материалы сайта и служебную информацию. Отдельно располагается статическое содержимое: картинки, java-скрипты, файлы.

Получив запрос веб-приложение извлекает данные из БД и заполняет ими указанный в запросе шаблон. Результат отдается веб-серверу, который дополняет сформированную таким образом страницу статическим содержимым (изображения, скрипты, стили) и отдает ее браузеру пользователя. Сама страница при этом нигде не сохраняется, разве что в кэше браузера пользователя, и при получении нового запроса произойдет повторная генерация страницы.

К достоинствам CGI можно отнести языковую и архитектурную независимость: CGI-приложение может быть написано на любом языке и одинаково хорошо работать с любым веб-сервером. Учитывая простоту и открытость стандарта, это привело к активному развитию веб-приложений.

Однако, кроме достоинств, CGI обладает и существенными недостатками. Основной из них – высокие накладные расходы на запуск и остановку процесса, что влечет за собой повышенные требования к аппаратным ресурсам и невысокую производительность. Использование стандартных потоков ввода-вывода ограничивает возможности масштабирования и обеспечения высокой доступности, так как требует, чтобы веб-сервер и сервер приложений находились в пределах одной операционной системы (ОС).

В настоящее время интерфейс CGI вытеснен более современными технологиями и практически не встречается на просторах интернета.

#### **3.1.4. FastCGI**

Основной целью разработки данной технологии было повышение производительности CGI. Являясь ее дальнейшим развитием, FastCGI представляет собой клиент-серверный протокол для взаимодействия веб-сервера и сервера приложений, обеспечивающий высокую производительность и безопасность.

FastCGI устраняет основную проблему CGI – повторный запуск процесса веб-приложения на каждый запрос, FastCGI процессы запущены постоянно, что позволяет существенно экономить время и ресурсы. Для передачи данных вместо стандартных потоков используются UNIX-сокеты или TCP/IP, что позволяет размещать веб-сервер и сервера приложений на разных хостах, таким образом обеспечивая масштабирование и/или высокую доступность системы. На рис. 3.5 представлена схема взаимодействия клиента и серверов.

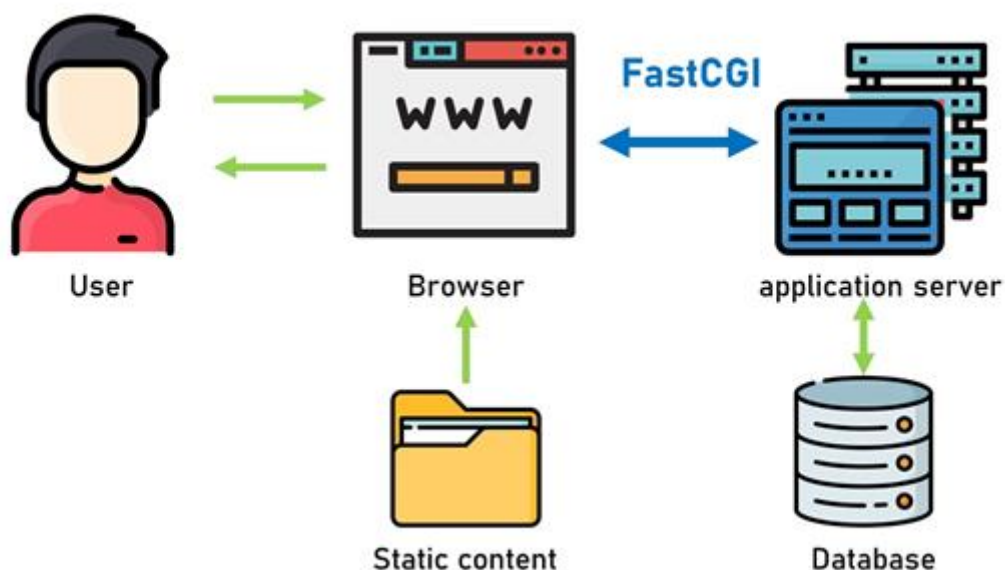


Рисунок 3.5. Схема взаимодействия клиента, веб-сервер и сервера приложений

Также мы можем запустить на одном компьютере несколько FastCGI-процессов, которые могут обрабатывать запросы параллельно, либо иметь различные настройки или версии скриптового языка. Например, можно одновременно иметь несколько версий PHP для разных сайтов, направляя их запросы разным FastCGI-процессам.

Для управления FastCGI-процессами и распределением нагрузки служат менеджеры процессов, они могут быть как частью веб-сервера, так и отдельными приложениями. Популярные веб-сервера Apache и Lighttpd имеют встроенные менеджеры FastCGI-процессов, в то время как Nginx требует для своей работы с FastCGI внешний менеджер.

### 3.1.5. PHP-FPM и spawn-fcgi

Из внешних менеджеров для FastCGI-процессов применяются PHP-FPM и spawn-fcgi. PHP-FPM первоначально был набором патчей к PHP от Андрея Нигматулина, решавший ряд вопросов управления FastCGI-процессами, начиная с версии 5.3 является частью проекта и входит в поставку PHP. PHP-FPM умеет динамически управлять количеством процессов PHP в зависимости от нагрузки, перезагружать пулы без потери запросов, аварийно перезапускать сбойные процессы и представляет собой менеджер с расширенным функционалом.

Spawn-fcgi является частью проекта, но в состав одноименного веб-сервера не входит, по умолчанию Lighttpd использует собственный, более простой, ме-

неджер процессов. Разработчики рекомендуют использовать его в случаях, когда вам нужно управлять FastCGI-процессами, расположенными на другом хосте, либо требуются расширенные настройки безопасности.

Внешние менеджеры позволяют изолировать каждый FastCGI-процесс в своем chroot окружении, отличном как от chroot окружений других процессов, так и от chroot окружения веб-сервера. (Chroot-окружение – это системный вызов, который временно перемещает root каталог в новую папку. Как правило, root каталог находится в «/». Но при помощи chroot можно задать другой каталог, который будет служить как root-каталог в окружении chroot. Любые приложения, которые запускаются внутри изолированного окружения, в принципе не могут взаимодействовать с остальной операционной системой. Кроме того, не-рутовый пользователь (non-root), помещённый в chroot-окружение, не сможет перемещаться по иерархии каталогов). Оба этих менеджера позволяют работать с FastCGI-приложениями, расположенными на других серверах через TCP/IP. В случае локального доступа следует выбирать доступ через UNIX-сокеты, как наиболее быстрый тип соединения. Схематично это представлено на рис. 3.6.

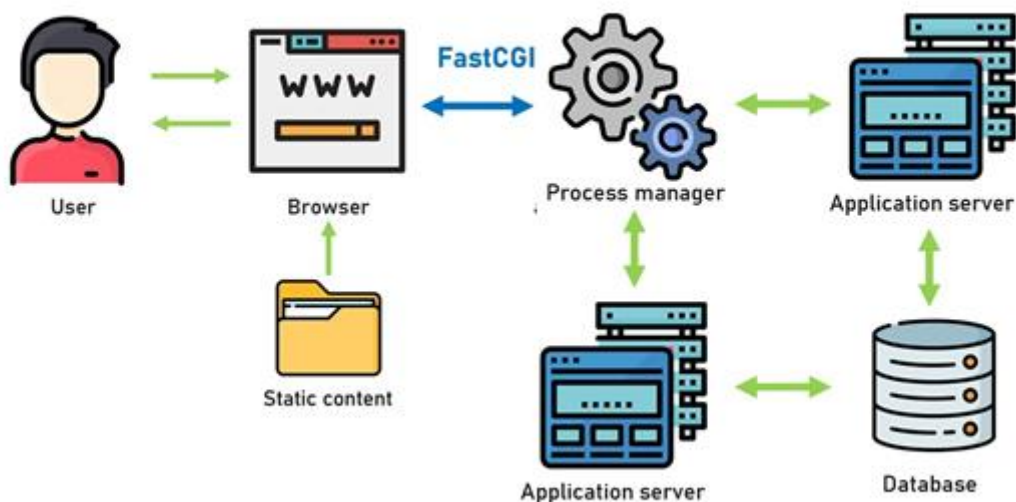


Рисунок 3.6. Схема взаимодействия серверов с использованием FastCGI

На схеме видно, что у нас появился новый элемент – менеджер процессов, который является посредником между веб-сервером и серверами приложений. Это несколько усложняет схему, так как настраивать и сопровождать приходится большее количество служб, но в то же время открывает более широкие возможности, позволяя настроить каждый элемент сервера исключительно для решения своих задач.

На практике, выбирая между встроенным менеджером и внешним, надо оценить ситуацию и выбирать именно тот инструмент, который наиболее подходит запросам. Например, создавая простой сервер для нескольких сайтов на типовых движках применение внешнего менеджера будет явно излишним. Данный подход хорош именно тем, что можно, как из конструктора, собрать именно то, что нужно для решения конкретной задачи.

### **3.1.6. SCGI, PCGI, PSGI, WSGI и прочие.**

Углубляясь в тему веб-разработки, мы непременно будем встречать упоминания о различных CGI-технологиях, наиболее популярные из которых SCGI, PCGI, PSGI, WSGI. Но имея понимание как работает CGI и FastCGI, разобраться с любой из этих технологий достаточно просто.

Несмотря на различия в реализациях того или иного решения базовые принципы остаются общими. Все эти технологии предоставляют интерфейс шлюза (Gateway Interface) для взаимодействия веб-сервера с сервером приложений. Шлюзы позволяют развязать между собой среды веб-сервера и веб-приложения, позволяя использовать любые сочетания не обращая внимание на возможную несовместимость. Неважно, поддерживает ли конкретный веб-сервер конкретную технологию или скриптовый язык, главное, чтобы он умел работать с нужным типом шлюза.

Рассмотрим каждую из перечисленных технологий более подробно.

SCGI (Simple Common Gateway Interface) – простой общий интерфейс шлюза – разработан как альтернатива CGI и во многом аналогичен FastCGI, но более прост в реализации. Все, что применимо к FastCGI, справедливо и для SCGI.

PCGI (Perl Common Gateway Interface) - библиотека Perl для работы с интерфейсом CGI, долгое время являлась основным вариантом работы с Perl-приложениями через CGI, отличается хорошей производительностью при скромных потребностях в ресурсах и неплохой защиты от перегрузки.

PSGI (Perl Web Server Gateway Interface) – технология взаимодействия веб-сервера и сервера приложений для Perl. Если PCGI представляет собой инструмент для работы с классическим CGI-интерфейсом, то PSGI более напоминает FastCGI. PSGI-сервер представляет среду для выполнения Perl-приложений которая постоянно запущена в качестве службы и может взаимодействовать с веб-сервером через TCP/IP или UNIX-сокеты и предоставляет Perl-приложениям те же преимущества, что и FastCGI.

WSGI (Web Server Gateway Interface) – предназначен для взаимодействия

веб-сервера с сервером приложений для программ, написанных на языке Python.

Все перечисленные технологии являются в той или иной степени аналогами CGI/FastCGI, но для специфических областей применения.

### **3.2. Архитектура сервера Apache**

HTTP-сервер Apache, является свободным, кроссплатформенным программным обеспечением, которое отлично работает со следующими семействами ОС:

- 1) Windows;
- 2) Mac OS;
- 3) Unix.

В архитектуру Apache входит: простое ядро, платформо-зависимый уровень (APR), и модули. Любое приложение для Apache - даже простейшее, обслуживающее "дефолтную" страницу Apache "It worked" - использует несколько модулей. Пользователи Apache не нуждаются в знании этого, но для разработчика программ, понимание модулей и API модуля Apache является необходимым знанием при работе с Apache. Большинство, но не все модули, связаны с различными аспектами обработки HTTP запроса. Преимущество модульного подхода состоит в том, что он позволяет фокусировать модуль на специфическую задачу, игнорируя при этом другие аспекты HTTP, не касающиеся данной задачи.

К преимуществам HTTP сервера Apache, необходимо отнести высокий уровень надежности, гибкие настройки, свободный доступ к программе. В частности, к нему можно подключать большое количество внешних модулей, систем управления базами данных и т.п. Apache поддерживает интернет-протокол IPv6.

Также к достоинствам Apache, необходимо отнести регулярно выпускаемые обновления и патчи (заплатки), которые позволяют быстро устранить различные проблемы с работой или безопасностью.

Удобство и легкость настройки этого программного обеспечения, делают его одним из самых популярных вариантов как для начинающих, так и для опытных веб-мастеров.

Все недостатки Apache, являются логическим продолжением его достоинств. Наиболее вескими недостатками являются:

1. Возможные проблемы с производительностью на высоконагруженных сайтах с большим трафиком, «тяжелым» контентом или приложениями, которые требуют высоких вычислительных мощностей.

2. Большое количество параметров настройки может привести к возникновению уязвимостей, из-за невнимательности при конфигурации.
3. Существует вероятность того, что в модули от независимых разработчиков будет внедрен вредоносный код.

### **3.2.1. Ядро веб-сервера Apache**

Ядро HTTP сервера Apache разрабатывается фондом Apache Software Foundation, который поддерживает огромное количество разработчиков по всему миру. Его основными функциями являются:

1. Передача данных по HTTP.
2. Обработка файлов.
3. Загрузка и поддержка модулей.

Сервер может функционировать без дополнительных модулей, однако в этом случае, его возможности крайне ограничены.

### **3.2.2. Конфигурация HTTP сервера Apache**

Конфигурацию Apache, можно разделить на три основных уровня:

1. Конфигурация сервера.
2. Конфигурация виртуального хоста.
3. Конфигурация уровней каталога.

Все настройки осуществляются при помощи специальных текстовых файлов, со своим собственным языком, который основан на блоках директив. С их помощью, могут быть изменены практически все параметры ядра.

Большинство модулей, имеет свои собственные параметры. Также они достаточно часто используют для своей работы настроечные файлы операционных систем.

Ряд настроек можно задавать с помощью параметров командной строки.

### **3.2.3. Сервер приложений как модуль Apache**

Можно сказать, что Apache – веб-сервер "по умолчанию". Рассмотрим любой массовый хостинг – там окажется Apache, возьмем любое веб-приложение – настройки по умолчанию выполнены под Apache.

Apache не является венцом технологий с технологической точки зрения, но именно он представляет золотую середину – прост, понятен, гибок в настройках, универсален.

Часто можно услышать, что Apache уже давно неактуален, все продвину-

тые программисты и веб-мастера уже работают с использованием Nginx и т.д. и т.п., поэтому поясним данный момент более подробно. Все популярные CMS из коробки сконфигурированы для использования совместно с Apache, это позволяет сосредоточить все внимание на работу именно с веб-приложением, исключив из возможного источника проблем веб-сервер.

Все популярные форумы тоже подразумевают в качестве веб-сервера Apache и большинство советов и рекомендаций будут относиться именно к нему. В тоже время, альтернативные веб-сервера, как правило, требуют более тонкой и тщательной настройки, как со стороны веб-сервера, так и со стороны веб-приложения. При этом пользователи данных продуктов обычно гораздо более опытные и типовые проблемы в их среде не обсуждаются. В итоге может сложиться ситуация, когда ничего не работает и спросить не у кого. С Apache такого гарантированно не произойдет.

Особенное положение, отводимое Apache в разнообразии веб-серверов, обусловлено наличием уникального (собственного) решения со стороны разработчиков: в то время как CGI предлагал абстрагироваться от конкретных решений, сосредоточившись на универсальном шлюзе, в Apache поступили по-другому – максимально интегрировали веб-сервер и сервер приложений.

Если запустить сервер приложений как модуль веб-сервера в общем адресном пространстве, то мы получим более простую схему, представленную на рис. 3.7.

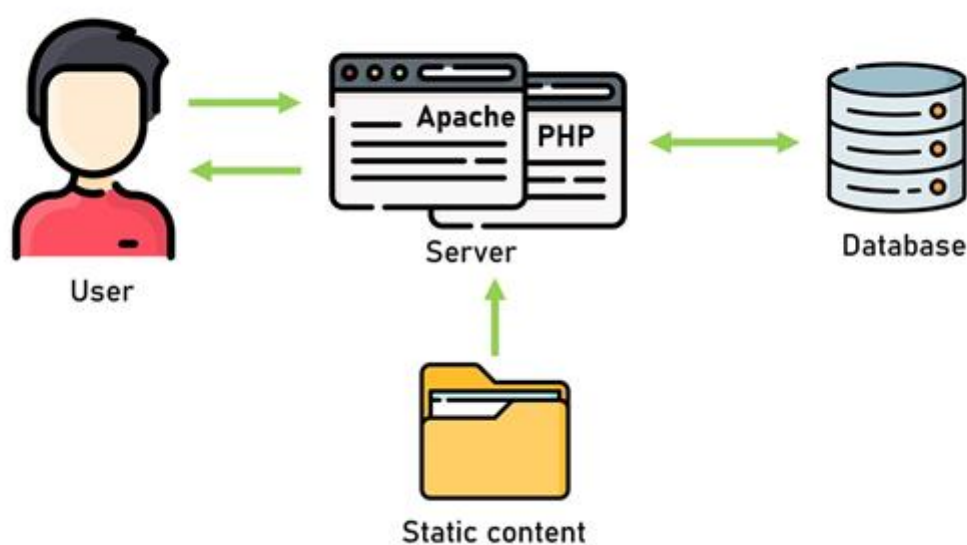


Рисунок 3.7. Схема взаимодействия серверов Apache и PHP



Основное конкурентное преимущество данной схемы в том, что чем проще схема и меньше в ней элементов, тем проще и дешевле сопровождать ее и обслуживать, тем меньше в ней точек отказа. Если для единичного сервера это может быть не так важно, то в рамках хостинга это весьма значительный фактор.

Второе преимущество – производительность. Благодаря работе в едином адресном пространстве, по производительности сервера приложений Apache + mod\_php всегда будет на 10-20% быстрее любого иного веб-сервера + FastCGI (или иное CGI решение). Но также следует помнить, что скорость работы сайта обусловлена не только производительностью сервера приложений, но и рядом иных условий, в которых альтернативные веб-сервера могут показывать значительно лучший результат.

И еще одно, достаточно серьезное преимущество, это возможность настройки сервера приложений на уровне отдельного сайта или пользователя. Вернемся немного назад: в FastCGI/CGI-схемах сервер приложений – это отдельная служба, со своими, отдельными, настройками, которая даже может работать от имени другого пользователя или на другом хосте. С точки зрения администратора одиночного сервера или какого-нибудь крупного проекта – это отлично, но для пользователей и администраторов хостинга – не очень.

Развитие интернета привело к тому, что количество возможных веб-приложений (CMS, скриптов, фреймворков и т.п.) стало очень велико, а низкий порог вхождения привлек к сайтостроению большое количество людей без специальных технических знаний. В тоже время разные веб-приложения могли требовать различной настройки сервера приложений. Как быть?

Решение нашлось довольно просто. Так как сервер-приложений теперь часть веб-сервера, то можно поручить последнему управлять его настройками. Традиционно для управления настройками Apache помимо конфигурационных файлов применялись файлы `htaccess`, которые позволяли пользователям писать туда свои директивы и применять их к той директории, где расположен данный файл и ниже, если там настройки не перекрываются своим файлом `htaccess`. В режиме `mod_php` данные файлы позволяют также изменять многие опции PHP для отдельного сайта или директории.

Для принятия изменений не требуется перезапуск веб-сервера и в случае ошибки перестанет работать только этот сайт (или его часть). Кроме того, внести изменения в простой текстовый файл и положить его в папку на сайте под силу даже неподготовленным пользователям и безопасно для сервера в целом.

Сочетание всех этих преимуществ и обеспечило Apache столь широкое

применение и статус универсального веб-сервера. Другие решения могут быть быстрее, экономичнее, лучше, но они всегда требуют настройки под задачу, поэтому применяются в основном в целевых проектах, в массовом сегменте безальтернативно доминирует Apache.

Перейдем к недостаткам. Некоторые из них просто являются обратной стороной достоинств. Тот факт, что сервер приложений является частью веб-сервера дает плюсы в производительности и простоте настройки, но в тоже время ограничивает нас как с точки зрения безопасности – сервер приложений всегда работает от имени веб-сервера, так и в гибкости системы, мы не можем разнести веб-сервер и сервер приложений на разные хосты, не можем использовать сервера с разными версиями скриптового языка или разными настройками.

Второй минус – более высокое потребление ресурсов. В схеме с CGI сервер приложений генерирует страницу и отдает ее веб-серверу, освобождая ресурсы, связка Apache + mod\_php держит ресурсы сервера приложений занятыми до тех пор, пока веб-сервер не отдаст содержимое страницы клиенту. Если клиент медленный, то ресурсы будут заняты на все время его обслуживания. Именно поэтому перед Apache часто ставят Nginx, который играет роль быстрого клиента, это позволяет Apache быстро отдать страницу и освободить ресурсы, переложив взаимодействие с клиентом на более экономичный Nginx.

#### **3.2.4. Многопроцессорные модели (MPM)**

Для веб-сервера Apache существует множество моделей симметричной многопроцессорности. Вот основные из них:

Таблица 3.1. Типы симметричной многопроцессорности веб-сервера Apache

Название	Разработчик	Описание	Назначение	Статус
worker	Apache Software Foundation	Гибридная многопроцессорно-многопоточная модель. Сохраняя стабильность многопроцессорных решений, она позволяет обслуживать большое число клиентов с минимальным использованием ресурсов.	Среднезагруженные веб-серверы.	Стабильный.

Продолжение табл. 3.1.

Название	Разработчик	Описание	Назначение	Статус
pre-fork	Apache Software Foundation	MPM, основанная на предварительном создании отдельных процессов, не использующая механизм threads.	Большая безопасность и стабильность за счёт изоляции процессов друг от друга, сохранение совместимости со старыми библиотеками, не поддерживающими threads.	Стабильный.
Perchild	Apache Software Foundation	Гибридная модель, с фиксированным количеством процессов.	Высоконагруженные серверы, возможность запуска дочерних процессов используя другое имя пользователя для повышения безопасности.	В разработке, нестабильный.
winnt	Apache Software Foundation	Многопоточная модель, созданная для операционной системы Microsoft Windows.	Серверы под управлением Windows Server.	Стабильный.
Apache-ITK	Steinar H. Gunderson	MPM, основанная на модели prefork. Позволяет запуск каждого виртуального хоста под отдельными uid и gid.	Хостинговые серверы, серверы, критичные к изоляции пользователей и учёту ресурсов.	Стабильный.
peruser	Sean Gabriel Heacock	Модель, созданная на базе MPM perchild. Позволяет запуск каждого виртуального хоста под отдельными uid и gid. Не использует потоки.	Обеспечение повышенной безопасности, работа с библиотеками, не поддерживающими threads.	Стабильная версия от 4 октября 2007 года, экспериментальная – от 10 сентября 2009 года.

Окончание табл. 3.1.

Название	Разработчик	Описание	Назначение	Статус
Event	Apache Software Foundation	Модель использует threads и thread-safe polling основана на worker. предназначен для одновременного обслуживания большего количества запросов путем передачи некоторой обработки в потоки слушателей, освобождая рабочие потоки для обслуживания новых запросов.	Обеспечение повышенной производительности. не очень хорошо работает на старых платформах, в которых отсутствует хорошая многопоточность, но требование EPoll или KQueue делает это спорным.	Стабильный.

### 3.3. Архитектура сервера nginx

Nginx – веб-сервер и почтовый прокси-сервер.

Игорь Сысоев начал разработку в 2002 году. Осенью 2004 года вышел первый публично доступный релиз. С июля 2011 работа над nginx продолжается в рамках компании Nginx.

Nginx позиционируется как простой, быстрый и надежный сервер, не перегруженный функциями.

Применение nginx целесообразно прежде всего для статических веб-сайтов и как обратного прокси-сервера перед динамическими сайтами.

#### 3.3.1. Основной функционал

HTTP-сервер:

- 1) обслуживание неизменяемых запросов, индексных файлов, автоматическое создание списка файлов, кэш дескрипторов открытых файлов;
- 2) акселерированное проксирование без кэширования, простое распределение нагрузки и отказоустойчивость;
- 3) поддержка кеширования при акселерированном проксировании и FastCGI;
- 4) акселерированная поддержка FastCGI и memcached-серверов, простое распределение нагрузки и отказоустойчивость;
- 5) модульность, фильтры, в том числе сжатие (gzip), byte-ranges (докач-

- ка), chunked-ответы, HTTP-аутентификация, SSI-фильтр;
- б) несколько подзапросов на одной странице, обрабатываемые в SSI-фильтре через прокси или FastCGI, выполняются параллельно;
- 7) поддержка SSL;
- 8) поддержка PSGI, WSGI;
- 9) экспериментальная поддержка встроенного Perl.

#### SMTP/IMAP/POP3-прокси сервер

- 1) перенаправление пользователя на SMTP/IMAP/POP3-бэкенд с использованием внешнего HTTP-сервера аутентификации;
- 2) простая аутентификация (LOGIN, USER/PASS);
- 3) поддержка SSL и STARTTLS.

### 3.3.2. Архитектура

В nginx рабочие процессы обслуживают одновременно множество соединений, мультиплексируя их вызовами операционной системы select, epoll или kqueue. Рабочие процессы выполняют цикл обработки событий от дескрипторов. Полученные от клиента данные разбираются с помощью конечного автомата. Разобранный запрос последовательно обрабатывается цепочкой модулей, задаваемой конфигурацией. Ответ клиенту формируется в буферах, которые хранят данные либо в памяти, либо указывают на отрезок файла. Буфера объединяются в цепочки, определяющие последовательность, в которой данные будут переданы клиенту. Если операционная система поддерживает эффективные операции ввода-вывода, такие, как writev и sendfile, то nginx применяет их по возможности.

Алгоритм работы HTTP-сервера выглядит следующим образом:

- 1) получить очередной дескриптор из kevent;
- 2) прочитать данные из файла и записать в socket, используя либо write/read, либо используя системный вызов sendfile, выполняющий те же действия, но в пространстве ядра;
- 3) перейти к шагу 1.

Конфигурация HTTP-сервера nginx разделяется на виртуальные серверы (директива «server»). Виртуальные серверы разделяются на location'ы («location»). Для виртуального сервера возможно задать адреса и порты, на которых будут приниматься соединения, а также имена, которые могут включать «\*» для обозначения произвольной последовательности в первой и последней части либо задаваться регулярным выражением.

Location'ы могут задаваться точным URI, частью URI либо регулярным

выражением. Location'ы могут быть сконфигурированы для обслуживания запросов из статического файла, проксирования на fastcgi/memcached сервер.

Для эффективного управления памятью nginx использует пулы. Пул - это последовательность предварительно выделенных блоков динамической памяти. Длина блока варьируется от 1 до 16 килобайт. Изначально под пул выделяется только один блок. Блок разделяется на занятую область и незанятую. Выделение мелких объектов выполняется путём продвижения указателя на незанятую область с учётом выравнивания. Если незанятой области во всех блоках не хватает для выделения нового объекта, то выделяется новый блок. Если размер выделяемого объекта превышает значение константы `NGX_MAX_ALLOC_FROM_POOL` либо длину блока, то он полностью выделяется из кучи.

Таким образом, мелкие объекты выделяются очень быстро и имеют накладные расходы только на выравнивание.

### **3.4. Apache vs Nginx: практический взгляд**

Одно из самых существенных отличий между Apache и Nginx состоит в том, как они обрабатывают соединения и отвечают на различные виды трафика.

#### **3.4.1. Apache**

Apache предоставляет несколько модулей мультипроцессинга (multi-processing modules, MPM), которые отвечают за то как запрос клиента будет обработан. Это позволяет администраторам определять политику обработки соединений. Ниже представлен список MPM-модулей Apache:

`mpm_prefork` – этот модуль создает по одному процессу с одним потоком на каждый запрос. Каждый процесс может обрабатывать только одно соединение в один момент времени. Пока число запросов меньше числа процессов этот MPM работает очень быстро. Однако производительность быстро падает, когда число запросов начинает превосходить число процессов, поэтому в большинстве случаев это не самый лучший выбор. Каждый процесс потребляет значительный объем RAM, поэтому этот MPM сложно поддается масштабированию. Но он может быть использован вместе с компонентами, которые не созданы для работы в многопоточной среде. Например, PHP не является потокобезопасным, поэтому этот MPM рекомендуется использовать как безопасный метод работы с `mod_php`.

`mpm_worker` – этот модуль создает процессы, каждый из которых может управлять несколькими потоками. Каждый поток может обрабатывать одно соединение. Потоки значительно более эффективны чем процессы, что означает

что `mpm_worker` масштабируется значительно лучше чем `mpm_prefork`. Так как потоков больше чем процессов это означает, что новое соединение может быть сразу обработано свободным потоком, а не ждать пока освободится процесс.

`mpm_event` — этот модуль похож на `mpm_worker`, но оптимизирован под работу с `keep-alive` соединениями. Когда используется `mpm_worker` соединение будет удерживать поток вне зависимости от того активное это соединение или `keep-alive`. `Mpm_event` выделяет отдельные потоки для `keep-alive` соединений и отдельные потоки для активных соединений. Это позволяет модулю не погрязнуть в `keep-alive` соединениях, что необходимо для быстрой работы. Этот модуль был отмечен как стабильный в Apache версии 2.4.

Как следует из рассмотренного выше Apache предлагает гибкие возможности для выбора различных алгоритмов обработки соединений и запросов.

### **3.4.2. Nginx**

Nginx появился на позднее Apache, по этой причине, его разработчик был лучше осведомлен о проблемах конкурентности, с которыми сталкиваются сайты при масштабировании. Благодаря этим знаниям Nginx изначально был спроектирован на базе асинхронных неблокирующих `event-driven` алгоритмов.

Nginx создает процессы-воркеры каждый из которых может обслуживать тысячи соединений. Воркеры достигают такого результата благодаря механизму, основанному на быстром цикле, в котором проверяются и обрабатываются события. Отделение основной работы от обработки соединений позволяет каждому воркеру заниматься своей работой и отвлекаться на обработку соединений только тогда, когда произошло новое событие.

Каждое соединение, обрабатываемое воркером, помещается в `event loop` вместе с другими соединениями. В этом цикле события обрабатываются асинхронно, позволяя обрабатывать задачи в неблокирующей манере. Когда соединение закрывается оно удаляется из цикла.

Этот подход к обработке соединений позволяет Nginx'у невероятно масштабироваться при ограниченных ресурсах. Поскольку сервер однопоточный и он не создает процессы под каждое соединение, использование памяти и CPU относительно равномерно, даже при высоких нагрузках.

## **3.5. Архитектура сервера IIS**

IIS (Internet Information Services, до версии 5.1 — Internet Information Server) — проприетарный набор серверов для нескольких служб Интернета от компании Microsoft.

Основным компонентом IIS является веб-сервер, который позволяет раз-

мещать в сети интернет-сайты. IIS поддерживает протоколы HTTP, HTTPS, FTP, POP3, SMTP, NNTP.

### **3.5.1. Архитектура службы**

В IIS 6.0, доступном в составе Windows Server 2003, служба веб-сервера претерпела серьезные изменения. Был добавлен новый режим обработки запросов, называемый режимом изоляции рабочих процессов (worker process isolation mode). В этом режиме все веб-приложения, обслуживаемые сервером, работают в разных процессах, что повышает стабильность и безопасность системы. Кроме того, для приёма запросов HTTP был создан новый драйвер http.sys, который работает в режиме ядра, что ускоряет обработку каждого запроса.

Все запросы к статическому содержимому, не требующие исполнения скриптов, исполняются самим драйвером http.sys в ядре, что сближает веб-сервер IIS с серверами режима ядра.

При этом запросы к динамическому содержимому исполняются рабочим процессом и загруженными в его адресное пространство модулями. С точки зрения пути исполнения запросов не существует центрального процесса, что повышает надежность в случае отказа, вызванного ошибкой в скрипте или ином модуле исполнения. Рабочие процессы автоматически перезапускаются при возникновении ошибок.

Протокол SSL поддерживается отдельным процессом HTTP SSL, который служит мостом между протоколом TCP и драйвером http.sys.

### **3.5.2. Реализация веб-приложений для IIS**

Веб-сервер IIS поддерживает несколько различных технологий создания веб-приложений:

1. ASP.NET - разработанная Microsoft технология; для IIS это – основное на сегодняшний день средство создания веб-приложений и веб-служб. IIS 6.0 поставляется вместе с операционными системами, в которые также изначально входит .NET Framework, так что поддержка ASP.NET как будто уже встроена в IIS 6.0; для более ранних версий необходимо отдельно загрузить и установить .NET Framework.
2. ASP – предшествовавшая ASP.NET технология создания динамических веб-страниц на основе сценариев. Входит в поставку IIS начиная с версии 3.0.
3. CGI – стандартная межплатформенная низкоуровневая технология



создания динамических веб-страниц.

4. FastCGI – клиент-серверный протокол взаимодействия веб-сервера и приложения.
5. ISAPI – низкоуровневая технология, аналогичная интерфейсу модулей Apache, предоставляющая полный доступ ко всем возможностям IIS, возможность разработки веб-приложений в машинном коде и возможность переопределения части функций IIS и добавления к нему функций, как связанных с генерацией контента, так и не связанных с этим. Подсистема исполнения скриптов ASP и подсистема ASP.NET выполнены как модули ISAPI.
6. SSI – включение в одни страницы текста из других страниц. Строго говоря, веб-приложением не является, поскольку IIS поддерживает лишь ограниченный набор возможностей и без того малофункционального SSI. В частности, IIS5 поддерживает только статическое включение и игнорирует команды условного ветвления.

Сам сервер поддерживает только CGI, FastCGI, ISAPI и SSI. Все остальные технологии являются надстройками, работающими через CGI, FastCGI или ISAPI.

При помощи CGI приложения для IIS могут разрабатываться на основе практически любых, в том числе сторонних, инструментов, допускающих запись в стандартный поток вывода и чтение переменных среды - Perl, C/C++ и даже средствами интерпретатора командной строки Cmd.exe.

Технология ISAPI позволяет, с одной стороны, создавать специальные приложения для IIS, требующие особенно тесного взаимодействия с механизмом сервера, а с другой стороны, является удобной платформой для организации эффективного взаимодействия IIS с другими технологиями разработки веб-приложений – например, PHP и Perl.

## 4. ПОСТРОЕНИЕ ВЗАИМОДЕЙСТВИЯ КЛИЕНТА И СЕРВЕРА

### 4.1. Протокол HTTP

#### 4.1.1. Основные понятия и принципы HTTP-протокола

Для понимания построения взаимодействия в клиент-серверных приложениях в данном разделе рассматриваются инструменты и методы этого взаимодействия. Для начала взаимодействие в рамках интернет-ресурсов между клиентом и сервером происходит чаще всего по протоколу HTTP или с помощью его безопасной надстройки HTTPS. Рассматривается как строится данное взаимодействие, его особенности и простейшая реализация данного взаимодействия.

В основе взаимодействия потребителей и поставщиков (клиентов и серверов) лежит инициация соединения клиентом и отправка запроса серверу, который в свою очередь ожидает соединения для получения запроса, производит необходимые действия и возвращает обратно сообщение с результатом. Для реализации данного взаимодействия используется протокол HTTP - протокол прикладного уровня (верхний, 7 уровень в модели OSI) передачи данных, используемый для передачи произвольных данных. Спецификации различных версий протокола описываются в RFC. RFC (с англ. рабочее окружение или рабочее предложение от англ. Request for Comments) - документ из серии пронумерованных информационных документов Интернета, содержащих технические спецификации и стандарты, широко применяемые во всемирной сети. Данные документы не только описывают спецификацию различных версий протокола HTTP, но и других компонентов Интернета. Например, RFC 2045-2049 описывают спецификацию MIME. Новые спецификации постоянно появляются с развитием различных технологий Интернета.

Всего существует 3 основные мажорные версии протокола: HTTP 1, HTTP 2 и HTTP 3. До сих пор используются все версии данного протокола из-за их существенных различий.

Основным объектом манипулирования HTTP является URI в запросе клиента. Ресурсом может быть не только файл, но и какой-либо логический или абстрактный объект. Для различия ресурсов в сообщении передаются параметры, описывающие объект, важные для дальнейшей работы и интерпретации. Главная особенность протокола HTTP, что является важным отличием от других, — это отсутствие сохранения промежуточного состояния между парами “Запрос-ответ”. То есть клиент после отправки запроса может, не дожидаясь ответа послать следующий запрос. На уровне протокола не сохраняется никакой инфор-

мации. Компоненты, использующие HTTP, могут самостоятельно осуществлять сохранение информации о состоянии, связанной с последними запросами и ответами. Протокол HTTP устанавливает отдельную TCP-сессию на каждый запрос; в более поздних версиях HTTP разрешено делать несколько запросов в ходе одной TCP-сессии, но браузеры обычно запрашивают только страницу и включенные в неё объекты, а затем сразу разрывают TCP-сессию. Для поддержки авторизованного (не анонимного) доступа в HTTP используются cookies, в которых хранятся данные о запросах и другая сопутствующая информация на клиенте.

#### **4.1.2. История протокола HTTP**

Нулевым отсчетом для истории протокола был 1991 год, когда появилась первая основная версия HTTP/0.9. Она была разработана в ЦЕРН Тимом Бернерсом-Ли. Тим разработал HTTP протокол для облегчения доступа и создания навигации при помощи гипертекста. Стандарт HTTP/0.9 содержит в себе основы синтаксиса и семантики протокола HTTP. Тогда данный протокол решал важную проблему доступа ученых к материалам лаборатории ЦЕРН и их хранения.

Следующим шагом в развитии данного протокола было появление спецификации RFC 1945, которая представляла стандарт HTTP/ 1.0, выпущенный в 1996 году. Версия расширяла исходную версию протокола, закрепляла коды ответа и вводила новый тип данных для передачи – application/octet-stream, что фактически позволило передавать нетекстовые данные.

Уже в июне 1999 года была опубликована версия протокола 1.1, которая фактически оставалась неизменной на протяжении 16 лет. Этому протоколу соответствуют современные модернизированные спецификации RFC 7230-7235, первой спецификацией была RFC 2068.

Вторая версия протокола была анонсирована в феврале 2015 года и была закреплена вышедшей в мае того же года спецификацией RFC 7540. HTTP/2 отличается от своего предшественника тем, что основана на прикладном протоколе передачи веб-контента SPDY. Данный протокол стал полностью бинарным, была изменена технология деления данных на пакеты и логика их передачи. Также теперь HTTP/2 сервер имеет право послать то содержимое, которое еще не было запрошено клиентом. Это позволит серверу сразу выслать дополнительные файлы, которые потребуются браузеру для отображения страниц, без необходимости анализа браузером основной страницы и запрашивания необходимых дополнений. С конца 2015 года вторая версия протокола поддер-

живается всеми веб-браузерами и ее также поддерживают все основные веб-сервера: Apache, Nginx, IIS. Нужно помнить, что все версии протокола совместимы.

Официально третья версия протокола еще не вышла, т.к. нет официальной спецификации, но данный протокол уже поддерживают основные браузеры с 2021 года, а в Chrome еще с 2020 года. Третья версия протокола HTTP, разработанная компанией Cloudflare на основе UDP. Нужно учитывать, что HTTP/3 – просто новый синтаксис HTTP, который работает на протоколе IETF QUIC, мультиплексированном и безопасном транспорте на основе UDP. Но до сих пор он активно и открыто не используется. Хотя еще в 2019 году компания-разработчик его представила и начала поддерживать. Сейчас данную технологию уже начал официально поддерживать веб-сервер Nginx.

#### **4.1.3. Структура HTTP-протокола**

Структура HTTP-протокола включает в себя сообщение и соединение. HTTP-соединение – это виртуальный канал транспортного уровня, установленный с целью связи, а HTTP-сообщение – это модуль связи, состоящий из структурированной последовательности байтов. Сообщения в свою очередь делятся на HTTP-запросы (request) и на HTTP-ответы (response). Сообщение состоит из 3 основных частей: строки состояния, которая определяет тип сообщения, заголовок сообщения, состоящего из одного и более полей для передачи служебной информации и тела сообщения, которое содержит HTTP-объекты. Под HTTP-объектом понимается метainформация в форме полей заголовка объекта и содержания в форме тела объекта. Пример HTTP-запроса к тестовой странице рассматривается в листинге 4.1:

Листинг 4.1. Пример HTTP-запроса

```
GET /index.php HTTP/1.1
Host: localhost
Connection: keep-alive
Cache-Control: max-age=0
sec-ch-ua: "Yandex";v="21", " Not;A Brand";v="99", "Chromium";v="93"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
DNT: 1
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/93.0.4577.82 YaBrowser/21.9.0.1044 Yowser/2.5 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/av
```

```

if,image/webp,image/apng,*/*;q=0.8,application/signed-
exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer:
https://www.yandex.ru/clck/jsredir?from=yandex.ru;suggest;brow
ser&text=
Accept-Encoding: gzip, deflate, br
Accept-Language: ru,en;q=0.9,tr;q=0.8

```

Первая строка строка состояния или в случае запроса это стартовая строка, которая состоит из HTTP-метода, URI ресурса, к которому идет обращение, и версии протокола. В примере это HTTP-метод GET (HTTP-методы будут рассмотрены далее), ресурс /index.php и версия протокола 1.1. Далее идет список заголовков. Их существует огромное множество для передачи различной служебной информации, разные пользовательские агенты (user agent) могут отправлять разные заголовки для получения клиентом одного и того же результата. Разные пользовательские агенты, также, могут поддерживать определенный набор заголовков, которые не поддерживаются другим пользовательским агентом. Но сейчас это работает только с устаревшими пользовательскими агентами. Рассматривается расшифрование заголовков примера в следующей таблице 4.1:

Таблица 4.1. Список заголовков примера

Заголовок	Значение
Host	самый главный и перманентный заголовок, обозначает домен ресурса, к которому идет обращение
Connection	закрывать или поддерживать TCP-соединение, доля дальнейшего взаимодействия клиента или сервера
Cache-Control	задание инструкций кэширования
sec-ch-ua	информация о бренде пользовательского агента и версии
sec-ch-ua-mobile	информация о желании пользовательского агента использовать мобильный интерфейс
sec-ch-ua-platform	информация о платформе, на которой запущен пользовательский агент
DNT	запрет на отслеживание действий пользователя сайтом

Окончание табл. 4.1.

Заголовок	Значение
Upgrade-Insecure-Requests	флаг о желании клиента получить зашифрованный и аутентифицированный ответ
User-Agent	информация о приложении, операционной системе, версии пользовательского агента
Accept	типы контента, который клиент может интерпретировать
Sec-Fetch-Site	информация серверу, поступает ли запрос на ресурс от того же источника, с того же сайта, с другого сайта или является запросом, инициированным пользователем. Затем сервер может использовать эту информацию, чтобы решить, следует ли разрешить запрос
Sec-Fetch-Mode	информация о типе запроса: является ли запрос навигационным между страницами, запрос на загрузку ресурса и т.п.
Sec-Fetch-User	идентификация того, что пользователь сам инициировал переход
Sec-Fetch-Dest	информация о месте назначения запроса. Это инициатор исходного запроса на выборку, который определяет, где (и как) будут использоваться извлеченные данные
Referer	содержит URL исходной страницы
Accept-Encoding	информация о поддерживаемых клиентом типов кодирования
Accept-Language	информация о языках, понимаемых клиентом и какой из них предпочтительнее

После заголовков идет тело запроса, в рассмотренном случае оно отсутствует. Далее рассматривается HTTP-ответ сервера на данный запрос в листинге 4.1. в листинге 4.2:

Листинг 4.2. HTTP-ответ на HTTP-запрос из листинга 4.1.

```

HTTP/1.1 200 OK
Date: Thu, 07 Apr 2021 08:29:30 GMT
Server: Apache/2.4.38 (Debian)
Upgrade: h2

```

```

Connection: Upgrade, Keep-Alive
X-Powered-By: PHP/7.2.34
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 140
Keep-Alive: timeout=5, max=100
Content-Type: text/html; charset=UTF-8

```

```

<html lang="en">
<head>
<title>Hello world page</title>
    <link rel="stylesheet" href="style.css" type="text/css"/>
</head>
<body>
Hello World!
</body>
</html>

```

Первая строка состояния содержит версию протокола HTTP, код статуса ответа и расшифровка статуса ответа. Далее идет список заголовков. Они рассматриваются в следующей табл. 4.2:

Таблица 4.2. Список заголовков примеров ответа

Заголовок	Значение
Date	Дата осуществления запроса
Server	информация о сервере
Upgrade	информирование клиента о том, что сервер может работать по другой версии протокола и предлагает на нее перейти
Connection	определяет, остаётся ли сетевое соединение активным после завершения текущей транзакции (запроса)
X-Powered-By	указывает платформу приложений, на которой работает сервер
Vary	используется сервером для указания того, какие заголовки он использовал при выборе представления ресурса в алгоритме согласования контента
Content-Encoding	кодировка контента
Content-Length	длина контента
Keep-Alive	информация о установке времени ожидания и максимального количества запросов
Content-Type	важная информация о типе контента и его дополнительных признаках

В представленном тестовом случае в листинге 4.2 далее содержится код, запрашиваемой HTML-страницы, которую клиент отображает.

Основные HTTP-методы и их краткое описание представлено в следующей табл. 4.3. Следует учитывать то, что названия методов чувствительны к регистру.

Таблица 4.3. HTTP-методы

Название	Описание	Особенности/Использование
GET	запрос содержимого ресурса	можно передать с параметрами в URI. Существует 3 типа: обычный, условный, частичный, - различающихся дополнительными заголовками
POST	передача данных ресурсу	передача данных осуществляется в теле запроса, поэтому, возможно, передача не только текстовых параметров, но и файлов
HEAD	запрос ресурса	аналогичен GET, но для ответа на этот запрос тело отсутствует, т.е. присутствует только строка состояния и заголовки. Используется для запроса метаданных и проверки наличия ресурса
PUT	загрузка содержимого запроса на ресурс	в отличие от POST, где предполагается обработка содержимого, то для метода PUT клиент предполагает соответствие содержимого ресурсу, также повторные идентичные запросы будут давать тот же результат, в отличие от POST
PATCH	частичное изменение ресурса	можно грубо назвать данный метод обновляющим ресурс, но его нельзя использовать в HTML-формах
OPTIONS	определение возможностей веб-сервера или параметров соединения для конкретного ресурса	в ответ серверу следует включить заголовок Allow со списком поддерживаемых методов. Также в заголовке ответа может включаться информация о поддерживаемых расширениях
DELETE	удаляет указанный ресурс	метод не разрешен к использованию в HTML-формах



Окончание табл. 4.3.

Название	Описание	Особенности/Использование
CONNECT	преобразует соединение запроса в TCP/IP-туннель	метод не разрешен к использованию в HTML-формах
TRACE	выполняет вызов возвращаемого тестового сообщения с ресурса	метод не разрешен к использованию в HTML-формах

Неотъемлемой частью HTTP-ответа является код состояния этого ответа. Код состояния представляет собой трехзначное целое число, где первой цифрой является группа кодов состояний: 1 - информационные, 2 - успешные, 3 - перенаправление, 4 - ошибки клиента и 5 - ошибки сервера. Коды состояния являются стандартом, и они описаны в соответствующих спецификациях RFC. В следующей табл. 4.4 представлены наиболее часто используемые коды состояний:

Таблица 4.4. Основные коды состояний

Номер	Название	Описание
200	OK	Данный запрос является успешным. Отдаются запрашиваемые данные клиенту, выводится требуемый ресурс.
300	Multiple Choices	На основе данного запроса сервер не может однозначно предоставить представление ресурса и передает список альтернатив клиенту на выбор.
301	Moved Permanently	Сообщение о том, что данный ресурс перемещен навсегда на новый адрес.
400	Bad Request	Сервер обнаружил в запросе клиента синтаксическую ошибку.
401	Unauthorized	Сервер сообщает о том, что для просмотра данного ресурса пользователь должен быть авторизован.
403	Forbidden	Означает, что клиент не имеет доступа к данному ресурсу.

Номер	Название	Описание
404	Not Found	Сервер не может найти ресурс с таким идентификатором.
500	Internal Server Error	Ошибка внутри сервера.
501	Not Implemented	Сервер не поддерживает возможностей, необходимых для обработки запроса

Также важным аспектом для работы с протоколом HTTP является вопрос кэширования. Этим механизмом управляет заголовок Cache-Control, который задает необходимые параметры кэширования. Данный механизм нужен для того, чтобы не делать много одинаковых запросов на получение одних и тех же ресурсов для разгрузки канала и увеличения скорости отображения клиенту ресурсов.

Важно помнить, что в протоколе HTTP полностью отсутствуют какие-либо механизмы безопасности! Безопасной надстройкой над протоколом HTTP является расширение HTTPS, которое является связкой протоколов HTTP и SSL. В модели OSI они разнесены на два уровня (7 – программный и 6 – уровень представления), а в модели IP они находятся на одном, четвертом уровне.

#### **4.1.4. Согласование контента**

Правилом хорошего тона для разработки серверных частей интернет-ресурсов является согласование контента. Различают два основных типа согласований: управляемое сервером (англ. server-driven) и управляемое клиентом (англ. agent-driven). Данный механизм предполагает выбор и отображение представления на основе HTTP-заголовков и кодов состояния HTTP-ответов.

Упреждающее согласование или согласование на уровне сервера предполагает отправку клиентом его предпочтений в заголовках, начинающихся с Ассерта, и на основе алгоритма выбора представления сервер отправляет клиенту наиболее подходящее представление. В листинге 1 примера HTTP-запроса клиент в заголовках сообщил серверу типы интерпретируемого контента, предпочитаемые языки и типы кодирования. Как видно из заголовков ответа, начинающихся с Content, клиент получил результат, который соответствует его критериям. Наряду с очевидными плюсами данного подхода возможная утечка

данных, увеличение количества заголовков в запросе и увеличение количества представлений на сервере для более точного удовлетворения как можно большего числа разнообразных запросов являются важными недостатками данного подхода.

Другим подходом, существовавшим с начала развития протокола, является подход согласования клиентом или реактивное согласование. В этом согласовании, сталкиваясь с неоднозначным запросом (запросом, по которому невозможно выдать однозначно один ресурс из имеющихся), сервер отправляет обратно страницу, содержащую ссылки на доступные альтернативные ресурсы и клиент выбирает тот, который будет использоваться. Предоставление выбора агенту является существенным преимуществом перед согласованием на стороне сервера, но недостатками данного подхода является отсутствие стандартизации страницы выбора альтернативного ресурса, что усложняет автоматизацию, а также вместо 1 запроса нужно делать несколько для получения конечного результата.

Также возможно комбинирование обоих типов, что официально выделяется как прозрачное согласование (англ. transparent negotiation) - предпочтительный вариант комбинирования обоих типов.

#### **4.1.5. Обработка HTTP-запросов на PHP**

Обработка HTTP-запросов на PHP является неотъемлемой частью его функциональности. Для обработки запросов используются суперглобальные переменные, содержащие данные о запросе и параметрах. Основными из них являются ассоциативные массивы `$_GET` и `$_POST`. В них соответственно содержатся соответствующие параметры запросов. В следующем листинге 4.3 приведен пример обработки простого GET запроса:

Листинг 4.3. Пример обработки GET-запроса

```
<html lang="en">
<head>
<title>Hello world page</title>
</head>
<body>
<?php
if (isset($_GET['name'])) {
    echo "Hello, {$_GET['name']}!";
}
?>
</body>
</html>
```

В данном примере скрипту передается параметр 'name', определяющий

имя. Страница генерирует персонализированное сообщение на основе переданного параметра. Следует обратить внимание на то, что идет обязательная проверка на существование передаваемого параметра.

Остается вопрос про обработку других HTTP-запросов, таких как PUT, DELETE и, например, OPTIONS. Для каждого из методов не существует отдельного ассоциативного массива с параметрами, но существует общий массив `$_REQUEST` с параметрами любого запроса и массив `$_SERVER` с подробной информацией о сервере и о среде исполнения, где содержатся такие важные параметры, как: `HTTP_ACCEPT`, `HTTP_ACCEPT_ENCODING`, `HTTP_ACCEPT_LANGUAGE`, `REQUEST_SCHEME`, `REQUEST_METHOD` и другие.

Запрос OPTIONS чаще всего используется для выявления возможностей сервера, например, конкретных типов обрабатываемых запросов. В листинге 4.4 приведен пример обработки данного запроса:

Листинг 4.4. Пример обработки запроса OPTIONS

```
<?php
if (isset($_SERVER['REQUEST_METHOD'])) {
    if($_SERVER['REQUEST_METHOD']=='OPTIONS') {
        header('Allow: OPTIONS, GET, HEAD, POST');
    }
}
```

В данном примере сервер предоставляет клиенту информацию об обрабатываемых типах HTTP-запросов, за счет постановки соответствующего заголовка.

#### **4.1.6. Отправка HTTP-запросов на PHP**

В функциональность PHP также входит возможность отправлять HTTP-запросы. Например, можно воспользоваться функциональными возможностями curl, представленными в библиотеке libcurl. В листинге 4.5 представлен пример отправки GET-запроса к информационному API про покемонов:

Листинг 4.5. Пример HTTP-запроса с использованием curl

```
<?php
$myCurl = curl_init();
curl_setopt_array($myCurl, array(
    CURLOPT_URL =>
    'https://rickandmortyapi.com/api/character/5',
    CURLOPT_RETURNTRANSFER => true,
));
$response = curl_exec($myCurl);
curl_close($myCurl);
```

```
echo $response;
```

В данном примере инициализируется сеанс, данному сеансу задаются параметры: URL и значение для возврата результата передачи в качестве строки, - после выполняется запрос и закрывается сеанс.

## 4.2. Обзор парадигм API и способы их реализации

Вспоминая определение понятия API из первой главы, получим: API (Application Programming Interface — прикладной программный интерфейс) - набор функций и подпрограмм, обеспечивающий взаимодействие программ и сервисов.

В проекции Интернета, API — это продукт, который предоставляет пользователю какую-либо функциональность в зависимости от его задач. Причем важно отличать внешние интерфейсы для разработчиков, которые добавляют внешние сервисы в свои веб-приложения, такие как API GitHub, API Spotify, API VK, и данные интерфейсы как самостоятельный продукт, например, API Twilio для построения коммуникации посредством SMS, голоса и обмена сообщений. Также в рамках Интернета возможно использование внутреннего API для взаимодействия сервисов внутри экосистемы веб-сервера.

API в рамках Интернета нужно для того, чтобы использовать уже существующую функциональность в веб-приложениях без переделывания уже существующих успешных решений. Например, в веб-приложении требуется функциональность машинного перевода текста с английского языка на русский. Написать собственный переводчик очень сложно и требует продолжительного времени, но существует множество сервисов, предлагающих функциональность машинного перевода посредством API. Использование стороннего решения ускорит разработку и внедрение продукта.

Организация API, построение его архитектуры сложный процесс, требующий определенных умений и навыков. Как писал Роберт Мартин в своей книге “Чистая архитектура. Искусство разработки программного обеспечения” любая парадигма программирования даже сама архитектура — это некоторые ограничения или рамки. Но они дают общую и неизменяемую структуру, что позволяет системам оставаться поддерживаемыми, работоспособными и т.д.

Выбирая парадигму проектирования интерфейса взаимодействия, можно выделить два основных вида: request-response API и event-driven API. К первой группе принадлежат такие технологии, как Representational State Transfer (REST), Remote Procedure Call (RPC) и GraphQL. Особенностью данной группы является работа с endpoints. Endpoint в проекции Интернета — это конечный ресурс на веб-сервере, к которому идет запрос. Идет обращение к серверу, сер-

вер посылает ответ клиенту - короткая и лаконичная схема взаимодействия в Интернете с использованием протокола HTTP.

А ко второй группе принадлежат такие технологии, как WebHooks, WebSockets и HTTP Streaming. Request-response API при использовании в сервисах с быстро изменяющимися данными является существенным недостатком из-за молниеносного изменения данных и устаревания данных в запросе. Постоянные запросы перегружают систему и не являются полезными. Второй группой механизмов реализации интерфейсов являются event-driven API. То есть интерфейс, зависимый от событий и реагирующий на происходящие события. А к этой группе принадлежат такие технологии, как WebHooks, WebSockets и HTTP Streaming.

В рамках данной главы будут рассмотрены такие технологии, как REST и GraphQL.

#### **4.2.1. REST**

Концепция REST, т.е. концепция взаимодействия распределенных элементов в сети лежит в основе Интернета, хотя была сформулирована только в диссертации создателя HTTP-протокола Роя Филдинга. Он обобщил и сформулировал подход «передачи представительного состояния» («Representational State Transfer»). Филдинг описал концепцию построения распределенного приложения, при которой каждый запрос (REST-запрос) клиента к серверу содержит в себе исчерпывающую информацию о желаемом ответе сервера (желаемом представительном состоянии), и сервер не обязан сохранять информацию о состоянии клиента («клиентской сессии»). Можно провести ясную аналогию с HTTP-запросами и HTTP-ответами. Приложения, поддерживающие данную концепцию, называются RESTful приложениями. Для того, чтобы приложение таковым являлось, нужно чтобы оно поддерживало и ограничивалось шестью принципами:

1. Применение модели клиент-сервер.
2. Отсутствие промежуточного состояния и отсутствие хранения клиентского состояния на сервере. Здесь также стоит вспомнить про протокол HTTP, который поддерживает данный принцип.
3. Использование механизма кэширования.
4. Наличие унифицированного интерфейса, которому предъявляются следующие четыре ограничительных условия:
  - а. Все ресурсы как-либо идентифицируемы. Например, при использовании протокола HTTP каждый ресурс идентифициру-

ется индивидуальным URI.

- b. Ресурс имеет некоторое представление.
  - c. Каждое сообщение содержит достаточно информации, чтобы понять, каким образом его обрабатывать.
5. Клиенты изменяют состояние системы только через действия, которые динамически определены в гипермедиа на сервере.
  6. Распределение архитектуры на слои.
  7. Расширение функциональности клиента за счёт загрузки кода с сервера в виде апплетов или сценариев. Данное ограничение является необязательным.

Важно понимать, что не существует официального стандарта для написания API в парадигме REST, но существуют общие рекомендации на основе требований и ограничений самой архитектуры. Самое главное, что передача данных осуществляется в том виде, что и сами данные, т.е. нет преобразований на уровне протокола, но нет запрета на хранение данных в базе данных в одном виде, а передачу в другом.

Для иллюстрации приведем пример. Для того, чтобы реализовать API нужно понять, что будем ресурсом. Предположим, в примере это будет некая информация о пользователе.

При реализации API в RESTful стиле следует придерживаться следующий рекомендаций для соблюдения парадигмы архитектуры REST:

1. Формирование идентификатора ресурса идет с соблюдением логики интерфейса, т.е. для того, чтобы получить информацию обо всех пользователях частью URL должно являться /users.
2. Для каждого ресурса реализуются два идентификатора: один для коллекции и один для элемента коллекции.
3. Предполагается использование существительных для идентификации ресурсов: вместо getUser использовать user.
4. Использование методов HTTP-протокола в соответствии с их предполагаемой функциональностью.
5. Возвращение кодов состояния ответа в соответствии с результатом выполнения запроса.
6. Возвращении данных в виде JSON или XML.

При реализации API, особенно для работы с данными в RESTful стиле предполагается реализации группы основных операций CRUD(создание, чтение, обновление, удаление). В следующей табл. 4.5. представлен один из вариантов назначения функциональности HTTP-методам. Данный пример не явля-

ется обязательным к применению, а лишь рекомендация.

Таблица 4.5. Пример соотношения CRUD-операций и HTTP-методов

CRUD-операция	HTTP-метод	URL: /users	URL: /users/1
Create	PUT	создать нового пользователя	-
Read	GET	получить информацию о всех пользователях	получить информацию об определенном пользователе
Update	POST	пакетное обновление информации о пользователях	обновление информации об определенном пользователе
Delete	DELETE	удаление всех пользователей	удаление определенного пользователя

Хорошим примером RESTful API является API такой популярной социальной сети как Twitter. Например, для того чтобы поставить “лайк” какому-либо твиту нужно отправить соответствующий запрос, представленный в следующем листинге 4.6:

Листинг 4.6. Пример запроса к Twitter API

```
POST /1.1/favorites/create.json?id=TWEET_ID_TO_FAVORITE
HTTP/1.1
Host: api.twitter.com
Content-Type: application/json
Authorization: OAuth
oauth_consumer_key="YOUR_CONSUMER_KEY",oauth_token="USERS_ACCESS_TOKEN",oauth_signature_method="HMAC-SHA1",oauth_timestamp="1633970883",oauth_nonce="xxxx",oauth_version="1.0",oauth_signature="xxxx"
```

#### *Реализация RESTful API на PHP*

В следующем разделе предлагается пример реализации RESTful API с использованием нативного PHP. В экосистеме PHP существует множество фреймворков для веб-разработки, включающих удобную функциональность для разработки API. Такая функциональность включена в экосистему фреймворков Laravel, Symfony, Yii2. Также существует множество отдельных фреймворков для построения различного взаимодействия: Lumen, Guzzle, Slim, Leaf PHP, Swift, Ubiquity, Fat Free Framework, Comet.



Предполагается, что существует некое веб-приложение. Существует задача реализации API для управления пользователями данного сервиса. Следует уточнить, что опускаются вопросы безопасности и другие. Данный пример является лишь демонстрацией некоторых возможностей PHP.

Для обращения к API предполагается обращение `ДОМЕН/api/{функциональность API}`. Для обработки создается контроллер, который будет обрабатывать все запросы к API. Но в первую очередь нужно перенаправить все обращения к API контроллеру. Пусть контроллером будет `control.php`, который будет лежать в реальной папке `api`. Для этого в конфигурационном файле `.htaccess` нужно сделать перенаправление всех запросов к контроллеру. Пример представлен в следующем листинге 4.7:

Листинг 4.7. Пример участка кода для перенаправления запросов

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
```

Предполагается создание класса `DataBase` для работы с базой данных с соответствующими методами получения, добавления, изменения и удаления данных.

Пользователь характеризуется тремя основными полями: числовым идентификатором, именем и фамилией. Передача данных будет производится в виде параметров запросов.

Обработка запроса предполагает: получение HTTP-метода запроса, выделение параметров запроса, совершение предполагаемой функциональности и отправка результата. Примерная реализация контроллера с некоторой опущенной функциональностью представлена ниже в листинге 4.8:

Листинг 4.8. Пример реализация контроллера API

```
<?php
// класс для взаимодействия с БД
$db = new Database();
// Определяем метод запроса
$method = $_SERVER['REQUEST_METHOD'];
// Получение строки запроса
$url = $_SERVER['REQUEST_URI'];
// Анализ и формирование структуры для дальнейшей обработки
$url_data = parse_url($url);
// На основе метода запускаем функциональность
switch ($method){
    case 'PUT':
        createUser($url_data);
        break;
    case 'GET':
```

```

        getUser($url_data);
        break;
    case 'POST':
        updateUser($url_data);
        break;
    case 'DELETE':
        deleteUser($url_data);
        break;
    default:
        unknownMethod();
}
...
function createUser($data) {
    $successCreated = $db->createUser($data);
    if (isset($successCreated)) {
        header("{$_SERVER['SERVER_PROTOCOL']} 200 OK");
        echo json_encode(array(
            'name' => $data['name'],
            'surname' => $data['surname'],
            'id' => $successCreated['id']));
    } else {
        header("{$_SERVER['SERVER_PROTOCOL']} 400 Bad Request");
        echo json_encode(array(
            'error' => 'Bad Request'
        ));
    }
}
... ?>

```

Из суперглобальных массивов вытягивается метод и данные из запроса, которые обрабатываются и выделяются основные элементы запроса. На основе метода запускается соответствующая функция контроллера, выдающая после определённых манипуляций заголовки ответа и тело ответа с соответствующими данными. В примере идет создание нового пользователя и при успехе возвращение соответствующего заголовка и тела ответа с необходимой информацией.

#### 4.2.2. *GRAPHQL*

При работе с REST API можно столкнуться с рядом ограничений, которые усложняют работу. Например, если с одним API работают веб-сайт и мобильное приложения. Некоторые данные получаемые при помощи запроса для веб-сайта необходимы, а для мобильного приложения избыточны и замедляют работу. В таких случаях приходится создавать новый endpoint, возвращающий только необходимые данные. Другой проблемой REST является сложность в работе с несколькими источниками данных.

Для решения данных проблем в Facebook был разработан GraphQL.

GraphQL — это язык запросов и манипулирования данными для API с открытым исходным кодом. Рядом преимуществ данной технологии является:

1. Позволяет клиенту точно указать, какие данные ему нужны.
2. Облегчает агрегацию данных из нескольких источников.
3. Использует систему типов для описания данных.

Из недостатков стоит отметить сложности в ограничении доступа к данным с клиента.

Основой GraphQL являются документы (documents), схема (schema) и распознаватели (resolvers).

Документом называется файл, содержащий операции и фрагменты, отправляемый клиентом и обрабатываемый клиентом

Для описания данных, которые могут быть получены клиентом от сервера используется схема. Поскольку сервисы GraphQL могут быть написаны на любом языке, появляется необходимость описывать данные вне зависимости от языка программирования. Для этого используется “GraphQL schema language”.

Для получения данных используются распознаватели: для каждого типа данных создается свой распознаватель, который «знает», как получить объект этого типа.

### *GraphQL схемы*

#### Типы объектов

Основу схемы составляют описания типов объектов. Далее представлен пример задания типа Character в листинге 4.9. Name и appearsIn называются полями. Они могут быть запрошены при работе с типом Character. Для каждого поля определяется его тип. Он может быть либо типом объекта, либо скалярным типом данных.

#### Листинг 4.9. Пример задания типа

```
type Character {  
  name: String!  
  appearsIn: [Episode!]!  
}
```

#### Скалярные типы данных

GraphQL по умолчанию содержит следующие скалярные типы данных:

1. Int: 32-битное целое число со знаком.
2. Float: Значение с плавающей запятой двойной точности со знаком.
3. String: Последовательность символов UTF-8.
4. Boolean: true или false.
5. ID: Уникальный идентификатор, часто используемый для повторной

выборки объекта или в качестве ключа для кеша.

6. Enum: перечисления. Так же как и в других языках, данный тип задает ограниченный набор значений, которые могут быть применены.

В следующем листинге 4.10 представлен пример перечисления:

Листинг 4.10. Пример перечисления

```
enum LengthUnit {  
    METER  
    FEET  
}
```

Также имеется возможность задать собственные скалярные типы, но для этого необходимо определить правила валидации, сериализации и десериализации.

### *Модификаторы типов*

К каждому полю может быть применен так называемый «модификатор типа (type modifiers)». Модификатор «!» означает, что данное поле является Non-null, то есть при заполнении данного поля значением null будет выброшена ошибка. Модификатор «[]» означает, что поле является массивом. Данные модификаторы можно комбинировать между собой. Пример представлен в следующем листинге 4.11:

Листинг 4.11. Пример комбинирования модификаторов типов

```
appearsIn: [Episode!]!
```

### *Аргументы*

Каждое поле типа объекта может иметь аргументы. Все аргументы в GraphQL являются именованными, передаются по имени и могут иметь значение по умолчанию. В данном примере в листинге 4.12 поле length имеет аргумент unit типа LengthUnit со значением по умолчанию METER.

Листинг 4.12. Пример именованного аргумента

```
type Starship {  
    id: ID!  
    name: String!  
    length (unit: LengthUnit = METER): Float  
}
```

### *Интерфейсы и объединения*

Система типов GraphQL поддерживает использование интерфейсов. Интерфейс является абстрактным типом, который включает в себя определенный набор полей. Тип, реализующий данный интерфейс, должен включать в себя эти поля. Интерфейсы полезны в случаях, когда при выполнении запроса могут быть возвращены объекты разных типов. В следующем листинге 4.13 представ-

лен пример интерфейса `Character`, содержащего определенный набор полей. Типы `Human` и `Droid` имплементируют данный интерфейс, что значит содержат набор полей, представленный интерфейсом.

Листинг 4.13. Пример интерфейса

```
interface Character {
    id: ID!
    name: String!
    friends: [Character]
    appearsIn: [Episode]!
}
type Human implements Character {
    id: ID!
    name: String!
    friends: [Character]
    appearsIn: [Episode]!
    starships: [Starship]
    totalCredits: Int
}
type Droid implements Character {
    id: ID!
    name: String!
    friends: [Character]
    appearsIn: [Episode]!
    primaryFunction: String
}
```

Также для работы с объектами разных типов используются объединения (`Union`). На примере, приведенном в листинге 4.14 объект типа `SearchResult` может быть либо `Human`, либо `Droid`, либо `Starship`.

Листинг 4.14. Пример объединения

```
union SearchResult = Human | Droid | Starship
```

### *Query и mutation*

Помимо типов объектов, в схеме описываются типы операций, такие как:

1. Запрос (`query`) – возвращает запрошенные клиентом данные.
2. Изменение (`mutation`) – производит манипуляции с данными и возвращает клиенту измененные данные.
3. Подписка (`subscription`) – при обновлении данных сервер выполняет определенный в подписке запрос и передает данные клиенту.

Важно понимать, что хотя технически любая операция может быть реализована так, чтобы перезаписать данные, по соглашению предполагается, что изменения должны быть отправлены явным образом через мутации. Здесь можно провести аналогию с REST, где предполагается, что GET-запросы не используются для изменения данных.

Также между `Query` и `Mutation` есть еще одно очень важное различие: поля

query выполняются параллельно, поля mutation – последовательно.

Они описываются аналогично остальным типам, но при этом с них начинается выполнение каждого запроса. Пример представлен в следующем листинге 4.15:

Листинг 4.15. Пример типа Query

```
type Query {  
  hero(episode: Episode): Character  
  droid(id: ID!): Droid  
}
```

В каждой схеме обязательно должен присутствовать тип Query.

### *Запросы*

Обычно, GraphQL сервер имеет один endpoint, принимающий все запросы. Ответ от сервера возвращается в формате JSON. Отличительной особенностью GraphQL является то, что ответ имеет точно такую же форму, как и запрос. Например, при выполнении самого простого запроса, представленного в листинге 4.16:

Листинг 4.16. Пример запроса

```
{  
  hero {  
    name  
    appearsIn  
  }  
}
```

Будет получен следующий ответ, представленный в листинге 4.17:

Листинг 4.17. Пример ответа на запрос

```
{  
  "data": {  
    "hero": {  
      "name": "R2-D2",  
      "appearsIn": [  
        "NEWHOPE",  
        "EMPIRE",  
        "JEDI"  
      ]  
    }  
  }  
}
```

В запросе указывается название операции которая должна быть выполнена, в данном случае это hero. Далее идет выборка тех полей, которые клиент хочет получить при выполнении запроса, в данном случае это name и appearsIn. Если выбранное поле является объектом, а не скалярным типом, для него также указываются поля, которые необходимо получить. В листинге 4.17 представлен

пример более сложного запроса с более глубоким определением типов:

Листинг 4.17. Пример более сложного запроса

```
{
  hero {
    name
    friends {
      name
    }
  }
}
```

#### *Название запроса и операции*

В запросе также могут быть указаны тип операции и название запроса. Они являются необязательными, но в production-разработке рекомендуется использовать их для того, чтобы сделать код менее двусмысленным и упростить отладку на стороне сервера. Пример именованного запроса представлен в листинге 4.18:

Листинг 4.18. Пример именованного запроса

```
query HeroAndFriendsName{
  hero {
    name
    friends {
      name
    }
  }
}
```

#### *Аргументы и переменные*

Еще одним преимуществом GraphQL является то, что каждое вложенное поле может получить свой набор аргументов, благодаря этому появляется возможность избежать нескольких запросов для выборок. Пример такого запроса представлен в листинге 4.19:

Листинг 4.19. Пример запроса

```
{
  human(id: "1000") {
    name
    height(unit: METER)
  }
}
```

Так как чаще всего аргументы будут динамическими, необходимо их передавать в запрос. Составление нового запроса для изменения аргументов является не очень хорошей практикой. Для того, чтобы избежать этого, в GraphQL используются переменные. Они передаются в отдельном словаре, обычно в виде JSON. Пример запроса представлен в листинг 4.20:

#### Листинг 4.20. Пример запроса

```
query HeroNameAndFriends($episode: Episode) {  
  hero(episode: $episode) {  
    name  
    friends {  
      name  
    }  
  }  
}
```

В следующем листинге 4.21 представлен пример словаря переменных:

#### Листинг 4.21. Пример словаря переменных

```
{  
  "episode": "JEDI"  
}
```

### *Директивы*

Помимо динамического изменения аргументов может потребоваться динамическое изменение структуры запроса. В данном случае используются специальные функции, влияющие на выполнение запроса - директивы. Спецификация GraphQL включает две основных директивы:

1. `@include(if: Boolean)` Включает это поле в результат, если аргумент имеет значение `true`.
2. `@skip(if: Boolean)` Пропускает это поле, если аргумент имеет значение `true`.

Пример использования директив приведен в листинге 4.22:

#### Листинг 4.22. Пример использования директив

```
query Hero($episode: Episode, $withFriends: Boolean!) {  
  hero(episode: $episode) {  
    name  
    friends @include(if: $withFriends) {  
      name  
    }  
  }  
}
```

### *Фрагменты*

При наличии нескольких запросов, возвращающих объекты одинаковых типов, может происходить дублирование и усложнение кода. В таких случаях рекомендуется использовать фрагменты, которые позволяют создать набор полей, а затем включить их в те запросы, где это необходимо. Далее приведен пример использования фрагментов в листинге 4.23:



#### Листинг 4.23. Пример использования фрагментов

```
{
  leftComparison: hero(episode: EMPIRE) {
    ...comparisonFields
  }
  rightComparison: hero(episode: JEDI) {
    ...comparisonFields
  }
}

fragment comparisonFields on Character {
  name
  appearsIn
  friends {
    name
  }
}
```

#### *Встроенные фрагменты*

Для работы с интерфейсами и соединениями GraphQL предоставляет возможность использования встроенных фрагментов. Пример приведен в листинге 4.24. Если запрос `hero` вернул объект типа `Droid`, то из него выбирается поле `primaryFunction`, а если типа `Human`, то выбирается поле `height`.

#### Листинг 4.24. Пример использования встроенных фрагментов

```
query HeroForEpisode($ep: Episode!) {
  hero(episode: $ep) {
    name
    ... on Droid {
      primaryFunction
    }
    ... on Human {
      height
    }
  }
}
```

#### *Мета-поля*

Также при работе с соединениями у клиента может появиться необходимость узнать тип полученного объекта. Для этого используется мета-поле `__typename`. Пример использования, данного мета-поля представлен в листинге 4.25:

#### Листинг 4.25. Пример использования мета-полей

```
// запрос
{
  search(text: "an") {
    __typename
    ... on Human {
      name
    }
  }
}
```

```

    }
    ... on Droid {
        name
    }
    ... on Starship {
        name
    }
}
}
// ответ
{
  "data": {
    "search": [
      {
        "__typename": "Human",
        "name": "Han Solo"
      },
      {
        "__typename": "Human",
        "name": "Leia Organa"
      },
      {
        "__typename": "Starship",
        "name": "TIE Advanced x1"
      }
    ]
  }
}

```

### *Реализация GraphQL сервера на PHP*

Для каждого из языков программирования существует множество библиотек для работы с GraphQL. Одной из самых популярных библиотек для PHP является `webonyx/graphql-php`.

Прежде всего для работы GraphQL сервера необходима схема. При использовании библиотеки `webonyx/graphql-php` схема может быть описана как при помощи GraphQL Schema Language, так и при помощи PHP.

Для использования GraphQL Schema Language требуется создать файл с расширением `.graphql`, содержащий схему, пример представлен в листинге 4.26:

Листинг 4.26. Пример файла с расширением `.graphql`

```

type Query {
  echo(message: String!): String!
}

type Mutation {
  sum(x: Int!, y: Int!): Int!
}

```

Затем данную схему требуется преобразовать в объект `php`. Для этого подключается файл `GraphQL\Utils\BuildSchema` и используется функция `build`. Рас-

познаватели для каждого из полей собираются в словарь. Затем происходит получение запроса. Для обработки запроса вызывается функция `executeQuery` из файла `GraphQL\GraphQL`. Данные действия представлены в листинге 4.27:

Листинг 4.27. Пример обработки запроса

```
<?php
use GraphQL\Utils\BuildSchema;

$contents = file_get_contents('schema.graphql');
$schema = BuildSchema::build($contents);

$rootValue = [
    'echo' => static fn (array $rootValue, array $args):
string => $args['message'],
    'sum' => static fn (array $rootValue, array $args):
int => $args['x'] + $args['y']
];

$rawInput = file_get_contents('php://input');
$input = json_decode($rawInput, true);
$query = $input['query'];
$variableValues = $input['variables'] ?? null;

use GraphQL\GraphQL;
$result = GraphQL::executeQuery($schema, $query, $rootValue,
null, $variableValues);
```

При описании схемы при помощи РНР чаще всего каждый тип задается как класс, наследующий класс `ObjectType`. В таком случае поля задаются как словарь `'fields'`, а распознаватель задается как элемент словаря для каждого из полей. Пример представлен в листинге 4.28:

Листинг 4.28. Пример создания типа

```
use GraphQL\Type\Definition\ObjectType;
class QueryType extends ObjectType
{
    public function __construct()
    {
        $config = [
            'fields' => [
                'hello' => [
                    'type' => Type::string(),
                    'resolve' => function () {
                        return 'Привет, GraphQL!';
                    }
                ]
            ]
        ];
        parent::__construct($config);
    }
}
```

Для того, чтобы при использовании типов не создавать каждый раз новый объект, создается отдельный класс, который выступает в роли «фабрики». В следующем примере в листинге 4.29 приведен файл, содержащий 2 типа: 1 скалярный и 1 пользовательский.

Листинг 4.29. Пример фабрики

```
<?php
namespace App;
use App\Type\QueryType;
use GraphQL\Type\Definition\Type;

class Types
{
    private static $query;

    public static function query()
    {
        return self::$query ?: (self::$query = new QueryType());
    }

    public static function string()
    {
        return Type::string();
    }
}
```

Объект схемы в таком случае создается следующим образом, представленным в листинге 4.30:

Листинг 4.30. Задание объекта схемы

```
$schema = new Schema([
    'query' => Types::query()
]);
```

### 4.2.3. Протокол SOAP

Протокол SOAP – набор правил для обмена структурированными сообщениями в распределенной вычислительной среде. Это набор правил для обмена сообщениями с использованием протоколов прикладного уровня, чаще всего HTTP. Данный протокол поддерживается консорциумом Всемирной паутины - организацией, разрабатывающей и внедряющей технологические стандарты для Всемирной паутины. Если коротко, то SOAP может расширить HTTP для обмена сообщениями в формате XML.

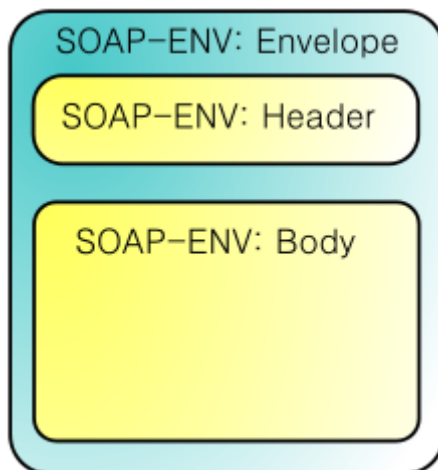
В данный момент самой актуальной версией протокола является версия 1.2. Для первой версии протокола аббревиатура SOAP расшифровывалась, как Simple Object Access Protocol. В версии 1.2 были внесены изменения в синтак-

сис и также спецификация версии 1.2 дает дополнительную (или поясняющую) семантику положений спецификации предыдущей версии 1.1. В спецификации второй версии аббревиатура не расшифровывается.

Важной частью протокола является SOAP-процессор. В общем случае это обработчик на сервере, отвечающий за обработку SOAP-сообщения в XML формате.

#### *Структура SOAP-сообщения*

Структура SOAP-сообщения представлена на рис. 4.1:



*Рисунок 4.1. Структура SOAP-сообщения*

SOAP-сообщение состоит из:

1. Envelope(конверт) - корневой элемент, который определяет сообщение и пространство имен, используемое в документе.
2. Header(заголовок) - содержит атрибуты сообщения, например: информация о безопасности или о сетевой маршрутизации.
3. Body(тело) - содержит сообщение, которым обмениваются приложения.
4. Fault(Неисправность) - необязательный элемент, который предоставляет информацию об ошибках, которые произошли при обработке сообщений.

Пример SOAP-сообщения для общения с сервисом Яндекс.Директ представлен в следующем листинге 4.31:

#### *Листинг 4.31. Пример SOAP-сообщения*

```
POST /v4/soap/ HTTP/1.1
Content-Length: 686
Content-Type: text/xml; charset=utf-8
SOAPAction: "API#GetClientInfo"
Host: api.direct.yandex.ru

<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
```

```

xmlns:ns0="API"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  <SOAP-ENV:Header>
    <locale>ru</locale>
    <token>0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f</token>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <ns0:GetClientInfo SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    <params soapenc:arrayType="xsd:string[]">
      <xsd:string>agrom</xsd:string>
      <Login>agrom</Login>
    </params>
  </ns0:GetClientInfo>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Данный пример иллюстрирует вызов метода `GetClientInfo` для получения данных о пользователе «agrom». Заголовок содержит элемент `locale`, устанавливающий русский язык для ответных сообщений, и элемент `token` – авторизационный токен, выданный OAuth-сервером Яндекса с согласия пользователя. Ниже в листинге 4.32 приведен пример SOAP-ответа:

#### Листинг 4.32. Пример SOAP-ответа

```

<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"

xmlns:namesp2="http://namespaces.soaplite.com/perl"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  <SOAP-ENV:Body>
    <namesp3:GetClientInfoResponse xmlns:namesp3="API">
      <SOAP-ENC:Array xsi:type="namesp2:ArrayOfClientInfo"
SOAP-ENC:arrayType="namesp2:ClientInfo[1]">
        <item xsi:type="namesp2:ClientInfo">
          <SendAccNews xsi:type="xsd:string">Yes</SendAccNews>
          <Discount xsi:nil="true" xsi:type="xsd:float"/>
          <Login xsi:type="xsd:string">agrom</Login>
          <Email xsi:type="xsd:string">agrom@yandex.ru</Email>
          <Role xsi:type="xsd:string">Client</Role>
          <FIO xsi:type="xsd:string">Andrew Smith</FIO>
          <DateCreate xsi:type="xsd:date">2011-01-
06</DateCreate>

```

```

        <StatusArch xsi:type="xsd:string">No</StatusArch>
        <SendNews xsi:type="xsd:string">Yes</SendNews>
        <Phone xsi:nil="true" xsi:type="xsd:string"/>
        <NonResident xsi:type="xsd:string">No</NonResident>
        <SendWarn xsi:type="xsd:string">Yes</SendWarn>
    </item>
</SOAP-ENC:Array>
</namespace3:GetClientInfoResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

В примерах запроса и ответа, представленных в листингах 4.31 и 4.32 можно заметить ярко выраженную структуру SOAP-сообщений. Далее будут подробнее рассмотрены элементы сообщения.

SOAP-конверт — это механизм упаковки. SOAP-конверт указывает на начало и конец сообщения, чтобы получатель знал, когда было получено все сообщение. Каждый конверт содержит ровно одно SOAP-тело. Описывается с помощью элемента `Envelope` с обязательным пространством имен <http://www.w3.org/2003/05/soap-envelope> для версии 1.2 и <http://schemas.xmlsoap.org/soap/> для версии 1.1. Важно понимать, что SOAP-процессор, совместимый с v1.1, генерирует ошибку при получении сообщения, содержащего пространство имен конверта v1.2. `Envelope` может иметь необязательный дочерний элемент `Header` с тем же пространством имен — заголовок. Если этот элемент присутствует, то он должен быть первым прямым дочерним элементом конверта. Элементы `Header` и `Body` могут содержать элементы из различных пространств имен.

SOAP-заголовок — это механизм расширения, который обеспечивает способ передачи информации в сообщениях SOAP, которая не является полезной нагрузкой приложения. Такая "контрольная" информация включает, например, директивы о передаче или контекстуальную информацию, связанную с обработкой сообщения. Это позволяет расширять сообщение SOAP в зависимости от конкретного приложения.

Элемент `SOAP-Body` обязательно записывается сразу за элементом `Header`, если он есть в сообщении, или первым в SOAP-сообщении, если заголовок отсутствует. В элемент `Body` можно вложить произвольные элементы, спецификация никак не определяет их структуру. Определен только один стандартный элемент, который может быть в теле сообщения — элемент сообщения `Fault`, содержащий сообщение об ошибке.

#### *SOAP-сообщения с вложениями*

Ограничение на использование только текстовых данных, заключенных в теги, отбрасывало технологию в стек неиспользуемых по причине ограничен-

ной функциональности. Это могут быть данные в форматах каких-то приложений, мультимедийные данные и т.п. Но в SOAP возможна передача данных нетекстовых типа, рассматриваемых в двоичном виде. Структура сообщения с вложением представлена на рис. 4.2:

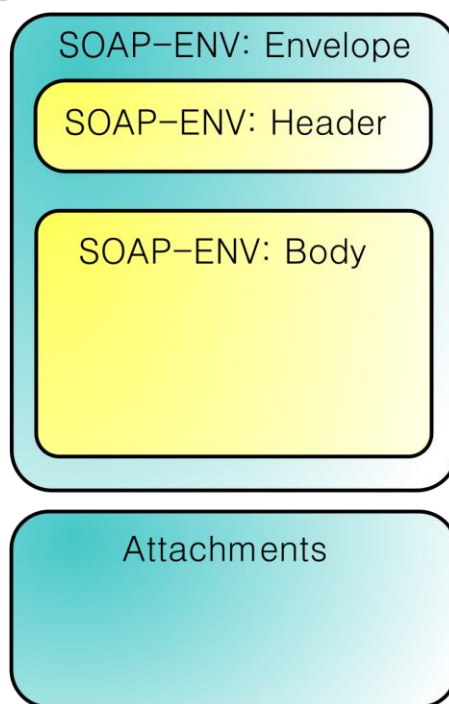


Рисунок 4.2. Структура сообщения с вложением

Данные в двоичном представлении включаются в сообщение в виде «вложения». В спецификации описаны правила включения SOAP-сообщения в MIME-сообщение типа multipart/related и правила пересылки его по протоколу HTTP.

Этот протокол определяет пересылку SOAP-сообщения внутри MIME-сообщения, состоящего из нескольких частей. Первая часть MIME-сообщения сообщения — часть SOAP — содержит XML: конверт SOAP с вложенными в него заголовком и телом сообщения. Остальные части - вложения - содержат данные в любом формате, двоичном или текстовом. Каждая часть предваряется MIME-заголовком, описывающим формат данных части и содержащим идентификатор части (Content-ID). По этому идентификатору тело SOAP-сообщения может ссылаться на вложения (href).

#### *Реализация SOAP на PHP*

В данном разделе будет предложена реализация обработчика SOAP-сообщений на сервере.

В первую очередь, важно для корректной работы SOAP-модуля необходим PHP-модуль libxml. Это означает, что также необходима передача опции --with-libxml, или до PHP 7.4 --enable-libxml, хотя неявно это уже сделано, так как



поддержка libxml по умолчанию включена. Если сборка PHP проводилась вручную, то важно установить данную библиотеку и включить расширение данное. В файле `php.ini` настройки данного расширения представлены в листинге 4.33:

Листинг 4.33. Раздел файла `php.ini` с настройками модуля SOAP

```
[soap]

; включает или отключает кэширование WSDL
; http://php.net/soap.wsdl-cache-enabled
soap.wsdl_cache_enabled = "1"

; задает имя директории в которой SOAP-расширение будет хра-
нить кэшированные файлы
; http://php.net/soap.wsdl-cache-dir
soap.wsdl_cache_dir = "/tmp"

; (время жизни) устанавливает время(в секундах) которое файлы
из кэша могут использоваться
; http://php.net/soap.wsdl-cache-ttl
soap.wsdl_cache_ttl = 86400

; задает максимальный размер кэша
soap.wsdl_cache_limit = 5
```

После настройки модуля следующим этапом разработки сервиса является создание файла WSDL.

Язык описания веб-служб (WSDL) — это язык описания интерфейса на основе XML, который используется для описания функций, предлагаемых веб-службой. Аббревиатура также используется для любого конкретного описания WSDL веб-службы (также называемого файлом WSDL), которое предоставляет машиночитаемое описание того, как может быть вызвана служба, какие параметры она ожидает и какие структуры данных она возвращает. Поэтому его назначение примерно аналогично назначению сигнатуры типа в языке программирования.

В создаваемом сервисе предполагается одна функция для получения некого кода по имени. В следующем листинге 4.34 представлен файл `wsdl`, описывающий данную функциональность.

Листинг 4.34. Содержание файла `userservice.wsdl`

```
<?xml version = '1.0' encoding = 'UTF-8' ?>
<definitions name = 'UserService'
    targetNamespace = 'http://localhost/UserService'
    xmlns:tns = 'http://localhost/UserService'

    xmlns:soap = 'http://schemas.xmlsoap.org/wsdl/soap/'
    xmlns:xsd = 'http://www.w3.org/2001/XMLSchema'
```

```

xmlns:soapenc='http://schemas.xmlsoap.org/soap/encoding/'
      xmlns:wSDL='http://schemas.xmlsoap.org/wSDL/'
      xmlns='http://schemas.xmlsoap.org/wSDL/'

    <message name='getUserRequest'>
      <part name='name' type='xsd:string'/>
    </message>
    <message name='getUserResponse'>
      <part name='Result' type='xsd:int'/>
    </message>

    <portType name='UserServicePortType'>
      <operation name='getUser'>
        <input message='tns:getUserRequest'/>
        <output message='tns:getUserResponse'/>
      </operation>
    </portType>

    <binding name='UserServiceBinding'
type='tns:UserServicePortType'>
      <soap:binding style='rpc'

transport='http://schemas.xmlsoap.org/soap/http'/>
        <operation name='getUser'>
          <soap:operation soapAction=''/>
          <input>
            <soap:body use='encoded'
              encod-
ingStyle='http://schemas.xmlsoap.org/soap/encoding/'/>
          </input>
          <output>
            <soap:body use='encoded'
              encod-
ingStyle='http://schemas.xmlsoap.org/soap/encoding/'/>
          </output>
        </operation>
      </binding>

    <service name='UserService'>
      <port name='UserServicePort' bind-
ing='UserServiceBinding'>
        <soap:address loca-
tion='http://localhost/userservice.php'/>
      </port>
    </service>
  </definitions>

```

Поскольку WSDL – это XML, то в самой первой строке необходимо это обозначить. Корневой элемент файла всегда должен называться «definitions». Обычно, WSDL состоит из 4-5 основных блоков. Самый первый блок – определение веб-сервиса или другими словами – точки входа. После того, как был

определен веб-сервис и указали для него точку входа – необходимо привязать к нему поддерживаемые процедуры. Для этого перечисляются какие операции и в каком виде у будут вызываться. Т.е. для порта «UserServicePort» определена привязка под именем «UserServiceBinding», которая имеет тип вызова «rpc» и в качестве протокола передачи (транспорта) используется HTTP. После этого описываются какие процедуры (operation) поддерживаются в веб-сервисе. После этого необходимо привязать процедуру к сообщениям. Для этого указывается, что наша привязка («binding») имеет тип «UserServicePortType» и в элементе «portType» с одноименным типу именем указываем привязку процедур к сообщениям. И так, входящее сообщение (от клиента к серверу) будет называться «getUserRequest», а исходящее (от сервера к клиенту) «getUserResponse». Как и все имена в WSDL, имена входящих и исходящих сообщения – произвольные.

Следующим этапом разработки сервиса является создание файла обработчика SOAP-сообщений SOAP-процессора. Файл для реализуемой задачи представлен в следующем листинге 4.35:

Листинг 4.35. Файл userservice.php

```
<?php
...
function getUser($name){
    global $users;
    return $users[$name] ?? -1;
}

$server = new SoapServer("userservice.wsdl");
$server->addFunction("getUser");
$server->handle();
```

В данном процессоре создается новый сервер для обработки запросов по протоколу, в конструктор передается WSDL с описанием спецификации взаимодействия. К данному серверу подключается функция, предполагаемая спецификацией, и запускается обработка получаемого сообщения.

На основе созданной спецификации сообщение на сервер представляется в листинге 4.36:

Листинг 4.36. SOAP-сообщение к сервису

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  SOAP-
```

```

ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
  <getUser>
    <name xsi:type="xsd:string">Mikhail</name>
  </getUser>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

На данное сообщение приходит соответствующий ответ, представленный в листинге 4.37:

#### Листинг 4.37. SOAP-ответ на сообщение

```

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <SOAP-ENV:getUserResponse>
      <Result xsi:type="xsd:int">0</Result>
    </SOAP-ENV:getUserResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

## 5. РАБОТА С ДАННЫМИ В PHP

### 5.1. Сессии и cookies

Пользователь заходит на веб-сайт интернет-магазина. В его задачу входят выбрать себе какой-либо товар и заказать его. Он авторизуется, кладет необходимые товары в корзину, но по какой-то причине заказ не оформляет до конца и уходит с веб-сайта. Одумавшись через какое-то время, пользователь снова возвращается на веб-сайт с того же юзер-агента и видит, что авторизацию ему проходить не нужно и его товары сохранились в корзине, можно продолжать оформление заказа. Одним из вариантов поддержания такого функционала для пользователя является механизм сессий.

Сессия в рамках интернет-ресурсов — это данные о пользовательской активности, сохраняемые между запусками сценария. Вспоминая слабую сторону CGI-интерфейсов, а именно невозможность открывать длительные соединения, позволив ему при этом обмениваться данными с пользователями.

Еще хорошей иллюстрацией необходимости хранения деятельности является хранение данных выборки фильтров поиска пользователя: пользователь с помощью фильтров сделал какую-либо выборку товаров и перешел на страницу определенного товара, нажимая назад пользователь хотел бы увидеть ту же выборку товаров, что была до этого. Очень важно сохранение индивидуальных данных каждого пользователя.

Вспомним механизм работы HTTP-протокола. Каждый запрос для сервера уникален в том плане, что HTTP-протокол не предполагает хранение промежуточного состояния клиента. Поэтому существует механизм сессий для хранения данных на клиенте и на сервере.

Еще одним примером иллюстрации необходимости механизма сессий является межстраничная навигация. Предположим, пользователь авторизовался на сайте и ему была загружена страница меню сайта. Это “грубо” говоря один запрос. Теперь, когда пользователь переходит на другую страницу сайта, серверу посылается запрос на новую страницу и нужно как-то сообщить серверу, что пользователь уже авторизован на сайте и данное действие не нужно проделывать снова.

Удобная реализация функциональности хранения истории действий пользователя на интернет-ресурсе предполагает хранение сессии на веб-сервере, а также хранение cookies на клиенте. Cookies — это данные от веб-сервера, хранящиеся на клиенте браузером. Чаще всего в cookies хранятся данные аутентификации пользователя, пользовательские предпочтения и настройки (например, темная тема интернет-ресурса вместо базовой светлой), данные состояния сеан-

са пользователя, а также пользовательская статистика.

Сессия позволяет обмениваться информацией на разных страницах одного сайта или приложения, что помогает поддерживать состояние. Это позволяет серверу знать, что все запросы исходят от одного и того же пользователя, что дает возможность отображать информацию и настройки пользователя.

Для иллюстрирования механизма сессий приведем пример базовой авторизации и дальнейших действий пользователя на интернет-ресурсе:

1. Пользователь заходит на страницу авторизации на интернет-ресурсе и заполняет форму авторизации.
2. Нажатием кнопки “Войти” на сервер отправляется HTTP-запрос с данными для авторизации.
3. Сервер проверяет данные, отправленные пользователем:
  - a. Если данные авторизации верны, то сервер создает новую сессию: генерируется уникальный идентификатор сессии, а также файл на сервере, хранящий данные сессии.
  - b. Если данные неверны, то ошибка авторизации отправляется клиенту.
4. Клиенту отправляется идентификатор сессии в виде заголовка Set-cookie.
5. Клиент получает идентификатор сессии от сервера и сохраняет его в файле cookies.
6. Последующие запросы берут данные из файла cookies и отправляют их на сервер вместе с запросами. Сервер по этим данным идентифицирует пользователя.

#### **5.1.1. Работа с сессиями и cookies в PHP**

Язык программирования PHP предлагает расширенный функционал работы с функционал сессий и cookies. В файле конфигурации `php.ini` содержится целая секция, в которой описаны возможные настройки. Рассмотрим подробнее данную секцию в файле `php.ini`. Также данную информацию можно посмотреть в результате вывода функции `phpinfo()`. Также следует напомнить, что возможна настройка во время исполнения скрипта, за которую отвечают одноименные функции директив в конфигурационном файле. В следующем листинге 5.1 представлен отрывок конфигурационного файла `php.ini` с некоторым набором основных настроек поведения для работы с сессиями:

### Листинг 5.1. Урезанный отрывок файла php.ini

```
[Session]
session.save_handler = files

session.use_strict_mode = 0

session.use_cookies = 1

session.use_only_cookies = 1

session.name = PHPSESSID

session.auto_start = 0

session.cookie_lifetime = 0

session.cookie_path = /

session.cookie_domain =

session.cookie_httponly =

session.serialize_handler = php

session.gc_probability = 1

session.gc_divisor = 1000

session.gc_maxlifetime = 1440

session.referer_check =

session.cache_limiter = nocache

session.cache_expire = 180

session.use_trans_sid = 0

session.sid_length = 26

session.trans_sid_tags = "a=href,area=href,frame=src,form="

session.sid_bits_per_character = 5
```

`session.save_handler` указывает на обработчика, который отвечает за чтение и хранение данных сессии. По умолчанию обработчик `files`. Возможно создание пользовательских обработчиков, например, для того, чтобы хранить данные сессии в базе данных.

`session.use_strict_mode` указывает на использование режима строгого идентификатора (ID). Данный режим предполагает, что когда браузер отправ-

ляет серверу неинициализированный идентификатор, то сервер отправляет обратно браузеру новый идентификатор. Данный режим рекомендуется использовать для обеспечения безопасности интернет-ресурсов.

`session.use_cookies` указывает будет ли модуль использовать cookies для хранения идентификатора сессии на стороне клиента.

`session.use_only_cookies` указывает на хранение идентификатора сессии только в cookies клиента и нигде еще для обеспечения большей безопасности. По умолчанию данный режим включен.

`session.name` указывает имя, которое используется как идентификатор названия сессии в cookies. По умолчанию равно `PHPSESSID`, возможно изменение в файле конфигурации и динамически с помощью функции `session_name()`.

`session.auto_start` указывает на то, будет ли модуль сессии запускать сессию автоматически при старте. По умолчанию данный механизм отключен.

`session.cookie_lifetime` указывает на время жизни данных cookie, отправляемых браузеру. Значение по умолчанию 0 информирует о времени жизни до закрытия браузера. Важно помнить про возможное разное время сервера и клиента. Все операции, связанные с временем, работают с часами сервера.

`session.cookie_httponly` указывает на то, что данные cookie, содержащие идентификатор сессии могут передаваться только через протокол HTTP. Это еще одна мера безопасности для обеспечения недоступности идентификатора сессий для скриптовых языков, например JavaScript.

`session.serialize_handler` указывает на имя обработчика, который используется для сериализации/десериализации данных.

`session.use_trans_sid` является еще одним важным аспектом безопасности и указывает на используется ли прозрачная поддержка sid или нет. По умолчанию данный функционал отключен. Следует напомнить, что Идентификатор безопасности (от англ. sid) - структура данных переменной длины, которая идентифицирует учетную запись пользователя, группы, службы, домена или компьютера.

Если идет работа в ситуации ручного запуска сессий, то нужно вызвать функцию старта сессии. Рассмотрим механизм работы с сессиями в листинге 5.2.

Листинг 5.2. Механизм работы с сессиями

```
<?php
if(isset($_POST['login']))
    $mysqli = new mysqli("db", "user", "password", "appDB");
    $result = $mysqli->query("SELECT * FROM loginInfo WHERE
    name='{$_POST['login']}'");
    foreach ($result as $row){
```



```

        if($row['password']==$_POST['password']){
            session_start();
            $_SESSION['user_authorized'] = 1;
            header("Location: http://localhost/menu.php");
            exit;
        }
    }
    header("Location: http://localhost/index.php");

```

Для работы с данными сессии и данными cookies существует два суперглобальных массива `$_SESSION` и `$_COOKIE`. В листинге 5.2 представлен код обработчика формы авторизации в очень упрощенном и небезопасном виде. Если аутентификация пользователя проходит успешно, то вызывается функция начала сессии `session_start`, в файл сессии сохраняется данные о том, что пользователь авторизован. Для этого в суперглобальный массив `$_SESSION` записывается новое значение. Потом идет перенаправление на страницу меню интернет-ресурса. Рассмотрим в листинге 5.3 код страницы меню веб-ресурса.

Листинг 5.3. Механизм работы с cookies

```

<?php
    session_start();
?>
<html lang="en">
<head>
    <title>Menu page</title>
</head>
<body>
<?php
    if(!$_COOKIE['views']) {
        setcookie('views', 1, time()+60*60*24);
    } else{
        setcookie('views', ++$_COOKIE['views'],
time()+60*60*24);
    }
    echo "Your session id is ".session_id()."<br>";
    echo "You have visited this page {$_COOKIE['views']}<br>";
    echo "You are authorized {$_SESSION['user_authorized']}";

?>
</body>
</html>

```

Команда `session_start` не только начинает новую сессию, но и стартует уже существующую. Далее идет работа с данными файла cookie. При каждом запросе браузер передает эти данные серверу и данный файл в большей части случаев один для всего интернет-ресурса. Данные в данном файле хранятся по типу “ключ-значение”, поэтому они представлены в PHP, как суперглобальный массив `$_COOKIE`. В предложенном скрипте сначала идет проверка на существо-

вание в cookie значения, обозначающего количество посещений страницы меню интернет-ресурса. Если данного значения не существует, то используется функция `setcookie` с параметрами нового значения для файла cookie. Ее функционал предполагает отправку в HTTP-ответе заголовка `set-cookie` с необходимой информацией о новом значении в файле cookie. Если же значение в данном файле уже существует, то производится его перезапись. Параметрами функции `setcookie` являются: ключ, значение и время до которого данное значение будет существовать по истечении которого браузером данное значение из файла cookie будет удалено. Данный механизм сделан для сохранения и экономии памяти на клиентской машине.

Механизм работы с сессиями не ограничивается только созданием и запуском новой сессии. Также возможна следующая функциональность, представленная в листинге 5.4:

Листинг 5.4. Механизм управления сессиями

```
<?php
session_start();

unset($_SESSION['user_authorized']);

session_destroy();
```

Кроме создания сессии предусмотрено удаление сессии, например при разлогировании пользователя с помощью функции `session_destroy`. Также возможно удаление отдельных значений сессии с помощью функции `unset`.

Важно понимать, что сессии и cookies не взаимосвязаны полностью. Сессия имеет определенный жизненный цикл и чаще всего ей соответствует действия одного авторизованного пользователя. Разные входы на веб-ресурс пользователя однаменены разными сессиями. В отличие от механизма сессий файл cookie привязан к одному сайту. Но веб-приложение может хранить ни один файл cookie на клиентской машине, например для разделения хранимой логики. При обновлении страницы сохраняется та же сессия и тот же файл cookie.

Существует проблема, которая заключается в том, что некоторые пользователи не хотят работать с технологией cookie, а значит мы не можем хранить на клиенте идентификатор сессии, но при этом веб-приложению нужно как-то поддерживать технологию сессий для хранения необходимой ему информации. Решением данной проблемы является передача идентификатора сессии в URL. Данный метод не является безопасным, но рабочим. Для поддержки данной технологии требуется настройки конфигурации PHP, например в файле `php.ini` представленные в листинге 5.5:

#### Листинг 5.5. Отрывок файла php.ini

```
session.use_cookies = 0

session.use_only_cookies = 0

session.use_trans_sid = 1

session.cache_limiter = nocache
```

Первые две директивы обозначают отключение использования файлов cookie. Директива `session.use_trans_sid` создает константу SID (string), которая содержит либо имя сессии и идентификатор в виде `"session_name = session_id"`. Для данного подхода рекомендуется использовать протокол HTTPS.

Также возможна настройка данной конфигурации во время исполнения приведенная в листинге 5.6:

#### Листинг 5.6. Обновление конфигурации во время исполнения

```
<?php
    ini_set("session.use_cookies", 0);
    ini_set("session.use_only_cookies", 0);
    ini_set("session.use_trans_sid", 1);
    ini_set("session.cache_limiter", "");
    session_start();
```

С помощью функции `ini_set` возможна настройка во время исполнения, но важно ее проводить до выполнения какой-либо функциональности.

В PHP существует возможность настройки хранения сессии не только в файловой системе, но и в базе данных, например. Или дополнительное шифрование. Базовым обработчиком сессий является обработчик `files`, но возможно создание пользовательских обработчиков на основе реализации класса наследника `SessionHandler` или имплементации интерфейса `SessionHandlerInterface`. Для регистрации обработчика можно воспользоваться функцией `session_set_save_handler` или аналогичной настройкой в файле конфигурации PHP. В следующем листинге 5.7 рассмотрим кратко класс `SessionHandler`.

#### Листинг 5.7. Класс SessionHandler

```
class SessionHandler implements SessionHandlerInterface, SessionIdInterface {
    /* Методы */
    public close(): bool
    public create_sid(): string
    public destroy(string $id): bool
    public gc(int $max_lifetime): int|false
    public open(string $path, string $name): bool
    public read(string $id): string|false
    public write(string $id, string $data): bool
}
```

Когда стартует сессия, PHP внутренне вызовет обработчик `open` с последующим вызовом обработчика `read`, который должен вернуть закодированную строку — в точности такую, какая передавалась для сохранения. После возвращения обработчиком `read` закодированной строки, PHP декодирует её и заполнит получившимся массивом суперглобальный массив `$_SESSION`.

Когда PHP завершает исполнение скрипта (или когда вызвана функция `session_write_close()`), PHP внутренне закодирует суперглобальный массив `$_SESSION`, и передаст эти данные с идентификатором сессии функции обратного вызова `write`. После того, как отработает функция обратного вызова `write`, PHP внутренне вызовет обработчик функции обратного вызова `close`.

Когда сессия специально уничтожена, PHP вызовет обработчик `destroy` с идентификатором сессии.

PHP будет вызывать обработчик функции обратного вызова `gc` время от времени, чтобы пометить сессии как истекшие в соответствии с временем жизни сессий. Эта операция удалит все записи из постоянного хранилища, доступ к которым не осуществлялся более чем интервал времени, указанный в параметре `$lifetime`.

Вопрос безопасности сессий очень важен для формирования безопасности всего веб-приложения. Существует несколько способов утечки существующего идентификатора сессии третьим лицам. Например, инъекции JavaScript, передача идентификатора сессии в URL, перехват пакетов, физический доступ к устройству и т.д. Перехваченный идентификатор сессии позволит третьим лицам получить доступ ко всем ресурсам, связанным с данной сессией. Во-первых, передача идентификатора сессии в URL. При переходе на внешний сайт идентификатор сессии пользователя и адрес ресурса могут попасть в статистику переходов данного сайта. Во-вторых, при более активной атаке возможно прослушивание сетевого трафика злоумышленником. Если канал передачи данных не зашифрован, идентификаторы сессии будут переданы в виде простого текста. В таком случае решением является обязательное использование SSL/TLS пользователями при доступе к сайту. Для этих целей следует применять HSTS.

Одним из вариантов повышения безопасности веб-приложения является неадаптивное управление сессиями. По умолчанию в PHP адаптивное управление сессиями, которые несут не всегда оправданные риски. Если `session.use_strict_mode` включён, и обработчик сохранения сессии это поддерживает, неинициализированный сессионный ID отвергается и создается новый. Это защищает от атак, которые принуждают пользователя использовать

заранее известный ID.

Вторым вариантом поддерживающим вариантом является пересоздание идентификаторов сессий. Пересоздание идентификаторов сессий сильно уменьшает риск кражи сессии, соответственно надо на периодической основе запускать `session_regenerate_id()`. Например, пересоздавать идентификатор сессии каждые 15 минут для особо секретных данных. Даже если сессию украдут, она достаточно скоро станет истекшей и попытка её использовать приведёт к ошибке истекшей сессии. Идентификатор сессии должен пересоздаваться при повышении привилегий пользователя, например при аутентификации. Функция `session_regenerate_id()` должна вызываться до записи авторизационной информации в `$_SESSION`. (`session_regenerate_id()` сохраняет данные текущей сессии автоматически). Убедитесь, что только текущая сессия отмечена как авторизованная. Разработчики НЕ ДОЛЖНЫ полагаться на механизм истечения срока действия идентификатора сессии с помощью `session.gc_maxlifetime`. Атакующие могут периодически получать доступ к сессии для предотвращения её срока действия и продолжать использовать идентификатор жертвы, включая аутентифицированные сессии.

Третьим методом защиты является удаление сессий. Данные истекших сессий должны быть недоступны и удалены. Существующий механизм управления сессиями делает это не очень хорошо. Данные истекших сессий надо удалять так быстро, как только возможно. С другой стороны, данные активных сессий НЕ ДОЛЖНЫ удаляться сразу же. Для обеспечения этих противоречивых требований, вы ДОЛЖНЫ самостоятельно реализовать механизм контроля за истекшими сессиями на базе временных меток. Устанавливайте и управляйте временными метками жизни сессии через `$_SESSION`. Запрещайте доступ к данным истекших сессий. Если обнаружена попытка доступа к данным устаревшей сессии, снимайте статус авторизации со всех активных сессий пользователя и вынуждайте его переавторизоваться. Доступ к данным истекшей сессии может означать атаку. Для обеспечения такого поведения вы должны отслеживать все активные сессии пользователя.

Данные сессии могут быть использованы для Dos-атаки, поэтому важно правильно воспользоваться механизмом блокировок сессии для предотвращения блокировки злоумышленником. Для уменьшения риска DoS с использованием блокировки сессий, минимизируйте их. Используйте блокировку "read only", когда сессию не нужно обновлять. Используйте опцию 'read\_and\_close' с `session_start()`. `session_start(['read_and_close'=>1]);` Закрывайте сессию с помощью `session_commit()` сразу, как только вы закончили обновлять `$_SESSION`.

Механизм автоматического входа, часто предполагающий использование долгоживущий сессий, является хорошим инструментом в работе злоумышленников. Автоматический вход в систему должен реализовываться разработчиком самостоятельно. Рекомендуется устанавливать безопасные хешированные одноразовые ключи в качестве ключей автологина с помощью `setcookie()`, а также использование безопасного хеширование, посильнее чем SHA-2, например SHA-256 или выше со случайными данными из `random_bytes()` или `/dev/urandom`.

## 5.2. Работа с файлами разных форматов

Управление файловой системой входит в функциональность PHP. Для начала ознакомимся с директивами конфигурации файловой системы и потоков в PHP. Они представлены в листинге 5.8:

Листинг 5.8. Директивы настройки конфигурации управления файловой системой

```
allow_url_fopen = On

allow_url_include = Off

user_agent="PHP"

default_socket_timeout = 60

from="john@doe.com"

auto_detect_line_endings = Off

sys_temp_dir = "/tmp"
```

`allow_url_fopen` указывает на обработку объектов URL как обычных файлов. Обёртки, доступные по умолчанию, служат для работы с удалёнными файлами с использованием ftp или http протокола. Некоторые модули, например, `zlib`, могут регистрировать собственные обёртки.

`allow_url_include` указывает на возможность использования оберток `fopen`, которые поддерживают работу с URL, в функциях `include`, `include_once`, `require`, `require_once`. Данная директива объявлена устаревшей с версии 7.4.0.

`user_agent` указывает на задаваемую PHP строку “User-agent”. По умолчанию данная директива не задается и поэтому значение данной строки равно пустой строке.

`default_socket_timeout` указывает на значение тайм-аута по умолчанию (в секундах) для потоков, использующих сокет. Отрицательное значения означает бесконечное время ожидания. Бесконечные времена ожидания использовать не рекомендуется, чтобы не перегружать систему.

`from` указывает на адрес email, используемый в соединениях FTP без авторизации, а также в качестве значения заголовка `From` в HTTP соединениях при использовании `ftp` и `http` оберток, соответственно.

`auto_detect_line_endings` указывает на включения функционала PHP для определения способа завершения строк.

`sys_temp_dir` указывает на путь к директории, в которой будут храниться временные файлы.

### 5.2.1. *Потоки*

Для работы с файлами файловая система использует потоки (streams) в качестве собственного типа ресурсов. Потоки были введены как инструмент для работы с файлами, сетевого обмена, сжатия данных и выполнения других операций с помощью одного общего набора функций. Выражаясь простыми понятиями, поток (stream) — это ресурс (resource), который ведёт себя как источник непрерывной последовательности данных. Это означает, что из потока можно последовательно читать данные, равно как и записывать в него. Также возможно перемещаться (`fseek()`) в разные позиции внутри потока.

Обёртка (wrapper) - дополнительный код, который объясняет потоку особенности работы со специфическими протоколами или кодировками. Например, обёртка `http` знает, как преобразовать URL в HTTP/1.0-запрос для файла на удалённом сервере. Существует множество оберток, как встроенных в PHP изначально, так и дополнительных. Дополнительные обертки можно добавлять отдельным скриптом с помощью функции `stream_wrapper_register()`. Добавлять можно произвольное количество оберток, что делает возможности работы с потоками практически безграничными. Посмотреть список зарегистрированных на данный момент оберток можно с помощью функции `stream_get_wrappers()`. В обыкновенной поставке это следующий набор оберток: `https`, `ftp`, `compress.zlib`, `php`, `file`, `glob`, `data`, `http`, `ftp`, `phar`.

Ссылка на поток записывается в следующем виде: `scheme://target`, где `scheme` - название обертки, если имя не указано, то используется обертка по умолчанию `file`. `target` - дополнительная информация, зависящая от типа обертки. Для потоков, связанных с файловой системой это обычно путь и имя файла. Для сетевых потоков это, как правило, имя хоста (зачастую с добавлением к нему пути).

Управление файлами и локальной файловой системой на сервере является неотъемлемой частью серверного функционала приложения. В следующем листинге 5.9 рассмотрим простой пример с чтением файлов:

#### Листинг 5.9. Пример чтения файла

```
<?php
$filename = "/usr/local/something.txt";
$handle = fopen($filename, "r");
$contents = fread($handle, filesize($filename));
fclose($handle);
?>
```

В данном примере идет открытие файла для чтения с помощью функции `fopen` и чтение всего содержимого файла в строку с помощью функции `fread`, первым параметром которой является дескриптор файла, а вторым количество байт для чтения. В данном случае мы считываем весь файл за счет получения размера файла с помощью функции `filesize`. После конца работы с файлом нужно обязательно закрыть его с помощью функции `fclose`.

Для иллюстрирования просто записи в файл рассмотрим простой пример в листинге 5.10:

#### Листинг 5.10. Простой пример записи в файл

```
<?php
$filename = 'test.txt';
$somecontent = "Добавить это к файлу\n";

if (is_writable($filename)) {

    if (!$handle = fopen($filename, 'a')) {
        echo "Не могу открыть файл ($filename)";
        exit;
    }

    if (fwrite($handle, $somecontent) === FALSE) {
        echo "Не могу произвести запись в файл ($filename)";
        exit;
    }

    echo "Ура! Записали ($somecontent) в файл ($filename)";

    fclose($handle);

} else {
    echo "Файл $filename недоступен для записи";
}
?>
```

Первым действием при записи в файл является проверка существования файла и его доступности для записи с помощью функции `is_writable`. Далее следует попытка открытия файла в режиме записи в конец и при успешности попытки идет запись заранее объявленной строки в конец. После всех манипуляций файл закрываем.



PHP определяет широкий функционал для получения разнообразной информации о файле. Рассмотрим некоторую функциональность в следующем листинге 5.11:

Листинг 5.11. Информационные функции о файлах

```
<?php
$filename = 'somefile.txt';
if (file_exists($filename)) {
    echo "В последний раз обращение к файлу $filename было
    произведено: " . date("F d Y H:i:s.", filemtime($filename));
    print_r(posix_getpwuid(fileowner($filename)));
    echo filesize($filename). " ". filetype($filename);
}
echo substr(sprintf('%o', fileperms('/tmp')), -4);
?>
```

`file_exists` проверяет существование указанного файла или каталога.

`filemtime` возвращает время последнего обращения к файлу.

`fileowner` возвращает идентификатор владельца файла.

`fileperms` возвращает информацию о правах на файл. В примере идет отображение прав доступа в виде восьмеричного числа.

`filesize` возвращает размер файла в байтах.

`filetype` возвращает тип файла.

### 5.2.2. XML и DOM

Чаще всего в веб-разработке идет работа с двумя основными типами файлов, в которых не только могут удобно храниться структурированные данные, но и производится передача. Это xml и json.

Для работы с xml одной из крупных встроенных библиотек является DOM. Аббревиатура расшифровывается как Document Object Model. Данный набор функциональности поддерживает обработку как XML-документов, так и HTML-документов. Рассмотрим пример работы с данной функциональностью на примере, представленном в листинге 5.12:

Листинг 5.12. Пример использования DOM

```
<?php
$doc = new DOMDocument('1.0');
// мы хотим красивый вывод
$doc->formatOutput = true;

$root = $doc->createElement('book');
$root = $doc->appendChild($root);

$title = $doc->createElement('title');
$title = $root->appendChild($title);
```

```

$text = $doc->createTextNode('Это заголовок');
$title->appendChild($text);

echo "Сохранение всего документа:\n";
echo $doc->saveXML() . "\n";

```

Сначала создается новый документ `DOMDocument` с параметром конструктора версией разметки `xml`, также вторым необязательным параметром является тип кодировки. Далее идет задание красивого форматированного вывода за счет изменения соответствующего свойства документа. Следующим шагом идет создание новых узлов документа: `book` и `title`. Документ содержит узлы, которые являются экземплярами класса `DOMElement`. Дальше идет добавление дочерних узлов в конец списка потомков с помощью метода `appendChild`. То есть мы создаем верхний узел – Документ, потом добавляем ему потомка первого уровня `book`, которому добавляем потомка `title`. После создаем для документа текстовый узел 'Это заголовок', которому добавляем в список потомков узла `title`. Результат представлен в листинге 5.13:

Листинг 5.13. Результат выполнения скрипта листинга 5.12

```

<!--?xml version="1.0"?-->
<book>
  <title>Это заголовок</title>
</book>

```

Представленный является максимально простым. В следующем листинге 5.13 рассмотрим следующий обратный пример:

Листинг 5.13. Пример использования DOM

```

<?php
$xml = <<< XML
<?xml version="1.0" encoding="utf-8"?>
<books>
  <book>Шаблоны корпоративных приложений</book>
  <book>Приёмы объектно-ориентированного проектирования. Пат-
терны проектирования</book>
  <book>Чистый код</book>
</books>
XML;

$dom = new DOMDocument;
$dom->loadXML($xml);
$books = $dom->getElementsByTagName('book');
foreach ($books as $book) {
    echo $book->nodeValue, PHP_EOL;
}
?>

```

В примере объявляется строка, содержащая содержимое xml-файла. Создается пустой DOMDocument и в него с помощью функции load загружается xml-файл. Далее возможен поиск по примеру всех тегов по имени (в примере book) с помощью метода getElementByTagName, возвращающего итерируемый тип DOMNodeList.

### 5.2.3. JSON

На первом месте по уровню использования типов файлов в Интернете стоит json. В PHP существует модуль, реализующий работу с текстовым форматом обмена данными - JavaScript Object Notation (JSON). Обработка осуществляется парсером, написанным специально для PHP и под лицензией PHP.

Модуль для обработки данного формата входит в стандартную поставку PHP и содержит минимальную и исчерпывающую функциональность. Рассмотрим пример использования данного модуля в следующем листинге 5.14:

Листинг 5.14. Пример использования модуля json

```
<?php
$json = '{"a":1,"b":2,"c":3,"d":4,"e":5}';

var_dump(json_decode($json));
var_dump(json_decode($json, true));

$arr = json_decode($json);
echo json_encode($arr);

$error = json_last_error();
?>
```

В данном примере значением переменной \$json является строка, содержащая строку в формате json. С помощью функции json\_decode идет декодирование строки в переменную PHP. Флаг \$associative, идущий вторым в параметрах функции указывает на тип возвращаемой переменной. В случае значения по умолчанию null объекты JSON будут возвращены как ассоциативные массивы (array) или объекты (object) в зависимости от того, установлен ли флаг JSON\_OBJECT\_AS\_ARRAY, являющейся JSON-константой, используемой для тонкой настройки кодирования и декодирования формата. В данном случае в первом случае будет возвращен объект, а во втором массив, содержащий соответственно пары ключ-значение.

Во второй части примера происходит обратный процесс из PHP-объекта получаем закодированную строку в формате JSON.

В третьей части примера идет получение сообщения о последней ошибке при кодировании или декодировании в формат json.

### 5.3. Работа с базами данных

Базы данных являются неотъемлемым компонентом любого веб-приложения. Определим драйвер — это специализированное ПО, созданное для взаимодействия с определенным сервером баз данных. Если ваше приложение должно взаимодействовать с базой данных, вы должны написать PHP-код для выполнения таких задач как соединение с базой данных, выполнение запросов и прочих функций. Для предоставления вашему приложению необходимого API и для обеспечения взаимодействия между приложением и базой данных требуется специальное программное обеспечение. Это ПО обычно называют коннектором. И именно оно позволяет вашему приложению соединиться (connect) с базой данных. Иногда люди используют термины коннектор и драйвер, понимая под ними одно и то же. Это неправильно и может привести к путанице. В документации относящейся к MySQL, термин драйвер обозначает ПО, предоставляющее специфичную для конкретного сервера баз данных часть коннектора.

#### 5.3.1. MySQL

Рассмотрим построение взаимодействия с одной из самых популярных СУБД для построения приложений MySQL. Модуль `mysqli`, или как его ещё называют улучшенный (improved) модуль MySQL, был разработан, чтобы дать возможность программистам в полной мере воспользоваться функционалом MySQL-сервера версий 4.1.3 и выше. Модуль `mysqli` включается в поставку PHP версий 5 и выше. Данный модуль обладает следующим продвинутым функционалом:

1. Объектно-ориентированный и процедурный интерфейсы.
2. Поддержка подготавливаемых запросов.
3. Поддержка мультизапросов.
4. Поддержка транзакций.
5. Улучшенные возможности отладки.

Рассмотрим в следующем листинге 5.15 пример подключения к СУБД MySQL с помощью процедурного интерфейса:

Листинг 5.15. Пример подключения к БД в процедурном интерфейсе

```
<?php
$mysqli = mysqli_connect("example.com", "user", "password",
"database");
```

```
$result = mysqli_query($mysqli, "SELECT 'Пожалуйста, не используйте устаревший модуль mysql в новых проектах.' AS _msg FROM DUAL");
$row = mysqli_fetch_assoc($result);
echo $row['_msg'];
```

В данном примере происходит подключение к базе данных database, базирующейся на хосте example.com под пользователем user. В следующей строке отправляется запрос. Из результата запроса с помощью функции `mysqli_fetch_assoc`, извлекается результирующий ряд в виде ассоциативного массива. Возможно также использование объектно-ориентированного интерфейса.

### 5.3.2. MongoDB

Одной из самых используемых и бурно развивающейся СУБД нереляционной БД является MongoDB. Для использования полноценно и удобно драйвера для этой СУБД нужно установить соответствующую библиотеку. После установки возможно простое и эффективное взаимодействие с данной СУБД. В следующем листинге 5.16 представлен простой пример, как вставить документ в коллекцию `beers` базы данных `demo`:

Листинг 5.16. Пример с СУБД MongoDB

```
<?php
require 'vendor/autoload.php'; // подключаем автоподгрузчик классов Composer

$client = new MongoDB\Client("mongodb://localhost:27017");
$collection = $client->demo->beers;

$result = $collection->insertOne( [ 'name' => 'Hinterland',
    'brewery' => 'BrewDog' ] );

echo "Идентификатор вставленного документа '{$result->getInsertedId()}'";
?>
```

В данном примере идет создание клиента подключения к СУБД MongoDB. Во второй строке идет выбор соответствующей коллекции и дальнейшая вставка в эту коллекцию нового документа. Разбор результата вставки документа дает информацию о вставленном документе.

## 5.4. Регулярные выражения

Регулярные выражения (regular expressions или RegEx) - используемый в компьютерных программах, работающих с текстом, формальный язык поиска и осуществления манипуляций с подстроками в тексте, основанный на использо-

вании метасимволов (wildcard characters). Для поиска используется строка-образец (pattern, по-русски её часто называют «шаблоном», «маской»), состоящая из символов и метасимволов и задающая правило поиска. Для манипуляций с текстом дополнительно задается строка замены, которая также может содержать в себе специальные символы.

Регулярные выражения являются инструментальным средством, которое применяется для различных вариантов изучения и обработки текста: поиска, проверки, поиска и замены того или иного элемента, состоящего из букв или цифр (или любых других символов, в том числе специальных символов и символов пунктуации). Изначально регулярные выражения пришли в мир программирования из среды научных исследований, которые проводились в 50-е годы в области математики. Спустя десятилетия принципы и идеи были перенесены в среду операционной системы UNIX (в частности, вошли в утилиту `grep`) и были реализованы в языке программирования Perl, который на заре интернета широко использовался на бэкенде для такой задачи, как, например, валидация форм.

Результатом работы с регулярным выражением может быть:

- 1) проверка наличия искомого образца в заданном тексте;
- 2) определение подстроки текста, которая сопоставляется образцу;
- 3) определение групп символов, соответствующих отдельным частям образца.

Если регулярное выражение используется для замены текста, то результатом работы будет новая текстовая строка, представляющая из себя исходный текст, из которого удалены найденные подстроки (сопоставленные образцу), а вместо них подставлены строки замены (возможно, модифицированные запомненными при разборе группами символов из исходного текста). Частным случаем модификации текста является удаление всех вхождений найденного образца, для чего строка замены указывается пустой.

Большинство символов в регулярном выражении представляют сами себя за исключением специальных символов `[ ] \ / ^ $ . | ? * + ( ) { }` (в разных типах регулярных выражений этот набор различается), которые могут быть экранированы символом `\` (обратная косая черта) для представления самих себя в качестве символов текста. Можно экранировать целую последовательность символов, заключив её между `\Q` и `\E`.

Аналогично могут быть представлены другие специальные символы. Часть символов, которые в той или иной реализации не требуют экранирования (например, угловые скобки `<>`), могут быть экранированы из соображений удо-

бочитаемости.

Метасимвол . (точка) означает один любой символ, но в некоторых реализациях исключая символ новой строки. Вместо символа '.' можно использовать `[\s\S]` (все пробельные и не пробельные символы, включая символ новой строки).

Набор символов в квадратных скобках `[]` именуется символьным классом и позволяет указать, что на данном месте в строке может стоять один из перечисленных символов. В частности, `[абв]` задает возможность появления в тексте одного из трёх указанных символов, а `[1234567890]` задаёт соответствие одной из цифр. Возможно указание диапазонов символов: например, `[А-Яа-я]` соответствует всем буквам русского алфавита, за исключением букв «Ё» и «ё».

Если требуется указать символы, которые не входят в указанный набор, то используют символ `^` внутри квадратных скобок, например `^[0-9]` означает любой символ, кроме цифр.

Некоторые символьные классы можно заменить специальными метасимволами:

Таблица 5.1. Метасимволы

Символ	Возможный эквивалент	Соответствие
<code>\d</code>	<code>[0-9]</code>	Цифровой символ
<code>\D</code>	<code>^[^0-9]</code>	Нецифровой символ
<code>\s</code>	<code>[\f\n\r\t\v]</code>	Пробельный символ
<code>\S</code>	<code>^[^\f\n\r\t\v]</code>	Не пробельный символ  Пример: Выражение вида <code>^\S.*</code> или <code>^[^\f\n\r\t\v].*</code> будет находить строки, начинающиеся с непробельного символа
<code>\w</code>	<code>[A-Za-z0-9_]</code>	Буквенный или цифровой символ или знак подчёркивания; буквы ограничены латиницей  Пример: Выражение вида <code>\w+</code> будет находить и выделять отдельные слова
<code>\W</code>	<code>^[^A-Za-z0-9_]</code>	Любой символ, кроме буквенного или цифрового символа или знака подчеркивания

Следующие символы позволяют позиционировать регулярное выражение относительно элементов текста:

- ^ Начало текста (или строки)
- \$ Конец текста (или строки)

Разберем, как можно в регулярных выражениях задать повторение символов. Квантификатор после символа, символьного класса или группы определяет, сколько раз предшествующее выражение может встречаться. Следует учитывать, что квантификатор может относиться более чем к одному символу в регулярном выражении, только если это символьный класс или группа.

Таблица 5.2. Квантификаторы

*	символы повторяются 0 и до бесконечности
+	повторяются от 1 и до бесконечности
{n}	повторяются точно n раз
{n,}	от n и до бесконечности
{,n}	не более n раз
{n1, n2}	от n1 и до n2 раз точно
?	0 или 1 символ, не больше

Круглые скобки используются для определения области действия и приоритета операций. Шаблон внутри группы обрабатывается как единое целое и, может быть, квантифицирован.

Пример регулярного выражения, которое универсально может находить различные вариации IP адресов.

```
((((25[0-5])|(2[0-4]\d)|(1\d{2})|(\d{1,2})))\.){3}(((25[0-5])|(2[0-4]\d)|(1\d{2})|(\d{1,2}))))
```

Жирным шрифтом выделено то, что будет удовлетворять шаблону

**255.255.255.255** максимальный возможный адрес

**191.198.174.192** какой-то сайт

**127.0.0.1** localhost

Здесь используется логический оператор | (или), который позволяет составить регулярное выражение, которое соответствует правилу, по которому составляются IP адреса. В IP адресе должно быть от 1 и до 3 цифр, в котором число из трех чисел может начинаться с 1, с 2 (или тогда вторая цифра должна



быть в пределах от 0 и до 4), или начинаться с 25, и тогда 3 цифра оказывается в пределах от 0 и до 5. Также между каждой комбинацией цифр должна стоять точка.

В PHP имеются следующие функции для поддержки регулярных выражений:

- 1) `preg_grep()` – выполняет поиск и возвращает массив совпадений;
- 2) `preg_match()` – выполняет поиск первого совпадения с помощью регулярных выражений;
- 3) `preg_match_all()` – выполняет глобальный поиск с помощью регулярных выражений;
- 4) `preg_quote()` – принимает шаблон и возвращает его экранированную версию;
- 5) `preg_replace()` – выполняет операцию поиска и замены;
- 6) `preg_replace_callback()` – тоже выполняет операцию поиска и замены, но используют `callback` – функцию для любой конкретной замены;
- 7) `preg_split()` – разбивает символьную строку на подстроки.

Во всех функциях `preg_replace()`, `preg_replace_callback()` и `preg_split()` поддерживается дополнительный аргумент, который вводит ограничения на максимальное количество замен или разбиений. Пример использования функции `preg_match` представлен в следующем листинге 5.17.

Листинг 5.17. Демонстрация использования функций

```
$pattern_name = '/\w{3,}/';
$pattern_mail = '/\w+@\w+\.\w+/';
$pattern_password = '/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)[a-zA-Z\d]{8,}$/';

if (preg_match($pattern_name, $_POST['$name']) &&
    preg_match($pattern_mail, $_POST['$mail']) &&
    preg_match($pattern_password, $_POST['password'])) {
    # тут происходит, к примеру, регистрация нового пользователя,
    # отправка ему письма, и внесение в базу данных
}
```

В данном примере создается три шаблона на регулярных выражениях для проверки имени (не менее трех символов), адреса электронной почты (слово `@` слово `.` слово) и пароля (состоит из английских букв и цифр, минимум восемь символов).

## 5.5. Валидация данных

Валидация или проверка достоверности модели – это проверка пользовательских данных. Эти данные можно (более того нужно) проверить дважды и

более того при непосредственном вводе их пользователем или перед отправкой формы с данными на сервер (клиентская валидация) и после связывания уже на сервере с помощью специального объекта - валидатора (серверная валидация).

Клиентская валидация — это принятая норма при написании клиентской части интернет-ресурса. Она реализуется с помощью JavaScript. Но её может быть недостаточно, поскольку JavaScript на клиенте может быть отключен по соображению безопасности. Поэтому прежде, чем сохранять полученные данные на сервере в БД их лучше проверить еще раз. Также существует ряд объективных причин, по которым невозможно выполнить валидацию данных на стороне клиента:

1. Для валидации требуется доступ к недоступной части состояния системы. Современные приложения построены с использованием многоуровневой архитектуры, которая предполагает, что реализация пользовательского интерфейса выделена в презентационный слой, а для проверки требуется доступ к другим слоям, вплоть до слоя базы данных.
2. Это особенно актуально для веб-приложений, где пользовательский интерфейс реализуется в браузере и выполняется на стороне клиента, а для проверки ввода требуется сравнение с тем, что хранится в базе данных. В этой ситуации проверку приходится выполнять уже после отправки данных на сервер. Благодаря появлению технологии AJAX эта проблема частично решена.
3. Валидация требует полностью повторить логику обработки. При применении многослойной архитектуры приложения пользовательский интерфейс обычно выделяется в специальный презентационный слой, а логика обработки данных находится на другом слое. И возникает такая ситуация, когда для валидации данных нужно полностью выполнить эту обработку, так как не существует более короткого способа понять, завершится она успехом или нет.

Валидация может выполняться несколькими способами, в зависимости от того, какие ограничения накладываются на данные.

#### *Посимвольная проверка.*

Такие проверки выполняются в пользовательском интерфейсе, по мере ввода данных. Но не только. Например, лексический анализатор компилятора тоже выявляет недопустимые символы непосредственно в процессе чтения компилируемого файла. Поэтому такие проверки можно условно назвать «лексическими».

### *Проверка отдельных значений.*

Для пользовательского интерфейса это проверка значения в отдельном поле, причем выполняться она может как по мере ввода (проверяется то неполное значение, которое введено к настоящему моменту), так и после завершения ввода, когда поле теряет фокус. Для программного интерфейса (API) это проверка одного из параметров, переданных в вызываемую процедуру. Такие проверки условно можно назвать «синтаксическими».

### *Совокупность входных значений.*

Можно предположить, что в программу сначала передаются какие-то данные, после чего подается некоторый сигнал, который инициирует их обработку. Например, пользователь ввёл данные в форму или в несколько форм и наконец нажал кнопку «ОК». В этот момент можно выполнить так называемые «семантические» проверки, нацеленные на валидацию не только отдельных значений, но и взаимосвязей между ними, взаимных ограничений.

Вполне возможна ситуация, когда каждое отдельное значение «синтаксически» корректно, но вместе они образуют не согласованный набор. Для программного интерфейса эта разновидность валидации предполагает проверку набора входных параметров вызываемой процедуры.

### *Проверка состояния системы после обработки данных.*

Это способ, к которому можно прибегнуть, если валидацию непосредственно входных данных выполнить не удастся - можно попытаться их обработать, но оставить возможность вернуть всё к исходному состоянию. Такой механизм часто называется транзакционным.

Транзакция — это последовательность действий, которые либо все завершаются успешно, либо происходит какой-то сбой при выполнении отдельного действия, и тогда отменяются результаты всех предыдущих действий этой цепочки. Валидацию можно выполнять в процессе выполнения транзакции, а последняя проверка может быть выполнена в самом конце транзакции по обработке данных. При этом мы валидируем уже не сами данные, а то состояние, которое получилось после их полной обработки, и если это состояние не удовлетворяет каким-то ограничениям, тогда мы признаем входные данные невалидными и возвращаем всё к исходному состоянию.

Валидацию данных можно и нужно выполнять в несколько этапов, усложняя проверки. Первоначально, по мере ввода, следим за тем, чтобы данные не содержали недопустимых символов. Например, для поля типа даты может быть запрещён ввод нецифровых символов. После того, как ввод завершен, можно проверить все значение целиком. Для введенной даты могут существовать ка-

кие-то ограничения, например, она не должна быть больше или меньше каких-то определённых дат. Когда заполнены все поля, можно проверить, согласованы ли введенные значения друг с другом. Например, если в форме кроме поля для указания даты доставки есть поле для ввода адреса, приложение может проверить, что доставка по данному адресу может быть осуществлена в указанную дату. Если всё введено корректно, можно попытаться начать обработку, выполняя проверки по бизнес-логике приложения, и, если что-то пошло не так, выполнить откат к исходному состоянию.

Разберём на примере, допустим, что в программу передан следующий набор данных, представленный в листинге 5.18:

Листинг 5.18. Пример поступающего набора данных

```
$form = [  
    'name'           => 'Илон Маск',  
    'name_wrong'     => 'Маск',  
    'login'          => 'mask',  
    'login_wrong'    => 'm@sk',  
    'email'          => 'elon@tesla.com',  
    'email_wrong'    => 'elon@tesla_com',  
    'password'       => 'correct_password',  
    'password_wrong' => '123456',  
    'date'           => '2021-10-05 15:52:00',  
    'date_wrong'     => '2021:10:05 15-52-00',  
    'ipv4'           => '192.168.1.1',  
    'ipv4_wrong'     => '402.28.6.12',  
    'uuid'           => '70fcf623-6c4e-453b-826d-  
072c4862d133',  
    'uuid_wrong'     => 'abcd-xyz-6c4e-453b-826d-  
072c4862d133',  
    'extra'          => 'это поле не является предметом вали-  
дации',  
    'empty'          => ''  
];
```

В следующем листинге 5.19 представлен код верификации данных на чистом PHP:

Листинг 5.19. Код верификации на нативном PHP

```
$errors = [];  
  
if (!preg_match('/^[A-Za-zА-Яа-я]+\s[A-Za-zА-Яа-я]+$/u',  
$form['name']))  
    $errors['name'] = 'Имя должно быть из двух слов';  
if (!preg_match('/^[A-Za-zА-Яа-я]+\s[A-Za-zА-Яа-я]+$/u',  
$form['name_wrong']))  
    $errors['name_wrong'] = 'Имя должно быть из двух слов';  
if (!preg_match('/^[a-zA-Z0-9-_]+$/u', $form['login']))  
    $errors['login'] = 'Должно состоять только из цифр и букв  
английского алфавита';
```

```

if (!preg_match('/^[a-zA-Z0-9]_+$/ ', $form['login_wrong']))
    $errors['login_wrong'] = 'Должно состоять только из цифр и
букв английского алфавита';

if (filter_var($form['email'], FILTER_VALIDATE_EMAIL) !=
$form['email'])
    $errors['email'] = 'введен некорректный email';
if (filter_var($form['email_wrong'], FILTER_VALIDATE_EMAIL) !=
$form['email_wrong'])
    $errors['email_wrong'] = 'введен некорректный email';

if (!is_string($form['password']) ||
    $form['password'] == '' ||
    strlen($form['password']) < 8 ||
    strlen($form['password']) > 64
)
    $errors['password'] = 'введен некорректный пароль';

if (!is_string($form['password_wrong']) ||
    $form['password_wrong'] == '' ||
    strlen($form['password_wrong']) < 8 ||
    strlen($form['password_wrong']) > 64
)
    $errors['password_wrong'] = 'введен некорректный пароль';

if (!preg_match('/^(\d{4})-(\d{2})-(\d{2})
(\d{2}):(\d{2}):(\d{2})$/ ', $form['date']))
    $errors['date'] = 'введена некорректная дата';
if (!preg_match('/^(\d{4})-(\d{2})-(\d{2})
(\d{2}):(\d{2}):(\d{2})$/ ', $form['date_wrong']))
    $errors['date_wrong'] = 'введена некорректная дата';

if (filter_var($form['ipv4'], FILTER_VALIDATE_IPV4) !=
$form['ipv4'])
    $errors['ipv4'] = 'введен некорректный ip4 адрес';
if (filter_var($form['ipv4_wrong'], FILTER_VALIDATE_IPV4) !=
$form['ipv4_wrong'])
    $errors['ipv4_wrong'] = 'введен некорректный ip4 адрес';

if (!preg_match('/^[0-9A-F]{8}-[0-9A-F]{4}-4[0-9A-F]{3}-
[89AB][0-9A-F]{3}-[0-9A-F]{12}$/i', $form['uuid']))
    $errors['uuid'] = 'предоставьте корректный uuid!';
if (!preg_match('/^[0-9A-F]{8}-[0-9A-F]{4}-4[0-9A-F]{3}-
[89AB][0-9A-F]{3}-[0-9A-F]{12}$/i', $form['uuid_wrong']))
    $errors['uuid_wrong'] = 'предоставьте корректный uuid!';

if (!isset($form['agreed']) || !is_bool($form['agreed']) ||
$form['agreed'] != true)
    $errors['agreed'] = 'вы должны дать согласие';

return $errors;

```

Рассмотрим проверку на корректность имени: `preg_match('/^[A-Za-zA-Яа-я]+\s[A-Za-zA-Яа-я]+$/'`

Строка должна начинаться (/^) с латинской заглавной или прописной буквы или русской заглавной или прописной буквы ([A-Za-zA-Яа-я]). И так повторяется один или несколько раз (+). Затем один пробел (\s). И повторяем проверку для второго слова, следующей комбинацией [A-Za-zA-Яа-я]+, убеждаясь что оно последнее (\$). Флаг `u` - unicode трактовка. Выражение может содержать специальные паттерны, характерные для юникода.

Затем валидируется email, который должен состоять из английских букв и цифр.

`filter_var($form['email'], FILTER_VALIDATE_EMAIL)` - используя стандартную функцию `filter_var` проверяем содержимое переменной с использованием существующих фильтров, в данном случае используя фильтр `FILTER_VALIDATE_EMAIL` проверяющий, что значение является корректным e-mail. В целом, происходит проверка `addr-spec`-синтаксиса адреса в соответствии с RFC 822.

Проверка валидности пароля заключается лишь в проверке его длины, от 8 до 64 символов.

Проверка корректности даты крайне проста, проверяется только соответствие шаблону, без валидации самой даты и времени.

Валидация адреса IPv4 происходит с помощью встроенного шаблона `FILTER_VALIDATE_IPV4` который проверяет, что значение является корректным IP-адресом только для протоколов IPv4, а также отсутствие вхождения в частные или зарезервированные диапазоны.

Проверка на корректность UUID представляет из себя не просто форму соответствия шаблону, но и требует использования корректных символов в определённых позициях.

И в завершении проверяется что логическая переменная содержит значение `true`.

Приведенный выше код в листинге 5.19 крайне тяжело читаем, приведем пример на каноне чистого ООП на PHP – Symfony в листинге 5.20.

Листинг 5.20. Валидация данных с функциональностью фреймворка Symfony

```
use Symfony\Component\Validator\Constraints\Length;
use Symfony\Component\Validator\Constraints\NotBlank;
use Symfony\Component\Validator\Validation;
use Symfony\Component\Validator\Constraints as Assert;
use Symfony\Component\Translation\MessageSelector;

$validator = Validation::createValidator();
```

```

$constraint = new Assert\Collection([
    'name' => new Assert\Regex('/^[A-Za-z]+\s[A-Za-z]+$/u'),

    'login' => new Assert\Regex('/^[a-zA-Z0-9]_+$/'),
    'email' => new Assert\Email(),
    'password' => [
        new Assert\NotBlank(),
        new Assert\Length(['max' => 64]),
        new Assert\Type(['type' => 'string'])
    ],
    'agreed' => new Assert\Type(['type' => 'boolean'])
]);

$violations = $validator->validate($form, $constraint);

$errors = [];
if (0 !== count($violations)) {
    foreach ($violations as $violation) {
        $errors[] = $violation->getPropertyPath() . ' : ' .
        $violation->getMessage();
    }
}

return $errors;

```

И для сравнения тот же функционал с использованием Rakit Validation приводится в листинге 5.21.

```

$validator = new \Rakit\Validation\Validator;
$validator->addValidator('uuid', new \Validation\RakitRule);

```

Листинг 5.21. Пример валидации данных с помощью модуля Rakit Validation

```

$validation = $validator->make($form, [
    'name' => 'regex:/^[A-Za-z]+\s[A-Za-z]+$/u',
    'name_wrong' => 'regex:/^[A-Za-z]+\s[A-Za-z]+$/u',
    'email' => 'email',
    'email_wrong' => 'email',
    'password' => 'required|min:8|max:64',
    'password_wrong' => 'required|min:8|max:64',
    'login' => 'alpha_dash',
    'login_wrong' => 'alpha_dash',
    'date' => 'date:Y-m-d H:i:s',
    'date_wrong' => 'date:Y-m-d H:i:s',
    'ipv4' => 'ipv4',
    'ipv4_wrong' => 'ipv4',
    'uuid' => 'uuid',
    'uuid_wrong' => 'uuid',
    'agreed' => 'required|accepted'
]);

$validation->setMessages([
    'uuid' => 'UUID should confirm RFC rules!',

```

```

        'required' => ':attribute is required!',
        // etc
    ]);

```

```

$validation->validate();

```

```

return $validation->errors()->toArray();

```

В завершении хотелось бы рассмотреть проверку состояния системы после обработки данных, то есть с помощью механизма транзакций.

Данный механизм применяется, когда нет возможности обработать проверку всей бизнес-логики приложения. Например, часть функционала реализована с помощью триггеров в слое базы данных.

К примеру, мы создаём нового сотрудника, помимо заведения его аккаунта в БД ему создаётся счёт с указанием валюты счёта, и некие страховые случаи. Данная функциональность приведена в листинге 5.22.

Листинг 5.22. Пример проверки состояния системы.

```

try {
    $dbh->setAttribute(PDO::ATTR_ERRMODE,
        PDO::ERRMODE_EXCEPTION);

    $dbh->beginTransaction();
    $dbh->exec("insert into staff (id, first, last) values (23,
        'Василий', 'Пупкин')");
    $dbh->exec("insert into salarychange (id, currency, amount,
        changedate)
        values (23, 2, 50000, NOW())");
    $dbh->exec("select ");
    $dbh->commit();

} catch (Exception $e) {
    $dbh->rollBack();
    echo "Ошибка: " . $e->getMessage();
}

```



## 6. СТАНДАРТНАЯ БИБЛИОТЕКА PHP

### 6.1. Модули и сторонние библиотеки PHP

#### 6.1.1. Установка сторонних модулей на PHP

Язык программирования PHP это один из языков самостоятельной сборки под требуемые задачи, что делает его эффективным при их решении. В стандартной сборке языка не всегда присутствуют нужные библиотеки и требуется их установка. Для разных сборок PHP на разных платформах механизм установки сторонних модулей отличается. Рассмотрим установку сторонних модулей на разных платформах.

##### *Установка модулей на Ubuntu/Debian*

Репозитории дистрибутивов содержат множество уже скомпилированных и готовых к установке модулей, которые просто установить силами системы. В PHP модули являются версия-зависимыми, поэтому с помощью команды `apt-cache search php` можно узнать список доступных готовых модулей из репозитория. Установка производится очень просто с помощью команды `apt install <имя доступного модуля>`. После установки следует проверить факт успешной установки модуля. Для этого в выводе информационной команды `phpinfo()` следует найти путь папки расширений языка php “`extension_dir`”. В данной папке располагаются файлы расширения `.so` обозначающие библиотеки. После установки модуля нужно перезагрузить веб-сервер и проверить наличие информации о модуле в выводе информационной команды `phpinfo()`. Алгоритм на других дистрибутивах отличается только использованием других пакетных менеджеров.

##### *Пакетный менеджер Composer*

Современным решением для управления пакетами для проектов на языке программирования PHP является Composer. Установить данную утилиту можно с помощью туториала на официальном сайте утилиты. Данная утилита также расположена в репозиториях дистрибутивов Unix систем. Нужно понимать, что Composer это более продвинутая система пакетного менеджмента на уровне одного проекта. С помощью данного средства невозможно установить глобальный пакет, только пакет для определенного проекта. Хотя существуют некоторые исключения для модулей анализа и статистики. Модули располагаются в репозитории по адресу <https://packagist.org/>. Там расположен удобный интерфейс для поиска пакетов и документация к ним.

Начало работы с помощью данного пакетного менеджера предполагает создание нового проекта. Для этого предполагается создать папку проекта и в ней выполнить один из предлагаемых сценариев:

1. Создание нового проекта на основе какого-либо модуля. Данный сценарий чаще всего используется при использовании модуля веб-разработки. Формат команды инициализации: `composer create-project <модуль> <папка>`.
2. Создание пустого `composer`-проекта с помощью команды: `composer init`. Данная команда запускает интерактивный режим создания нового проекта, где пользователь задает основную информацию о проекте и `composer` создаст свои основные файлы конфигурации.

После проведения инициализации проекта одним из сценариев мы получаем в первом случае папку `vendor`, содержащую загруженные модули и файлы автозагрузки, и файлы `composer.json`, содержащий основную информацию о проекте, а также все зависимости проекта, и файл `composer.lock`, содержащий текущий список установленных зависимостей и их версии и другую сопутствующую информацию. В итоге при передаче `composer`-проекта все версии зависимостей сохраняются, что позволяет избежать ошибок версионного несоответствия.

Установка пакетов производится с помощью команды `composer require <название пакета>`. Установка предполагает установку самого пакета и добавление его в папку `vendor`, обновление зависимостей и файлов `composer`. Удаление пакета совершается аналогичной командой `composer remove <название пакета>`. Данным способом зависимость будет безопасно удалена. Вручную данную процедуру делать не рекомендуется. Постоянное обновление зависимостей является хорошим тоном для разработки и использования последних стабильных версий пакетов, чтобы это сделать нужно воспользоваться командой `composer update`.

Для того, чтобы использовать модули, установленные с помощью `composer` нужно подключить к скрипту файл `vendor/autoload.php`. В следующем листинге 6.1 представлен простой пример парсинга `html`-страницы с помощью модуля `phpquery`:

Листинг 6.1. Использование модуля `phpquery`, установленного с помощью `composer`

```
<?php
require_once __DIR__ . '/../vendor/autoload.php';
use PhpQuery\PhpQuery;
$page=file_get_contents('sample.html');
$pq=new PhpQuery;
$pq->load_str($page);
//return a list of 3 elements element
var_dump($pq->query('ul li'));
```

В данном примере идет подключение файла автозагрузки, который загру-

зита все необходимое для работы модулей всего проекта. Далее идет использование определенного пространства имен. Все вопросы, связанные с ООП в PHP будут разобраны в следующей главе. И далее простое использование функционала данной библиотеки.

### *Установка модулей на официальные дистрибутивы Docker*

При использовании технологии контейнеризации для разработки веб-приложения на языке программирования PHP с использованием сторонних модулей нужно быть особо внимательным и иметь в виду некоторые факты, связанные с их установкой. Рассмотрим официальный образ PHP. Самые часто используемые модули уже находятся в сборке и их список можно посмотреть с помощью команды `php -m` или `php -i`.

Так как исходный код PHP в образе находится в сжатом виде для уменьшения его веса облегченная установка сторонних модулей производится с помощью специальных скриптов, созданных разработчиком данного официального образа: `docker-php-ext-configure`, `docker-php-ext-install` и `docker-php-ext-enable`. Также чтобы облегчить связывание исходного кода PHP с любым расширением, предоставляется вспомогательный скрипт `docker-php-source`, позволяющий легко извлекать `tar` или удалять извлеченный источник. Так как данные скрипты требуют ручного управления зависимостями, устанавливаемых модулей, которых очень ограниченное количество, создан специальный дополнительный скрипт управления и дополнительной установки недостающих зависимостей `install-php-extensions`, работающий и обладающий функциональностью всех вышеописанных скриптов. Пример использования данного скрипта в `Dockerfile` представлен в листинге 6.2:

Листинг 6.2. Пример использования установочного скрипта `install-php-extensions`

```
FROM php:7.2-cli

ADD https://github.com/mlocati/docker-php-extension-
installer/releases/latest/download/install-php-extensions
/usr/local/bin/

RUN chmod +x /usr/local/bin/install-php-extensions && \
    install-php-extensions gd xdebug
```

С помощью данного скрипта легко устанавливаются модули `gd` и `xdebug`, со всеми необходимыми зависимостями и операциями с ними.

Еще одним официальным методом установки расширений на PHP является использование репозитория PECL, содержащего большой и довольно актуальный список расширений для PHP, реализованных на языке программирования C. Доступ к данному репозиторию предоставляется через устаревший менеджер

расширений PEAR, на смену которому пришел вышеописанный Composer. Для того чтобы использовать данную функциональность нужно вызвать команды `pecl install <имя пакета>-<версия пакета>`. Пример работы с данным репозиторием представлен в листинге 6.3:

Листинг 6.3. Пример использования PECL для установки пакетов

```
FROM php:7.4-cli
RUN pecl install redis-5.1.1 \
    && pecl install xdebug-2.8.1 \
    && docker-php-ext-enable redis xdebug
```

В данном примере идет установка расширений `redis` и `xdebug` из репозитория PECL. Потом идет их включение с помощью скрипта `docker-php-ext-enable`. Если этого не сделать, то модули будут недоступны. Очень важно указывать вручную версии устанавливаемых пакетов и следить вручную с их совместимостью с другими расширениями с рабочей версией PHP.

Еще одним рабочим способом установки расширений на PHP является полностью ручная установка на основе исходника модуля в формате `tar.gz`. Пример такой установки представлен в листинге 6.4:

Листинг 6.4. Установка расширения на основе исходника

```
FROM php:5.6-cli
RUN curl -fsSL
'https://xcache.lighttpd.net/pub/Releases/3.2.0/xcache-
3.2.0.tar.gz' -o xcache.tar.gz \
    && mkdir -p xcache \
    && tar -xf xcache.tar.gz -C xcache --strip-components=1 \
    && rm xcache.tar.gz \
    && ( \
        cd xcache \
        && phpize \
        && ./configure --enable-xcache \
        && make -j "$(nproc)" \
        && make install \
    ) \
    && rm -r xcache \
    && docker-php-ext-enable xcache
```

Для использования продвинутого пакетного менеджера для языка программирования PHP Composer можно воспользоваться официальным образом, содержащим сборку PHP и данный пакетный менеджер. Но данный вариант не всегда предпочтителен, поэтому можно при сборке образа установить `composer` с помощью команды `RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --filename=composer`. Данная команда установит пакетный менеджер глобально и позволит его использовать в проектах.

### *Пакетный менеджер PEAR*

PEAR (PHP Extension and Application Repository) - это репозиторий приложений и модулей PHP. Включает обширный репозиторий PHP-пакетов, систему для управления пакетами (многими разработчиками признана устаревшей, но до сих пор используется), стандарт написания кода и суб-репозитории PECL, App, Gtk. Данная функциональность входит в поставляемую базовую конфигурацию PHP и можно использовать инструмент “из-коробки”. Если по какой-то причине используется сборка без данного пакетного менеджера, то его можно установить из репозитория дистрибутива. Проверить наличие данного пакетного менеджера в системе можно с помощью команды: `pear version`, которая выведет версию пакетного менеджера, версию используемого языка программирования PHP и версию движка PHP. Для данного пакетного менеджера существует набор конфигурационных данных, которые можно получить с помощью команды: `pear config-show`. В данный набор входят: адрес сервера PEAR, путь к директории выполнения, путь к директории с документацией, путь к конфигурационному файлу PEAR и т.д. Данные настройки редко поддаются изменению, но важно понимать их наличие и возможность изменения. Подробное описание каждой директивы конфигурации представлено в официальной документации пакетного менеджера. Для того, чтобы установить какое-либо расширение, нужно воспользоваться командой: `pear install <имя модуля>`. При появлении новой версии пакет можно обновить с помощью команды: `pear upgrade <имя модуля>`.

Попытка установить пакет с требуемыми зависимостями без их наличия дает сообщение об ошибке, что установка не удалась. Более глубокое изучение и фактическое чтение сообщений показывает, что пакету нужны зависимости, которые не установлены в системе. Существует несколько способов выхода из данной проблемы:

1. Установка зависимых пакетов вручную, что исключительно трудоемко и неоправданно чаще всего.
2. Позволить PEAR автоматически устанавливать только необходимые зависимости.
3. Позволить PEAR автоматически устанавливать необходимые и необязательные зависимости.

Использование последних двух способов предполагает использование либо `--onlyreqdeps` (установить только необходимые зависимости), либо `--alldeps` (установить все зависимости) флагов в паре с командой `install`.

Доступ к суб-репозиторию REAR PECL предоставляется с помощью вы-

зова команд `pecl`. Данный репозиторий содержит исходники стандартных пакетов PHP, реализованных на языке программирования C. Для загрузки модуля нужно воспользоваться командой: `pecl install <имя пакета>`. Также предоставляется функциональность для компиляции модулей, соответствующих парадигме PECL.

### 6.1.2. Обзор сторонних библиотек PHP

#### *PHP Benchmark*

PHP Benchmark является удобным инструментом получения статистики приложения и ее визуализации. Можно получать не только конечные данные, но и собирать обширную промежуточную статистику. Данную библиотеку можно установить с помощью пакетного менеджера `composer`. Важно помнить, что данное расширение использует библиотеку `bcmath`, которая не всегда входит в базовую сборку PHP (особенно это актуально при использовании контейнеризации). Пример использования данного сервиса представлен в листинге 6.5:

*Листинг 6.5. Пример использования PHP Benchmark*

```
<?php

require __DIR__.'../../vendor/autoload.php';

use PHPBenchmark\testing\FunctionComparison;

FunctionComparison::load()
    ->addFunction('Short sleep', function() {
        usleep(1);
    })
    ->addFunction('Longer sleep', function() {
        usleep(10);
    })
    ->addFunction('Even longer sleep and allocating memory',
function() {
    static $str='';
    usleep(100);
    $str .= str_repeat((string)mt_rand(), 10);
    })
    ->exec();
```

В примере, представленном в листинге 6.5 идет сопоставление нескольких функций по 3 параметрам: время выполнения, использование памяти, сравнение скорости выполнения относительно остальных. Данной функциональностью данное расширение не ограничивается. Результатом выполнения будет следующая табл. 6.1:

Таблица 6.1. Результат выполнения примера из листинга 6.5

Function description	Execution time (s)	Memory usage (mb)	Time faster
Longer sleep	0.052600	0.001000	16%
Short sleep	0.061300	0.001000	102%
Even longer and allocating memory	0.124200	0.047000	0

### *RegExp Builder*

RegExp Builder - полноценный аналог одноименной библиотеки для языка программирования JavaScript. RegExp Builder интегрирует регулярные выражения в язык программирования, тем самым облегчая их чтение и обслуживание. Регулярные выражения создаются с помощью цепных методов и переменных, таких как массивы или строки. Установить данный модуль можно также с помощью composer или другим удобным способом, описанным в подробной документации. Пример валидации имен файлов представлен в следующем листинге 6.6:

Листинг 6.6. Пример валидации данных с помощью RegExpBuilder

```
<?php
require __DIR__.'../vendor/autoload.php';
use Gherkins\RegExpBuilderPHP\RegExpBuilder;
$builder = new RegExpBuilder();
$regExp = $builder
    ->startOfInput()
    ->exactly(4)->digits()
    ->then("_")
    ->exactly(2)->digits()
    ->then("_")
    ->min(3)->max(10)->letters()
    ->then(".")
    ->anyOf(array("png", "jpg", "gif"))
    ->endOfInput()
    ->getRegExp();

//true
var_dump($regExp->matches("2020_10_hund.jpg"));
var_dump($regExp->matches("2030_11_katze.png"));
var_dump($regExp->matches("4000_99_maus.gif"));

//false
var_dump($regExp->matches("123_00_nein.gif"));
var_dump($regExp->matches("4000_0_nein.pdf"));
var_dump($regExp->matches("201505_nein.jpg"));
```

В данном примере идет легкое нативное построение регулярное выражения: точно 4 цифры в начале, потом нижнее подчеркивание, потом точно 2

цифры, снова нижнее подчеркивание, далее от 3 до 10 букв, потом точка и любой вариант расширения файла. Далее идет валидация различных имен файлов, соответствуют ли они заданному регулярному выражению.

### *phpQuery*

Существует множество реализаций данного расширения. В данном обзоре описывается расширение, которое устанавливается пакетным менеджером Composer. Это простая оболочка библиотеки xpath, которая позволяет выполнять запросы на html-странице с более удобным синтаксисом jquery. Внутри использует Document Object Model. Пример запросов представлен в следующем листинге 6.7:

Листинг 6.7. Пример использования PhpQuery

```
<?php

require_once __DIR__ . '/../vendor/autoload.php';
use PhpQuery\PhpQuery;

$page=file_get_contents('sample.html');
$pq=new PhpQuery;
$pq->load_str($page);

//return a list of 2 components
var_dump($pq->query('.cl'));

//return the first element
var_dump($pq->query('.cl')[0]);
```

На вход обработчику PhpQuery подается HTML-страница в виде строки, обработчику посылаются запросы на основе синтаксиса jQuery, ответ приходит в виде объектов типа DOM. В случае примера DOMNodeList.

### *mPDF*

mPDF – это библиотека PHP, которая генерирует PDF-файлы из HTML-кода в кодировке UTF-8. Она основана на инструментах FPDF и HTML2FPDF с рядом улучшений, позволяющих данной библиотеке являться одним из лучших инструментов генерации PDF-файлов для языка программирования PHP. В отличие от своих конкурентов, mPDF в полной мере поддерживает русский язык, вставку картинок, форматирование и, самое главное, HTML и CSS. Установка данного расширения также возможна с помощью пакетного менеджера composer. Пример создания простого PDF-документа представлен в листинге 6.8:

Листинг 6.8. Пример использования mPDF

```
<?php
require_once __DIR__ . '/../vendor/autoload.php';
```



```
$mpdf = new \mPDF();
$mpdf->WriteHTML('<h1>Hello world!</h1>');
$mpdf->Output();
```

В данном примере результатом будет являться файл, открытый браузером, как PDF. Данная функциональность открывает уникальные возможности генерации быстрой генерации документов и выдачи их пользователю.

Следует уточнить, что использование данного пакета предполагает точный контроль версий, тк названия классов в разных версиях могут быть одинаковыми по названию, но разными по регистру.

### *Goutte*

Goutte – это библиотека для использования экрана и обхода веб-страниц для PHP. Goutte предоставляет удобный API для обхода веб-сайтов и извлечения данных из ответов HTML/XML. Лицензия - MIT. Установка данного расширения возможна с помощью пакетного менеджера Composer. Следует отметить, что данное расширение требует версию языка программирования PHP 7.1+. Также данный модуль использует возможности и технологии фреймворка веб-разработки для языка программирования PHP Symfony. В следующем листинге 6.9 представлен пример авторизации на GitHub с помощью данной библиотеки:

Листинг 6.9. Пример использования библиотеки Goutte

```
<?php
require_once __DIR__ . '/../vendor/autoload.php';
use Goutte\Client;

$client = new Client();
$crawler = $client->request('GET', 'https://github.com/');
$crawler = $client->click($crawler->selectLink('Sign in')->link());
$form = $crawler->selectButton('Sign in')->form();
$crawler = $client->submit($form, ['login' => 'fabpot', 'password' => 'xxxxxx']);
$crawler->filter('.flash-error')->each(function ($node) {
    print $node->text()."\n";
});
```

В данном примере происходит переход по ссылкам, кликание кнопок, заполнение формы авторизации, а также извлечение данных из страницы.

### *GoogChart*

GoogChart является классом PHP для создания динамичных диаграмм с помощью Google Charts. Не распространяется на весь API Google Charts, но делает их очень простыми в использовании. Установить данный модуль можно скачав архив с данным классом с официального репозитория code google. Рас-

пространяется по лицензии MIT. Данное расширение немного устарело, но это стабильно и позволяет использовать простую функциональность Google для построения графиков. Пример построения круговой диаграммы с помощью данного модуля представлен в листинге 6.10:

Листинг 6.10. Пример построения круговой диаграммы с помощью GoogChart

```
<?php
include 'GoogChart.class.php';
$chart = new GoogChart();
$data = array(
    'IE7' => 22,
    'IE6' => 30.7,
    'IE5' => 1.7,
    'Firefox' => 36.5,
    'Mozilla' => 1.1,
    'Safari' => 2,
    'Opera' => 1.4,);
$color = array(
    '#99C754',
    '#54C7C5',
    '#999999',
);
$chart->setChartAttrs( array(
    'type' => 'pie',
    'title' => 'Browser market 2008',
    'data' => $data,
    'size' => array( 400, 300 ),
    'color' => $color
));
echo $chart;
```

В данном примере мы создаем новый экземпляр класса GoogChart. Далее создаем ассоциативный массив данных и список используемых цветов. Потом задаем требуемые атрибуты графика: тип, заголовок, данные, размеры и цвета, — и выводим график на странице. Полученный график представлен на рис. 6.1. Данный класс также создавать другие различные типы графиков с более сложным набором данных.

**Pie chart**

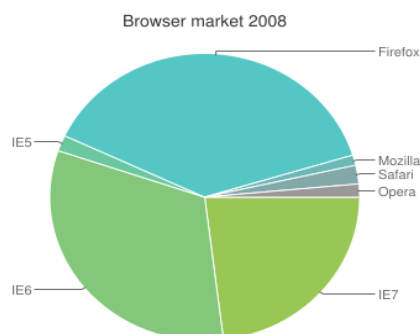
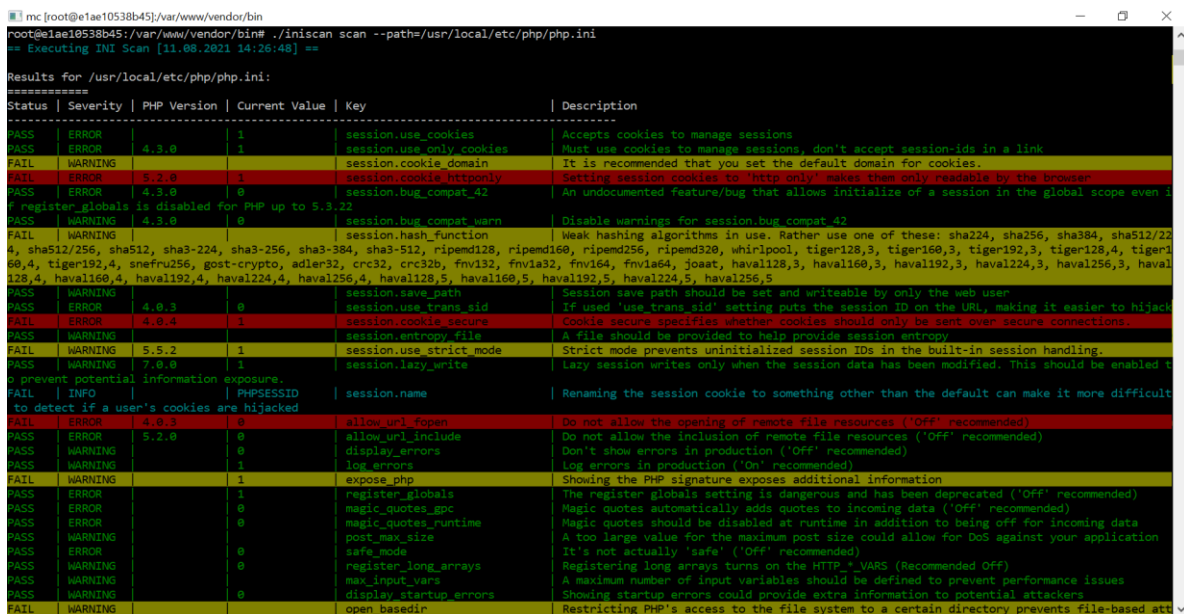


Рисунок 6.1. График, построенный с помощью скрипта из листинга 6.10

## IniScan

IniScan — это инструмент, предназначенный для сканирования конфигурационного файла `php.ini` на предмет общих методов обеспечения безопасности и выдачи отчета о результатах сканирования. В настоящее время он предназначен только для использования в командной строке и выводит данные как прохождения, так и провала каждого теста. Единственной зависимостью является функционал командной строки библиотеки веб-разработки `Symfony`. Установить данное расширение с помощью пакетного менеджера `Composer`: локально для только своего проекта и с глобальной функциональностью (такую возможность предоставляет `Composer`). Для проверки `php.ini` файла воспользуемся командой из корня проекта - `vendor/bin/iniscan scan --path=/path/to/php.ini`. Результат выполнения команды представлен на рис. 6.2:



Status	Severity	PHP Version	Current Value	Key	Description
PASS	ERROR		1	session.use_cookies	Accepts cookies to manage sessions
PASS	ERROR	4.3.0	1	session.use_only_cookies	Must use cookies to manage sessions, don't accept session-ids in a link
FAIL	WARNING			session.cookie_domain	It is recommended that you set the default domain for cookies.
FAIL	ERROR	5.2.0	1	session.cookie_httponly	Setting session cookies to 'http only' makes them only readable by the browser
PASS	ERROR	4.3.0	0	session.bug_compat_42	An undocumented feature/bug that allows initialize of a session in the global scope even if register_globals is disabled for PHP up to 5.3.22
PASS	WARNING	4.3.0	0	session.bug_compat_warn	Disable warnings for session.bug_compat_42
FAIL	WARNING			session.hash_function	Weak hashing algorithms in use. Rather use one of these: sha224, sha256, sha384, sha512/224, sha512/256, sha512, sha3-224, sha3-256, sha3-384, sha3-512, ripemd128, ripemd160, ripemd256, ripemd320, whirlpool, tiger128,3, tiger160,3, tiger192,3, tiger128,4, tiger160,4, tiger192,4, snet-crypto, adler32, crc32, crc32b, fnv132, fnv132, fnv164, fnv164, joaat, haval128,3, haval160,3, haval192,3, haval128,4, haval160,4, haval192,4, haval224,4, haval256,4, haval128,5, haval160,5, haval192,5, haval224,5, haval256,5
PASS	WARNING			session.save_path	Session save path should be set and writable by only the web user
PASS	ERROR	4.0.3	0	session.use_trans_sid	If used 'use_trans_sid' setting puts the session ID on the URL, making it easier to hijack
FAIL	ERROR	4.0.4	1	session.cookie_secure	Cookie secure specifies whether cookies should only be sent over secure connections.
PASS	WARNING			session.entropy_file	A file should be provided to help provide session entropy
FAIL	WARNING	5.5.2	1	session.use_strict_mode	Strict mode prevents uninitialized session IDs in the built-in session handling.
PASS	WARNING	7.0.0	1	session.lazy_write	Lazy session writes only when the session data has been modified. This should be enabled to prevent potential information exposure.
FAIL	INFO			session.name	Renaming the session cookie to something other than the default can make it more difficult to detect if a user's cookies are hijacked
FAIL	ERROR	4.0.3	0	allow_url_fopen	Do not allow the opening of remote file resources ('Off' recommended)
PASS	ERROR	5.2.0	0	allow_url_include	Do not allow the inclusion of remote file resources ('Off' recommended)
PASS	WARNING		0	display_errors	Don't show errors in production ('Off' recommended)
PASS	WARNING		1	log_errors	Log errors in production ('On' recommended)
FAIL	WARNING		1	expose_php	Showing the PHP signature exposes additional information
PASS	ERROR		1	register_globals	The register_globals setting is dangerous and has been deprecated ('Off' recommended)
PASS	ERROR		0	magic_quotes_gpc	Magic quotes automatically adds quotes to incoming data ('Off' recommended)
PASS	ERROR		0	magic_quotes_runtime	Magic quotes should be disabled at runtime in addition to being off for incoming data
PASS	WARNING		0	post_max_size	A too large value for the maximum post size could allow for DoS against your application
PASS	ERROR		0	safe_mode	It's not actually 'safe' ('Off' recommended)
PASS	WARNING		0	register_long_arrays	Registering long arrays turns on the HTTP_*_VARS (Recommended Off)
PASS	WARNING		0	max_input_vars	A maximum number of input variables should be defined to prevent performance issues
PASS	WARNING		0	display_startup_errors	Showing startup errors could provide extra information to potential attackers
FAIL	WARNING			open_basedir	Restricting PHP's access to the file system to a certain directory prevents file-based att

Рисунок 6.2. Результат проверки конфигурационного файла утилитой *iniscan*

Данный конфигурационный файл является дефолтным для официального образа `php-apache`. Как видно из результата конфигурационный файл нуждается в исправлениях для безопасности, что выясняется из исчерпывающего описания отчета по сканированию `php.ini`. Обращаясь к отчету, можно заметить, что тесты, связанные с включением технологии `cookies` успешно пройдены. А вот тест, связанный с защитной директивой `cookies`, запрещающий всем, кроме браузера видеть их, не пройден. Данный тест чрезвычайно важен, но в разрабатываемой системе данное требование присутствует, поэтому для системы это не ошибка и мы игнорируем данное замечание. А вот замечание на использование небезопасного алгоритма хэширования для генерации идентификатора сессии и предложение использовать более продвинутый и безопасный следует принять к сведению. Также предлагается выключить возможности директивы

`allow_url_fopen`. Данная директива включает поддержку оберток URL (URL wrappers), которые позволяют работать с объектами URL как с обычными файлами. Обёртки, доступные по умолчанию, служат для работы с удалёнными файлами с использованием ftp или http протокола. Это не совсем безопасная функция и ее включение рекомендуется только при использовании данной функциональности.

Возможна настройка формата вывода и наполнения вывода для удобства исследования конфигурации php с помощью инструментария модуля Iniscan.

### *HTML Purifier*

HTML Purifier – библиотека для фильтрации HTML, разработанная для удаления всего вредоносного кода. Может быть использована как мощный инструмент для защиты кода от XSS атак. Эта библиотека использует надежные белые списки, агрессивный разбор, и убеждается, что в результате разметка соответствует стандартам. Лицензия LGPL v2.1+. Установка возможна с помощью пакетных менеджеров PEAR и Composer и другими надежными способами, описанными в официальной документации. Возможна модернизация функциональности на основе возможностей данного модуля. Простой пример использования и верификации и чистки HTML-страницы из листинга 6.9 представлен в листинге 6.10.

#### Листинг 6.10. Пример HTML-страницы для фильтрации

–Исходный вариант–

```
<p invalidAttribute="value">О, я хочу безумно  
<strike>жить</strike>:</p>  
    <p>Всё сущее – <invalid-  
Tag>увековечить</invalidTag>,</p>  
    <p class="header error">Безличное – вочеловечить,</p>  
    Несбывшееся – воплотить!  
    <script type="text/javascript">alert("hacked by Alex-  
ander Blok");</script>
```

–Модернизированный вариант–

```
<p>О, я хочу безумно <span style="text-decoration:line-  
through;">жить</span>:</p>  
    <p>Всё сущее – увековечить,</p>  
    <p class="header">Безличное – вочеловечить,</p>  
    <p>Несбывшееся – воплотить!</p>
```

В примере HTML-страницы используется недопустимый код, теги и т.д.

#### Листинг 6.11. Пример использования инструментов для фильтрации в HTML

```
<?php  
$config = HTMLPurifier_Config::createDefault();  
$config->set('Attr.AllowedClasses',array('header'));  
$config->set('AutoFormat.AutoParagraph',true);  
$config->set('AutoFormat.RemoveEmpty',true);
```

```
$config->set('HTML.Doctype','HTML 4.01 Strict');
$purifier = new HTMLPurifier($config);
$clean_html = $purifier->purify($html);
```

В примере для начала создается конфигурация с правилами фильтрации: допустимые CSS-классы, автоформатирование - добавление нового образа при обнаружении новой строки, удаление пустых тегов, тип документа(всего доступны: XHTML 1.0 Transitional (default), XHTML 1.0 Strict, HTML 4.01 Transitional, HTML 4.01 Strict, XHTML 1.1). После создается новый Очиститель с созданной конфигурацией и производится очистка.

### *JpGraph*

JpGraph – PHP библиотека для создания объектно-ориентированных графиков. Имеет более 200 встроенных флагов стран, 400 именованных цветов и поддерживает дополнительно диаграммы Ганта, несколько Y-осей, альфа-смешивание и внутреннее кэширование (с таймаутом). Эта библиотека позволяет строить линейные диаграммы, гистограммы, пироги, карты, свечи, полярные, радар, круговые и контурные диаграммы любого размера. Является одним из мощнейших инструментов построения графиков для языка программирования PHP. Существует 2 версии данного инструментария: с свободной лицензией для некоммерческого использования и версия для коммерческого использования с дополнительным расширенным функционалом. Существует вариант установки данной библиотеки с официального сайта на основе файла исходника и интеграции ручной в проект, другого официального варианта получить данную библиотеку нет. Данный способ рекомендуется, но существует friendly port данного инструментария в репозитории composer со свободно распространяемой лицензией. В следующем примере будет использован именно этот вариант библиотеки, который практически полностью совпадает функционалом с комьюнити версией официальной библиотеки. Для работы функционала библиотеки обязательно нужны библиотеки gd и libpng-dev, чаще всего они входят в стандартную сборку, но в случае использования официальных докер образов php это будет скорее всего не так. В следующем листинге 6.11 представлен пример построения круговой диаграммы, результат построения диаграммы представлен на рисунке 6.3:

Листинг 6.11. Пример создания простой круговой диаграммы с помощью JpGraph

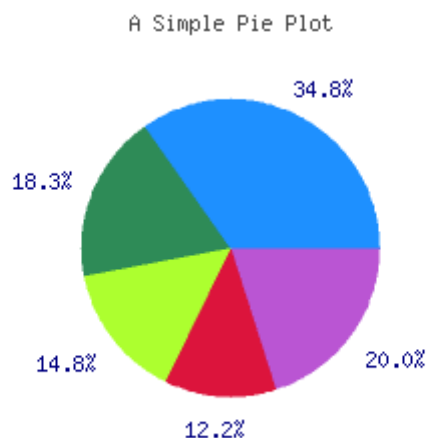
```
<?php
require __DIR__.'../vendor/autoload.php';

use Amenadiel\JpGraph\Graph;
use Amenadiel\JpGraph\Plot;
```

```
// Create the Pie Graph.
$graph = new Graph\PieGraph(350, 250);
$graph->title->Set("A Simple Pie Plot");
$graph->SetBox(true);

$data = array(40, 21, 17, 14, 23);
$p1 = new Plot\PiePlot($data);
$p1->ShowBorder();
$p1->SetColor('black');
$p1->SetSliceColors(array('#1E90FF', '#2E8B57', '#ADFF2F',
'#DC143C', '#BA55D3'));

$graph->Add($p1);
$graph->Stroke();
```



*Рисунок 6.3. График, полученный с помощью функционала библиотеки JpGraph*

### *Upload*

Upload – библиотека, которая облегчает загрузку и валидацию файлов. Когда форма отправлена, библиотека может проверить тип файла и его размер, что облегчает работу с функциональностью загрузки файлов на сервер и их дальнейшей обработки. Распространяется по лицензии MIT. Установка данного пакета возможна с помощью пакетного менеджера composer. Загрузка файлов чаще всего производится с помощью HTML-формы, представленной в следующем листинге 6.12:

Листинг 6.12. Пример формы для загрузки файлов

```
<form method="POST" enctype="multipart/form-data">
  <input type="file" name="foo" value=""/>
  <input type="submit" value="Upload File"/>
</form>
```

На серверной стороне производится обработка запроса данной формы. Код обработчика представлен в следующем листинге 6.13:

### Листинг 6.13. Обработка загруженного файла с помощью расширения Upload

```
<?php
require __DIR__.'../vendor/autoload.php';
use Upload\Storage;
use Upload;
use Upload\Validation;

$storage = new FileSystem('/path/to/directory');
$file = new File('foo', $storage);

$new_filename = uniqid();
$file->setName($new_filename);

$file->addValidations(array(
    new Mimetype('image/png'),
    new \Upload\Validation\Size('5M')
));

$data = array(
    'name'      => $file->getNameWithExtension(),
    'extension' => $file->getExtension(),
    'mime'      => $file->getMimetype(),
    'size'      => $file->getSize(),
    'md5'       => $file->getMd5(),
    'dimensions' => $file->getDimensions()
);

try {
    $file->upload();
} catch (\Exception $e) {
    $errors = $file->getErrors();
}
```

В примере создается класс управления директорией хранения файлов, получаемых от пользователя, а также экземпляр класса для обрабатываемого файла. Потом идет изменение имени обрабатываемого файла, для того чтобы не было конфликтов с уже существующими файлами или недопустимых имен или недопустимых символов в имени файла во избежание ошибок. Далее идет создание данных для валидации: задание возможных типов и допустимого размера файла. Потом из данных файла возможна выгрузка различных статистических данных о файле. Последним действием является попытка сохранения файла в файловую систему.

#### *Respect Validation*

Respect Validation – это самый потрясающий механизм валидации, когда-либо созданный для PHP. Таковым он признан даже самыми рядовыми разработчиками, которые используют минимум его функционала. Пример использования данного пакета был приведен в предыдущей главе. Предоставляется бо-

лее 150 возможных встроенных правил и контроль детализации для расширенной отчетности. Распространяется как свободное программное обеспечение под лицензией MIT. Установка возможна с помощью пакетного менеджера Composer.

Для начала валидации необходимо получить объект класса Validator для выбранного правила валидации. Правила валидации определены в специальных классах библиотеки и определяют, по какому принципу будет проверяться некоторое значение. Таких классов очень много и есть что выбрать даже для какого-либо редкого типа валидации. Пример использования данной функциональности представлен в следующем листинге 6.14:

Листинг 6.14. Пример валидации данных с помощью Respect Validation

```
<?php
require __DIR__.'../vendor/autoload.php';
use Respect\Validation\Validator as v;

$number = 123;
v::numericVal()->validate($number); //true

$usernameValidator = v::alnum()->noWhitespace()->length(1,
15);
$usernameValidator->validate('alganet'); // true

$user = new stdClass;
$user->name = 'Alexandre';
$user->birthdate = '1987-07-01';
$userValidator = v::attribute('name', v::stringType()-
>length(1, 32))
    ->attribute('birthdate', v::date()->minAge(18));

$userValidator->validate($user); // true
```

В представленном примере первым идет создание правила для валидации чисел и производится валидация простого целого числа. Это тривиальный пример создания простого односложного правила. Вторым в примере идет создание многосложного правила, состоящего из следующих правил: строка должна состоять из чисел и букв, в строке не должно быть пробельных символов, и длина строки должна быть от 1 до 15 символов. И далее идет валидация строки с помощью метода `validate`. Третьим в примере идет валидация пользовательского класса, содержащего поля имя и дата рождения. Создается валидатор, в котором с помощью метода `attribute()` описываются правила валидации для каждого свойства класса: для имени это должна быть строка длиной от 1 до 32 символов и для даты рождения это должна быть дата и минимально допустимый возраст это 18 лет. Как видно за счет задания правил и их тривиального



использования валидация данных проходит быстро и безболезненно для разработчика.

### *Faker*

Существует потребность генерации фейковых данных для тестирования приложений или других ситуаций, где требуется набор адекватных данных. Независимо от того, нужно ли заполнять базу данных, создавать красивые XML-документы, заполнять данные для стресс-тестирования или анонимизировать данные, взятые из производственной службы, *Faker* поможет автоматизировать генерацию данных под эти задачи. Библиотека создавалась под влиянием функционала одноименных библиотек для Perl и Ruby. Распространяется как открытое программное обеспечение по лицензии MIT. Установка данного расширения возможна с помощью пакетного менеджера *Composer*. В следующем листинге 6.15 приведен простой пример использования данного модуля для генерации фейковых данных:

Листинг 6.15. Пример генерации данных с помощью библиотеки *Faker*

```
<?php
require __DIR__.'../../vendor/autoload.php';

$faker = Faker\Factory::create();
echo $faker->name().'<br>';
echo $faker->email().'<br>';
echo $faker->text().'<br>';
echo $faker->address();
```

В данном примере создается фабрика и с помощью ее генерируются следующие данные: имя, адрес электронной почты, текст и адрес. Все методы генерации данных в *Faker* созданы на основе реализации провайдеров. По умолчанию используются провайдеры, представленные в следующем листинге 6.16:

Листинг 6.16. Провайдеры по умолчанию фабрики *Faker*

```
<?php
$faker = new Faker\Generator();
$faker->addProvider(new Faker\Provider\en_US\Person($faker));
$faker->addProvider(new Faker\Provider\en_US\Address($faker));
$faker->addProvider(new Faker\Provider\en_US\PhoneNumber($faker));
$faker->addProvider(new Faker\Provider\en_US\Company($faker));
$faker->addProvider(new Faker\Provider\Lorem($faker));
$faker->addProvider(new Faker\Provider\Internet($faker));
```

Данные провайдеры актуальны для Соединенных Штатов, существуют провайдеры для всех основных стран, в том числе и для России. Задав российского провайдера вместо вывода по умолчанию, представленного в 1 части листинга 6.17, получим вывод, представленный во 2 части листинга 6.17. Задание

полностью всех российских провайдеров можно сделать с помощью параметра “Ru-Ru” конструктора фабрики генератора. Возможно создание пользовательских провайдеров для расширения возможностей данной библиотеки.

Листинг 6.17. Модифицирование вывода

-Первая часть-

Estrella McKenzie

abby85@feeney.org

Qui quas quo impedit eaque enim voluptas laborum. Vel vel  
possimus qui doloremque iusto sed. Nulla veniam aut et eaque  
cupiditate.

95659 Verona Dam Suite 498 Lake Edmundtown, NJ 01285

-Вторая часть-

Николай Александрович Некрасов

nika.rozkov@sazonov.ru

Quidem quod at placeat sequi molestiae possimus voluptatem.  
Iste est aut cupiditate laboriosam assumenda quidem. Dolore  
culpa blanditiis ut recusandae. Qui blanditiis qui tempore et  
et.

378057, Псковская область, город Сергиев Посад, пр. Гоголя, 69

## 6.2. Стандартная библиотека РНР

### 6.2.1. Время и дата

Управление временем и датой в веб-приложении важно для использования данной функциональности. Дата и время не заканчиваются на уровне ввода пользователем даты рождения при регистрации на каком-либо веб-ресурсе. Это календарь с заметками и дедлайнами дел в соответствующем приложении, получение специального предложения и скидок на день-рождения пользователя в интернете-магазине и т.д. Когда пользователями веб-приложения являются люди с разных концов планеты требуется грамотное управление таким типом данных как время, чтобы не было казусов и проблем.

#### *Календарь*

Модуль календарь предоставляет ряд функций для упрощения перевода дат из одного формата в другой. Промежуточной датой или стандартом является исчисление с Юлианского дня. Исчисление с Юлианского дня – это исчисление, начинающееся с 1 января 4713г. до н.э. Для конвертации между календарными форматами сначала нужно перевести дату в Юлианское исчисление, а затем в любой другой календарный формат на ваш выбор. Данное расширение чаще всего входит в базовую поставку РНР, кроме ситуаций если это официальный Docker-образ РНР, где его нужно установить с помощью скрипта `docker-php-ext-install`, или РНР скомпилирован без поддержки календаря. Директив настройки в файле конфигурации данное расширение не имеет.

Рассмотрим основную функциональность данного исключительно важного модуля в листинге 6.18. Следует упомянуть основные поддерживаемые календари, определенные в виде предопределенных констант: `CAL_GREGORIAN=0`, `CAL_JULIAN=1`, `CAL_JEWISH=2` и `CAL_FRENCH=3`.

Листинг 6.18. Использование функциональности модуля `Calendar`

```
<?php
$info = cal_info(0);
print_r($info);echo '<br>';
$newdate = cal_to_jd(CAL_GREGORIAN, 1, 1, 2000);
print_r($newdate);echo '<br>';
print_r(date("M-d-Y", easter_date(2000)));echo '<br>';
print_r(gregoriantojd(1,1,2000));echo '<br>';
print_r(jddayofweek(gregoriantojd(1,1,2000), 1));echo '<br>';
print_r(jdtounix(gregoriantojd(1,1,2000)));echo '<br>';
```

В следующем примере представлены основные функции модуля `Calendar`. Функция `cal_info` отображает основную информацию о календаре по его коду или предопределенной константе. Полезно при преобразовании дат для получения информации о лично мало используемом календаре. Формат вывода представлен в листинге 6.19:

Листинг 6.19. Вывод `cal_info`

```
Array ( [months] => Array ( [1] => January [2] => February [3]
=> March [4] => April [5] => May [6] => June [7] => July [8]
=> August [9] => September [10] => October [11] => November
[12] => December ) [abbrevmonths] => Array ( [1] => Jan [2] =>
Feb [3] => Mar [4] => Apr [5] => May [6] => Jun [7] => Jul [8]
=> Aug [9] => Sep [10] => Oct [11] => Nov [12] => Dec )
[maxdaysinmonth] => 31 [calname] => Gregorian [calsymbol] =>
CAL_GREGORIAN )
```

Обращаясь к листингу 6.19, можно заметить, что получаемая информация содержит информацию о количестве месяцев, их полных и кратких названиях, максимальном количестве дней в месяце, название календаря и обозначение константы календаря.

Обратимся снова к листингу 6.18. Преобразование любых дат идет по следующему алгоритму: один календарь <-> юлианское исчисление <-> второй календарь. Универсальной функцией для преобразования из любого календаря в Юлианское исчисление является функция `cal_to_jd`, которая принимает на вход календарь, из которого выполняется преобразование, и дату в разделенных значениях: день, месяц, год. Особой идеологической функциональностью являются функции определения даты и месяца Пасхи, например с помощью функции `easter_date`, которая получает на вход год и возвращает дату в юлианском исчислении. Также существует набор функций для преобразования из определен-

ного календаря в юлианское счисление и обратно, например, в листинге 6.18 представлено преобразование из григорианского календаря в юлианское исчисление. Также существует функциональность получения дня недели на основе даты в юлианском счислении: в формате номера дня недели, полного названия дня недели и сокращенного наименования дня недели, - с помощью функции `jddayofweek`. Также доступны преобразования в счисление Unix систем из и в юлианское счисление, например, с помощью функции `jdtounix`.

### *Время и дата*

Модуль `time` и `date` предоставляет набор необходимых базовых функций. Эти функции позволяют получить дату и время с сервера, на котором запущен скрипт PHP. Их можно использовать для форматирования даты и времени различными способами. Информация о дате и времени хранится в памяти в виде 64-разрядных чисел. Таким образом, поддерживаются все пригодные представления даты (включая отрицательные года). Диапазон составляет примерно 292 миллиарда лет в прошлое и будущее. Убедительная просьба принимать во внимание, что эти функции зависят от региональных настроек сервера. Поэтому, если сервер располагается на Багамах, а пользователи в Москве нужно учитывать часовой пояс. Данный модуль в любом случае является частью ядра PHP и его устанавливать не нужно. Максимум, что можно сделать – это установить последнюю актуальную базу часовых поясов с помощью команды `pecl install timezonedb`. Последнее обновление на момент написания материала выполнено в июне 2021 года.

В конфигурационном файле `php.ini` возможна настройка данного модуля на основе следующего списка директив:

- 1) `date.default_latitude` - Широта по умолчанию. в диапазоне от 0 на экваторе до +90 к северу и -90 к югу;
- 2) `date.default_longitude` - Долгота по умолчанию. в диапазоне от 0 на нулевом меридиане до +180 на восток и -180 на запад;
- 3) `date.sunrise_zenith` - Угол, под которым солнце светит во время восхода;
- 4) `date.sunset_zenith` - Угол, под которым солнце светит во время заката;
- 5) `date.timezone` - Часовой пояс, используемый по умолчанию всеми функциями даты/времени.

Можно сделать вывод о том, что доступен обширнейший функционал настроек для функций даты и времени на уровне ядра.

Класс `DateTimeImmutable` является неизменяемым представлением даты и времени. Существует изменяемый аналог класс `DateTime`. Создание экземпляра

данного класса возможно в процедурном (используется функция `date_create_immutable`) и объектно-ориентированном стиле. По умолчанию создается объект, содержащий текущее время и в текущем часовом поясе, но данные моменты настраиваются. Также возможно создание нового объекта типа `DateTimeImmutable` на основе разбора даты некоторого формата с помощью функциональности `date_create_immutable_from_format`. Важной особенностью данного класса является метод `modify`, который изменяет временную метку, на который указывал изначальный объект, и возвращает новый объект, оставляя старый без изменения. По причине неизменяемой идеологии функциональности данного класса. Также данный класс реализует базовую временную арифметику: сложение и вычитание дат, - с помощью функций `add` и `sub`, а вход которым подаются объекты типа `DateInterval`, обозначающие временной интервал. Так как класс неизменяемый, то возвращается новый измененный экземпляр класса. Рассмотрим следующий листинг 6.20:

Листинг 6.20. Действия с экземплярами класса `DateTimeImmutable`

```
<?php
$datetime = date_create_immutable();
$newdatetime = $datetime->add(new DateInterval('P1W2D'));
echo "Было ".$datetime->format('Y-m-d H:i:s')." Стало
".$newdatetime->format('Y-m-d H:i:s');
```

В примере создается новый экземпляр класса `DateTimeImmutable` в процедурном стиле на основе текущего времени. Далее идет создание еще одного экземпляра класса `DateTimeImmutable` за счет прибавления к текущему таймштампу интервала. Функция `format` преобразует класс к строковому представлению на основе переданного формата.

Класс `DateTimeZone` представляет собой часовой пояс и различные информационные методы о часовых поясах: информация о расположении часового пояса, имя часового пояса, смещение часового пояса от UTC (GMT), все переходы для часового пояса и другая сопутствующая информация. В следующем листинге 6.21 представлен простой пример использования вышеописанной функциональности:

Листинг 6.21. Пример получения информации из класса `DateTimeZone`

```
<?php
$timezone = timezone_open('Europe/London');
var_dump($timezone->getLocation());
var_dump($timezone->getName());
```

В данном примере создается экземпляр класса `DateTimeZone` на основе часового пояса столицы Великобритании города Лондон и идет получение информационной справки о данном часовом поясе.

### 6.2.2. Работа с изображениями

Язык программирования PHP предоставляет множество стандартных расширений для обработки изображений, самые известные из них упомянуты в официальной документации: `exif` (изменяемая информация об изображении), `gd` (мощный инструмент обработки изображений), `Gmagick`, `ImageMagick` и другие. Самой известной и широко используемой является библиотека `GD`, на основе ее возможностей реализовано множество пакетов для продвинутой и упрощенной обработки и генерации изображений. Например, ее функциональность использует вышеописанная библиотека для построения графиков `JpGraph`. Рассмотрим функциональные возможности данной библиотеки подробнее.

PHP не ограничен созданием только HTML страничек. Он так же позволяет создавать и работать с файлами изображений в различных форматах включая `GIF`, `PNG`, `JPEG`, `WBMP`, и `XPM`. Что более удобно, PHP позволяет выводить изображение в потоке непосредственно в браузер. Для этого необходимо скомпилировать PHP с графической библиотекой `GD`, содержащей функции для работы с изображениями. Для работы с `GD` могут потребоваться другие библиотеки (в зависимости от формата изображений, с которыми возникла необходимость работать). Для определения с какими форматами может работать версия `GD` проекта нужно воспользоваться функцией `gd_info()`, возвращающей информационный массив о поддержке различных форматов изображений из следующего списка: `JPEG`, `PNG`, `GIF`, `XDM`, `XPM`, `WBMP`, `WebP`, `BMP` и т.д.

Данная библиотека чаще всего идет в стандартной поставке PHP, если PHP скомпилирован с определенными библиотеками и важно это контролировать и варьировать. Для дистрибутивов докер доступна установка через скрипт `docker-php-ext-install`. Для точного определения всех потребностей рекомендуется обратиться к официальной документации.

Для демонстрации возможностей данной библиотеки приведем простой пример генерации синего квадрата в листинге 6.22:

Листинг 6.22. Генерация синего квадрата с помощью библиотеки `GD`

```
<?php
header('content-type: image/png');

$image = imagecreate(125, 125);
$blue = imagecolorallocate($image, 0, 0, 255);
imagepng($image);
imagedestroy($image);
```

В данном примере с помощью заголовка ответа задается тип ответа изоб-

ражение в формате png. Далее идет создание изображения с помощью функции `imagecreate` с параметрами высота и ширина изображения в пикселях. Возвращается идентификатор изображения `GdImage`, представляющий из себя пустое изображение заданного размера. Далее идет создание цвета для изображения, цвет задается в формате RGB. Первый вызов данной функции задает цвет фона в палитровых изображениях – изображениях, созданных функцией `imagecreate()`. Следующим действием является Вывод PNG изображения в браузер с помощью функции `imagepng`. Последним действием является уничтожение изображения, то есть освобождение ресурсов. Уже с PHP 8.0.0 данное действие не требуется.

Рассмотрим более функциональный пример генерации изображения с текстом в листинге 6.23:

Листинг 6.23. Генерация изображения с текстом

```
<?php
header('content-type: image/png');

$image = imagecreate(300, 300);
$dark_grey = imagecolorallocate($image, 102, 102, 102);
$white = imagecolorallocate($image, 255, 255, 255);
$font_path = 'advent_light';
$string = 'Hello World!';
imaggotfttext($image, 50, 0, 10, 160, $white, $font_path,
$string);
imagepng($image);
imagedestroy($image);
```

В данном примере с помощью заголовка ответа задается тип ответа изображение в формате png. Далее идет создание изображения с помощью функции `imagecreate` с параметрами высота и ширина изображения в пикселях. Возвращается идентификатор изображения `GdImage`, представляющий из себя пустое изображение заданного размера. Далее идет задание светло-серого фона изображения и привязка белого цвета к изображению. Выделяется используемый шрифт и текст для написания с помощью функции `imaggotfttext` идет рисование текста на изображении шрифтом TrueType. Далее изображение выводится в браузер и ресурсы очищаются.

## СПИСОК ИСТОЧНИКОВ

1. Документация // PHP URL: <https://www.php.net/manual/ru/index.php> (дата обращения: 16.10.2021). – Текст: электронный.
2. Дергачев, А. М. Проблемы эффективного использования сетевых сервисов / А. М. Дергачев // Научно-технический вестник СПбГУ ИТМО. – 2011. – № 1 (71). С. 83-87.
3. Розенфельд, Л. Информационная архитектура в Интернете / Л. Розенфельд, П. Морвиль, 2 е издание. – Пер. с англ. – СПб: Символ Плюс, 2005 – 544 с.
4. Спинеллис, Д. Идеальная архитектура. Ведущие специалисты о красоте программных архитектур / Д. Спинеллис, Г. Гусиос. – Пер. с англ. – СПб.: Символ Плюс, 2010 – 528 с.
5. Фаулер, М. Архитектура корпоративных программных приложений / М. Фаулер Пер. с англ. – М.: Издательский дом "Вильямс", 2006 – 544 с.
6. Информационные технологии / О.Л. Голицына, Н.В. Максимов, Т.Л. Партыка, И.И. Попов. – 2-е изд. – Москва: ФОРУМ – ИНФРА-М, 2008. – 395 с.
7. Таненбаум, Э. Распределенные системы. Принципы и парадигмы / Э. Таненбаум, М. ван Стеен. – СПб.: Питер, 2003. – 93 с.
8. Котеров, Д.В. PHP 7 / Д.В. Котеров, И. В. Симдянов. - СПб.: БХВ-Петербург, 2021. – 1088 с.
9. Скляр, Д. PHP. Сборник рецептов / Д. Скляр, А. Трахтенберг. – Пер. с англ. – СПб: Символ Плюс, 2005 – 672 с.
10. Хокинс, С. Администрирование веб-сервера Apache и руководство по электронной коммерции = Apache Web Server Administration and e-Commerce Handbook / С. Хокинс. – М.: Вильямс, 2001. – 336 с.
11. Станек, У. Р. Internet Information Services (IIS) 7.0. Справочник администратора / У. Станек. – СПб.: Русская редакция, 2009. – 528 с.
12. Адамс, К. Администрирование сервера IIS 7 / К. Адамс. – М.: Бином, 2010. – 362 с.
13. Apache vs Nginx: практический взгляд // Хабр URL: <https://habr.com/ru/post/267721/> (дата обращения: 20.10.2021). – Текст: электронный.
14. Internet Information Services // Википедия URL: <https://ru.wikipedia.org/wiki?curid=58959> (дата обращения: 20.10.2021). – Текст: электронный.
15. Клиент (информатика) // Википедия URL:



[https://ru.wikipedia.org/wiki/Клиент\\_\(информатика\)](https://ru.wikipedia.org/wiki/Клиент_(информатика)) (дата обращения: 16.10.2021). – Текст: электронный.

16. Протокол передачи данных // Википедия URL: [https://ru.wikipedia.org/wiki/Протокол\\_передачи\\_данных](https://ru.wikipedia.org/wiki/Протокол_передачи_данных) (дата обращения: 16.10.2021). – Текст: электронный.

17. Клиент – сервер // Википедия URL: [https://ru.wikipedia.org/wiki/Клиент\\_–\\_сервер](https://ru.wikipedia.org/wiki/Клиент_–_сервер) (дата обращения: 16.10.2021). – Текст: электронный.

18. Тузовский, А. Ф. Проектирование и разработка web-приложений: учебное пособие для вузов / А. Ф. Тузовский. – Москва: Издательство Юрайт, 2021. – 218 с.

19. Хоффман, Э. Безопасность веб-приложений. / Э. Хоффман – СПб.: Питер, 2021. – 336 с.

20. Мартин, Р. Чистая архитектура. Искусство разработки программного обеспечения / Р. Мартин. – СПб.: Питер, 2021. – 352 с.

21. Персиваль, Г. Паттерны разработки на Python: TDD, DDD и событийно-ориентированная архитектура / Г. Персиваль, Б. Грегори. – СПб.: Питер, 2022. – 336 с.

22. Раджпут, Д. Spring. Все паттерны проектирования / Д. Раджпут. - СПб.: Питер, 2019. – 320 с.

23. Меджуи, М, Непрерывное развитие API. Правильные решения в изменчивом технологическом ландшафте / М. Меджуи, Э. Уайлд, Р. Митра, М.Амундсен. – СПб.: Питер, 2020. – 272 с.

24. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон. 5-е изд. – СПб.: Питер, 2021. – 320 с.

25. Бэнкс, А. GraphQL: язык запросов для современных веб-приложений / А. Бэнкс, Е. Порсело. – СПб.: Питер, 2019. – 816 с.

26. Антонова И. И., Кашкин Е. В. Разработка web-сервисов с использованием HTML, CSS, PHP и MySQL [Электронный ресурс]: учебно-методическое пособие. – М.: РТУ МИРЭА, 2019. – Режим доступа: <http://library.mirea.ru/secret/15052019/2022.iso>

27. Диков А. В. Клиентские технологии веб-дизайна. HTML5 и CSS3 [Электронный ресурс]: учебное пособие. – Санкт-Петербург: Лань, 2019. – 188 с. – Режим доступа: <https://e.lanbook.com/book/122174>

28. Токарчук, А. М. Применение грид-систем при разворачивании веб-сайта / А. М. Токарчук // Информационно-управляющие системы. – 2010. – №

3(46). – С. 51-55.

29. Как описать архитектуру программы // INFO-LITE.RU URL: <https://info-lite.ru/raznoe/kak-opisat-arhitekturu-programmy.html> (дата обращения: 16.10.2021). – Текст: электронный.

30. PHP // Википедия URL: <https://ru.wikipedia.org/wiki/PHP> (дата обращения: 16.10.2021). – Текст: электронный.

31. Росляков, А. В. Интернет-портал оборудования мультисервисных платформ сетей следующего поколения NGN / А. В. Росляков // Инфокоммуникационные технологии. – 2016. – Т. 14. – № 4. – С. 373-379. – DOI 10.18469/ikt.2016.14.4.04.

32. API // Википедия URL: <https://ru.wikipedia.org/wiki/API> (дата обращения: 16.10.2021). – Текст: электронный.

33. HTTP // Википедия URL: <https://ru.wikipedia.org/wiki?curid=935> (дата обращения: 16.10.2021). – Текст: электронный.

34. REST // Википедия URL: <https://ru.wikipedia.org/wiki?curid=1628507> (дата обращения: 16.10.2021). – Текст: электронный.

35. HTTP/2 // Википедия URL: <https://ru.wikipedia.org/wiki?curid=5957019> (дата обращения: 16.10.2021). – Текст: электронный.

## **СВЕДЕНИЯ ОБ АВТОРАХ**

Волков Михаил Юрьевич, преподаватель кафедры инструментального и прикладного программного обеспечения Института информационных технологий РТУ МИРЭА

Литвинов Владимир Владимирович, ассистент кафедры инструментального и прикладного программного обеспечения Института информационных технологий РТУ МИРЭА

Лобанов Александр Анатольевич, к.т.н., доцент кафедры инструментального и прикладного программного обеспечения Института информационных технологий РТУ МИРЭА

Синицын Анатолий Васильевич, старший преподаватель кафедры инструментального и прикладного программного обеспечения Института информационных технологий РТУ МИРЭА