Веб-прога рубеж

1. Преимущества и недостатки AJAX
2. Директива `page`: назначение, особенности, атрибуты
3. Написать конфигурацию сервлета (`org.xxx.MyServlet`) с помощью аннотации. Сервлет должен принимать все запросы от файлов `.html`, `.xhtml`

Билет 19

1. HTTP методы
2. Thymeleaf. Особенности архитектуры. Отличия от FreeMarker.
3. Напишите JS функцию, которая заменит все текстовые поля на кнопки с тем же текстом

Билет 20

1. Filter: постобработка и предобработка
2. Thymeleaf выражения. Стандартные выражения
3. Написать FastCGI скрипт, который формирует форму с логином и паролем, которая при сабмите отправляет данные на `authenticate.fcgi` посредством SuperAgent, и если успешно, тогда перенаправить пользователя на `main.fcgi`

Билет 21

1. Коды состояния HTTP
2. JSP Expression Language (EL): что это, зачем нужно, чем отличается от JSP, как обрабатывается веб-контейнером
3. Повернуть все картинки на 90 градусов в форме с `id="dhdhgd"`

Билет 22

1. CSS: назначение, правила, приоритеты
2. MVC: назначение, элементы, примеры реализации
3. Реализовать функцию на JavaScript, которая будет закрывать текущее окно, если в нем открыт `https://www.google.ru`

Билет 23

1. DOM и BOM
2. Управление сессиями. HttpSession
3. Функция JavaScript, запрещающая для всех текстовых полей ввод символов, если они не латинские буквы или не цифры

Билет 24

1. ECMAScript
2. Правила трансляции JSP
3. HTML форма, которая отправляет ответ на вопрос из теста, при этом должен отправляться номер ответа (a,b,c...f) и номер вопроса

Билет 25

1. Структура HTTP-запроса
2. Типы данных в PHP
3. Написать код сервлета, который будет перенаправлять все запросы на `https://google.com`

Билет 1

1. Основные принципы JavaScript

JavaScript — это высокоуровневый, интерпретируемый язык программирования, широко используемый для создания динамических и интерактивных веб-страниц.

Основные принципы JavaScript:

- **Динамическая типизация**: Переменные могут содержать значения любого типа без явного объявления типа.
- **Прототипное наследование**: Вместо классов используется прототипное наследование, где объекты наследуют свойства и методы друг от друга.

- **Функции как объекты первого класса**: Функции можно присваивать переменным, передавать как аргументы и возвращать из других функций.
- **Асинхронность и событийно-ориентированное программирование**: Использование колбэков, промисов и `async/await` для обработки асинхронных операций.
- **Замыкания**: Функции имеют доступ к переменным из внешней области видимости, в которой они были созданы.

2. JSP Action

JSP Actions — это специальные теги в JSP (JavaServer Pages), которые выполняют встроенные функции и взаимодействуют с сервлетами и JavaBeans без необходимости писать Java-код внутри JSP-страницы.

Основные JSP Action теги:

- `<jsp:include>`: Включает содержимое другого ресурса во время выполнения.

```
<jsp:include page="header.jsp" />
```

- `<jsp:forward>`: Перенаправляет запрос на другой ресурс.

```
<jsp:forward page="login.jsp" />
```

- `<jsp:useBean>`: Создает или находит экземпляр JavaBean.

```
<jsp:useBean id="user" class="com.example.UserBean" scope
="session" />
```

- `<jsp:setProperty>`: Устанавливает значение свойства JavaBean.

```
<jsp:setProperty name="user" property="name" value="John D
oe" />
```

- `<jsp:getProperty>`: Получает значение свойства JavaBean и выводит его.

```
<jsp:getProperty name="user" property="name" />
```

3. FastCGI + Fetch API, делающий HTML с приветствием через GET (на Java и Apache)

Серверная часть (FastCGI на Java):

```
import com.fastcgi.FCGIInterface;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.util.HashMap;
import java.util.Map;

public class GreetingFastCGIServer {
    public static void main(String[] args) throws IOException
{
        FCGIInterface fcgiInterface = new FCGIInterface();
        while (fcgiInterface.FCGIaccept() >= 0) {
            try {
                String queryString = System.getProperty("QUER
Y_STRING");
                Map<String, String> params = new HashMap<>();
                if (queryString != null && !queryString.isEmp
ty()) {
                    String[] pairs = queryString.split("&");
                    for (String pair : pairs) {
                        String[] keyValue = pair.split("=");
                        if (keyValue.length == 2) {
                            params.put(keyValue[0], keyValue
[1]);
                        }
                    }
                }

                String name = params.getOrDefault("name", "Го
```

```
сть");

                String html = "<html><body><h1>Привет, " + na
me + "!</h1></body></html>";
                String response = "Status: 200 OK\\r\\n" +
                        "Content-Type: text/html; charset=utf
-8\\r\\n" +
                        "Content-Length: " + html.getBytes(St
andardCharsets.UTF_8).length + "\\r\\n\\r\\n" +
                        html;

                FCGIInterface.request.outStream.write(respons
e.getBytes(StandardCharsets.UTF_8));
                FCGIInterface.request.outStream.flush();

            } catch (Exception e) {
                String errorHtml = "<html><body><h1>Ошибка: "
+ e.getMessage() + "</h1></body></html>";
                String response = "Status: 500 Internal Serve
r Error\\r\\n" +
                        "Content-Type: text/html; charset=utf
-8\\r\\n" +
                        "Content-Length: " + errorHtml.getByt
es(StandardCharsets.UTF_8).length + "\\r\\n\\r\\n" +
                        errorHtml;

                FCGIInterface.request.outStream.write(respons
e.getBytes(StandardCharsets.UTF_8));
                FCGIInterface.request.outStream.flush();
            }
        }
    }
}
```

Конфигурация Apache для FastCGI:

```
<IfModule mod_fcgid.c>
    ScriptAlias /fcgi-bin/ "/var/www/html/fcgi-bin/"

    <Directory "/var/www/html/fcgi-bin/">
        AllowOverride None
        Options +ExecCGI
        Require all granted
        SetHandler fcgid-script
    </Directory>
</IfModule>
```

Клиентская часть (HTML с Fetch API):

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>Приветствие</title>
</head>
<body>
    <script>
        fetch('/fcgi-bin/GreetingFastCGIServer?name=Иван')
            .then(response => response.text())
            .then(html => {
                document.body.innerHTML = html;
            })
            .catch(error => console.error('Ошибка:', error));
    </script>
</body>
</html>
```

Пояснение:

- **Серверная часть**: FastCGI сервер на Java, который обрабатывает GET-запросы, извлекает параметр `name` и возвращает HTML с приветствием.

- **Apache конфигурация**: Настраивает Apache на передачу запросов к FastCGI скрипту.
- **Клиентская часть**: Использует Fetch API для отправки запроса и отображения ответа на странице.

Билет 2

1. Интеграция FastCGI скриптов с веб-сервером (Apache)

Интеграция FastCGI с Apache:

1. **Установка модуля mod_fcgid:**
   - Убедитесь, что модуль **mod_fcgid** установлен и загружен.
2. **Конфигурация Apache:**

```
LoadModule fcgid_module modules/mod_fcgid.so

<IfModule mod_fcgid.c>
    ScriptAlias /fcgi-bin/ "/var/www/html/fcgi-bin/"

    <Directory "/var/www/html/fcgi-bin/">
        AllowOverride None
        Options +ExecCGI
        Require all granted
        SetHandler fcgid-script
    </Directory>
</IfModule>
```

3. **Размещение FastCGI скриптов:**
   - Поместите ваши FastCGI скрипты в директорию `/var/www/html/fcgi-bin/`.
4. **Запуск FastCGI приложений:**
   - Убедитесь, что FastCGI приложения на Java запущены и готовы обрабатывать запросы.

5. **Тестирование:**
    - Отправьте запросы к FastCGI скриптам через браузер или инструменты вроде `curl`.

2. Что такое FreeMarker template engine, механизм работы и особенности

FreeMarker — шаблонизатор на Java для генерации текстовых выходных данных на основе шаблонов и модели данных.

Механизм работы:

- **Шаблоны**: Файлы с расширением `.ftl`, содержащие текст и директивы FreeMarker.
- **Модель данных**: Java-объекты (обычно `Map` или JavaBeans), передаваемые в шаблон.
- **Процессор**: FreeMarker объединяет шаблон и модель данных для генерации выходного текста.

Особенности:

- **Разделение логики и представления**: Позволяет разработчикам и дизайнерам работать независимо.
- **Гибкий синтаксис**: Поддержка условий, циклов, макросов.
- **Расширяемость**: Возможность добавления пользовательских функций и директив.

3. Написать фрагмент кода HTML и стили CSS, которые блокируют определенный контент при ширине экрана < 1024 пикселей

HTML:

```
<div class="restricted-content">
    <p>Этот контент доступен только на экранах шириной более 1024 пикселей.</p>
</div>
```

CSS:

```css
@media screen and (max-width: 1023px) {
    .restricted-content {
        display: none;
    }
}
```

Пояснение:

- Используем медиазапрос `@media` для применения стилей при определенных условиях (ширина экрана).
- Правило `display: none;` скрывает элемент с классом `restricted-content` при ширине экрана меньше 1024 пикселей.

Билет 3

1. FastCGI + Fetch API: особенности синтаксиса, использование в веб-приложениях (на Java и Apache)

FastCGI на Java:

- **Особенности синтаксиса:**
  - Использование библиотек для работы с FastCGI протоколом (например, `FCGIInterface`).
  - Обработка запросов в цикле, используя метод `FCGIaccept()`.
  - Чтение переменных окружения и потоков ввода/вывода для получения данных запроса и отправки ответа.
- **Использование в веб-приложениях:**
  - **Высокая производительность**: Постоянные процессы позволяют обрабатывать больше запросов с меньшей задержкой.
  - **Масштабируемость**: Возможность запуска нескольких экземпляров приложения.

- **Гибкость**: Возможность писать серверную логику на Java с использованием всех возможностей языка.

Fetch API:

- **Особенности синтаксиса:**
  - Современный способ выполнения HTTP-запросов из браузера.
  - Использует промисы для обработки асинхронных операций.
  - Поддерживает методы `GET`, `POST`, `PUT`, `DELETE` и т.д.
- **Использование в веб-приложениях:**
  - **Обмен данными с сервером** без перезагрузки страницы.
  - **Динамическое обновление контента** на странице на основе ответов от сервера.
  - **Улучшение пользовательского опыта** за счет быстрой и интерактивной работы приложения.

2. Long Polling и WebSockets: зачем нужны, сходства и различия

Long Polling:

- **Зачем нужен:**
  - Для получения данных с сервера в реальном времени, имитируя постоянное соединение.
- **Механизм:**
  - Клиент отправляет запрос к серверу и держит соединение открытым до тех пор, пока не появятся новые данные.
  - После получения данных соединение закрывается, и клиент сразу же отправляет новый запрос.

WebSockets:

- **Зачем нужны:**
  - Для установления постоянного двунаправленного соединения между клиентом и сервером.

- **Механизм:**
    - Использует один сокет для обмена данными в режиме реального времени без дополнительной нагрузки на сервер.

Сходства:

- Оба метода используются для обновления данных на клиенте в реальном времени.

Различия:

- **Эффективность:** WebSockets более эффективны, так как устанавливается одно постоянное соединение.
- **Сложность:** WebSockets требуют специальной поддержки на сервере и могут быть сложнее в настройке.
- **Нагрузка на сервер:** Long Polling может создавать дополнительную нагрузку из-за большого количества открытых соединений.

3. Написать JSP страницу, которая выводит количество пользователей, которые отправляли запросы за последние 60 секунд

JSP-код:

```
<%@ page import="java.util.concurrent.ConcurrentHashMap, jav
a.util.Map, java.util.Iterator" %>
<%
    ServletContext context = getServletContext();
    ConcurrentHashMap<String, Long> userMap = (ConcurrentHash
Map<String, Long>) context.getAttribute("userMap");

    if (userMap == null) {
        userMap = new ConcurrentHashMap<>();
        context.setAttribute("userMap", userMap);
    }

    String userIP = request.getRemoteAddr();
    long currentTime = System.currentTimeMillis();
```

```
    userMap.put(userIP, currentTime);

    // Удаление записей старше 60 секунд
    Iterator<Map.Entry<String, Long>> iterator = userMap.entr
ySet().iterator();
    while (iterator.hasNext()) {
        Map.Entry<String, Long> entry = iterator.next();
        if ((currentTime - entry.getValue()) > 60000) {
            iterator.remove();
        }
    }

    int userCount = userMap.size();
%>
<p>Количество пользователей за последние 60 секунд: <%= userC
ount %></p>
```

Пояснение:

- **ConcurrentHashMap** используется для хранения IP-адресов пользователей и времени их последнего запроса.
- При каждом запросе текущий IP-адрес и время сохраняются в `userMap`.
- Проходит очистка записей старше 60 секунд.
- Отображается количество уникальных пользователей за последний минуту.

Билет 4

1. Библиотека jQuery: её назначение и основные API

Назначение:

- **jQuery** — популярная JavaScript-библиотека, созданная для упрощения работы с DOM, обработки событий, создания анимаций и выполнения

AJAX-запросов с кроссбраузерной совместимостью.

Основные API:

- **Манипуляция DOM:**
  - Выбор элементов: `$('.class')`, `$('#id')`, `$('[attribute=value]')`.
  - Изменение содержимого: `.html()`, `.text()`, `.val()`.
  - Управление атрибутами: `.attr()`, `.removeAttr()`, `.addClass()`, `.removeClass()`.
- **Обработка событий:**
  - Привязка событий: `.on()`, `.click()`, `.hover()`.
  - Снятие обработчиков: `.off()`.
- **Анимации и эффекты:**
  - Показ и скрытие элементов: `.show()`, `.hide()`, `.toggle()`.
  - Анимации: `.animate()`, `.fadeIn()`, `.fadeOut()`, `.slideUp()`, `.slideDown()`.
- **AJAX-запросы:**
  - Простые запросы: `.load()`, `.get()`, `.post()`.
  - Расширенные настройки: `$.ajax()`.
- **Утилиты:**
  - Обход коллекций: `$.each()`.
  - Определение типа данных: `$.type()`.

2. Архитектура Model 1 и Model 2: особенности, отличия, сферы применения

Model 1:

- **Особенности:**
  - Логика и представление объединены в одной JSP-странице.
  - JSP отвечает за обработку запросов, бизнес-логику и генерацию HTML.
- **Преимущества:**

- Простота реализации для небольших приложений.

- **Недостатки:**
  - Трудно поддерживать и масштабировать.
  - Нарушение принципа разделения логики и представления.
- **Сфера применения:**
  - Небольшие приложения с простой логикой.

Model 2 (MVC):

- **Особенности:**
  - Разделение на Model (модель), View (представление), Controller (контроллер).
  - Контроллер (обычно сервлет) обрабатывает запросы, взаимодействует с моделью и определяет, какое представление вернуть.
- **Преимущества:**
  - Легче поддерживать и масштабировать.
  - Четкое разделение обязанностей.
- **Недостатки:**
  - Более сложная архитектура для небольших приложений.
- **Сфера применения:**
  - Средние и крупные приложения с сложной бизнес-логикой.

3. Шаблон и код инициализации контекста Thymeleaf, формирующие HTML-страницу с текущими курсами валют

Контроллер (Spring MVC):

```
@Controller
public class CurrencyController {

    @GetMapping("/exchange-rates")
```

```
    public String getExchangeRates(Model model) {
        // Получение данных о курсах валют из сервиса
        CurrencyData data = currencyService.getLatestRates();
        model.addAttribute("data", data);
        return "exchangeRates";
    }
}
```

Шаблон Thymeleaf (`exchangeRates.html`):

```
<!DOCTYPE html>
<html xmlns:th="<http://www.thymeleaf.org>">
<head>
    <title>Курсы валют</title>
</head>
<body>
    <h1>Текущие курсы валют</h1>
    <p>Доллар США (USD): <span th:text="${data.usdRate}">USD
Rate</span> RUB</p>
    <p>Евро (EUR): <span th:text="${data.eurRate}">EUR Rate</
span> RUB</p>
    <h2>Динамика изменения</h2>
    <p>USD: <span th:text="${data.usdChange}">USD Change</spa
n>%</p>
    <p>EUR: <span th:text="${data.eurChange}">EUR Change</spa
n>%</p>
</body>
</html>
```

Инициализация контекста Thymeleaf:

```
@Configuration
public class ThymeleafConfig {

    @Bean
    public SpringTemplateEngine templateEngine() {
```

```
        SpringTemplateEngine templateEngine = new SpringTempl
ateEngine();
        // Настройка шаблонов и диалектов
        return templateEngine;
    }

    @Bean
    public ViewResolver viewResolver() {
        ThymeleafViewResolver viewResolver = new ThymeleafVie
wResolver();
        viewResolver.setTemplateEngine(templateEngine());
        return viewResolver;
    }
}
```

Билет 5

1. Jakarta EE: архитектура, компоненты, контейнер

Jakarta EE (ранее Java EE) — это платформа для разработки корпоративных приложений на языке Java. Она предоставляет стандартизированные API для создания масштабируемых, безопасных и надежных приложений.

Архитектура:

- **Многоуровневая архитектура**: Состоит из клиентского уровня, уровня веб-компонентов, уровня бизнес-логики и уровня данных.
- **Контейнеры**: Исполняющая среда для компонентов Jakarta EE, обеспечивающая управление жизненным циклом, безопасность, транзакции и другие системные службы.

Компоненты Jakarta EE:

- **Servlets**: Обработка HTTP-запросов.
- **JavaServer Pages (JSP)**: Динамическое создание веб-страниц.
- **Enterprise JavaBeans (EJB)**: Реализация бизнес-логики.

- **Java Persistence API (JPA)**: Работа с базами данных.
- **Java Message Service (JMS)**: Асинхронный обмен сообщениями.

Контейнер:

- **Web Container**: Исполняет веб-компоненты (Servlets, JSP).
- **EJB Container**: Исполняет EJB-компоненты.

2. Синтаксис шаблонов и модель данных FreeMarker

FreeMarker — шаблонизатор на Java для генерации текстовых выходных данных на основе шаблонов и модели данных.

Синтаксис шаблонов:

- **Переменные**: `${variable}`
- **Директивы**:
  - **Условия**: `<#if condition> ... </#if>`
  - **Циклы**: `<#list collection as item> ... </#list>`
  - **Макросы**: `<#macro name> ... </#macro>`
- **Комментарии**: `<#-- Comment -->`

Модель данных:

- Представляет собой `Map` или JavaBean.
- Передается в шаблон и используется для заполнения переменных и управления логикой.

3. Заменить все гиперссылки на текстовое поле со значением ссылки

JavaScript-код:

```
document.querySelectorAll('a').forEach(function(link) {
    var input = document.createElement('input');
    input.type = 'text';
    input.value = link.href;
```

```
    link.parentNode.replaceChild(input, link);
});
```

Билет 6

1. Конфигурация FastCGI + Fetch API, способы интеграции FastCGI с веб-приложением

Конфигурация FastCGI на Apache:

- **Установка модуля mod_fcgid**:

```
LoadModule fcgid_module modules/mod_fcgid.so
```

- **Настройка Apache для обработки FastCGI-приложений**:

```
<IfModule mod_fcgid.c>
    FcgidInitialEnv QUERY_STRING ""
    FcgidInitialEnv REQUEST_METHOD "GET"

    <Directory "/var/www/html/fastcgi/">
        Options +ExecCGI
        Require all granted
        AddHandler fcgid-script .fcgi
    </Directory>
</IfModule>
```

- **Размещение FastCGI-приложений в директории** `/var/www/html/fastcgi/`.

Способы интеграции:

- **Через модуль mod_fcgid**: Apache обрабатывает запросы и передает их FastCGI-приложению на Java.
- **Использование Fetch API на клиенте**: Асинхронные запросы к FastCGI-приложению.

2. ServletContext — что это и для чего применяется

ServletContext — объект, предоставляющий информацию о веб-приложении и позволяющий сервлетам взаимодействовать.

Применение:

- **Хранение общих данных**: Доступных всем сервлетам и JSP в приложении.
- **Получение параметров инициализации**: Заданных в `web.xml`.
- **Доступ к ресурсам**: Чтение файлов внутри приложения.

Пример использования:

```
// Сохранение атрибута
getServletContext().setAttribute("attributeName", attributeVa
lue);

// Получение атрибута
Object value = getServletContext().getAttribute("attributeNam
e");
```

3. JSP-страница, отображающая корзину покупателя

Класс ShoppingItem.java:

```
public class ShoppingItem {
    private String name;
    private double price;

    // Конструктор
    public ShoppingItem(String name, double price) {
        this.name = name;
        this.price = price;
    }

    // Геттеры
```

```
    public String getName() { return name; }
    public double getPrice() { return price; }
}
```

Managed Bean (ShoppingCart.java):

```
import java.util.ArrayList;
import java.util.List;

public class ShoppingCart {
    private List<ShoppingItem> items = new ArrayList<>();

    public void addItem(ShoppingItem item) {
        items.add(item);
    }

    public List<ShoppingItem> getItems() {
        return items;
    }
}
```

JSP-страница:

```
<%@ page import="com.example.ShoppingCart, com.example.Shoppi
ngItem" %>
<%
    ShoppingCart cart = (ShoppingCart) session.getAttribute
("cart");
    if (cart == null) {
        cart = new ShoppingCart();
        session.setAttribute("cart", cart);
    }
%>
<h2>Ваша корзина</h2>
<ul>
<% for (ShoppingItem item : cart.getItems()) { %>
```

```
    <li><%= item.getName() %> - <%= item.getPrice() %> руб.</
li>
<% } %>
</ul>
```

Билет 7

1. LESS, SASS, SCSS — отличия от CSS, синтаксис, взаимодействие с браузером

Отличия от CSS:

- **Переменные**: Позволяют хранить повторяющиеся значения.

```
$primary-color: #333;
```

- **Вложенность**: Улучшает читаемость и структуру стилей.

```
nav {
  ul {
    li {
      a {
        color: $primary-color;
      }
    }
  }
}
```

- **Миксины и функции**: Повторное использование блоков кода.

Синтаксис:

- **SASS**: Использует отступы вместо фигурных скобок.
- **SCSS**: Расширение синтаксиса CSS, поддерживает все возможности SASS.
- **LESS**: Похож на SCSS, но с некоторыми отличиями.

Взаимодействие с браузером:

- Препроцессоры компилируются в CSS.
- Компиляция происходит до передачи стилей браузеру.

2. JSTL — что это, зачем это, если есть JSP-элементы, основные теги

JSTL — JavaServer Pages Standard Tag Library.

Зачем нужна:

- Обеспечивает стандартный набор тегов для задач, часто встречающихся в JSP.
- Позволяет уменьшить количество Java-кода в JSP.

Основные теги:

- **Ядро (`c`):**
  - `<c:if>`, `<c:choose>`, `<c:forEach>`, `<c:set>`, `<c:out>`.
- **Форматирование (`fmt`):**
  - `<fmt:formatNumber>`, `<fmt:formatDate>`.
- **Функции (`fn`):**
  - Строковые функции, например, `fn:contains()`, `fn:length()`.

3. Структура HTTP запроса, отправляющего логин и пароль пользователя

Пример запроса:

```
POST /login HTTP/1.1
Host: www.example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: [length]

username=user&password=pass
```

Билет 8

1. SuperAgent

SuperAgent — JavaScript-библиотека для выполнения AJAX-запросов.

Особенности:

- Простота использования.
- Поддержка промисов и колбэков.
- Поддержка Node.js и браузеров.

Пример использования:

```
superagent.get('/api/users')
  .then(response => {
    console.log(response.body);
  })
  .catch(error => {
    console.error(error);
  });
```

2. Что такое шаблон проектирования и чем отличается от архитектурного шаблона

Шаблон проектирования — повторяемое решение общей проблемы в определенном контексте разработки ПО.

Архитектурный шаблон — высокоуровневое решение для общей структуры системы.

Отличия:

- **Шаблон проектирования** фокусируется на решении задач на уровне классов и объектов.
- **Архитектурный шаблон** определяет макроуровневую структуру приложения.

3. Написать AJAX-запрос, который получает JSON и выводит его элементы

Используя SuperAgent:

```
superagent.get('/api/data')
  .end((err, res) => {
    if (res.ok) {
      const data = res.body;
      data.forEach(item => {
        console.log(item);
      });
    } else {
      console.error(err);
    }
  });
```

Билет 9

1. Как сервлеты обрабатывают HTTP-запрос

- **Получение запроса**: Веб-сервер передает запрос сервлет-контейнеру.
- **Создание объектов**: `HttpServletRequest` и `HttpServletResponse`.
- **Вызов метода сервлета**: В зависимости от HTTP-метода (`doGet()`, `doPost()`).
- **Обработка запроса**: Чтение параметров, бизнес-логика.
- **Формирование ответа**: Установка статуса, заголовков, запись данных в ответ.
- **Отправка ответа**: Клиент получает ответ от сервера.

2. Многоуровневая архитектура: элементы, зачем нужна

Элементы:

- **Презентационный уровень**: Интерфейс пользователя.

- **Логический уровень**: Бизнес-логика.
- **Уровень доступа к данным**: Взаимодействие с БД.
- **Уровень данных**: Базы данных.

Зачем нужна:

- **Разделение обязанностей**: Легче поддерживать и масштабировать.
- **Повторное использование**: Компоненты могут быть использованы в разных приложениях.
- **Масштабируемость**: Возможность масштабировать уровни независимо.

3. С помощью FreeMarker сверстать страничку с оценками студентов, отсортировать по времени получения оценки

Java-код:

```
List<Grade> grades = getGrades(); // Получение списка оценок
grades.sort(Comparator.comparing(Grade::getTimestamp).reverse
d()); // Сортировка по времени

Map<String, Object> dataModel = new HashMap<>();
dataModel.put("grades", grades);

// Загрузка шаблона и генерация страницы
```

Шаблон FreeMarker (`grades.ftl`):

```
<h1>Оценки студентов</h1>
<table>
  <tr>
    <th>Студент</th>
    <th>Оценка</th>
    <th>Дата</th>
  </tr>
  <#list grades as grade>
    <tr>
```

```
      <td>${grade.studentName}</td>
      <td>${grade.value}</td>
      <td>${grade.timestamp?string("dd.MM.yyyy HH:mm")}</td>
    </tr>
  </#list>
</table>
```

Билет 10

1. REST и RPC

REST (Representational State Transfer):

- **Описание:**
  - Архитектурный стиль для создания распределенных гипермедиа систем.
  - Использует стандарты веба, такие как HTTP методы (GET, POST, PUT, DELETE).
  - Ресурсы идентифицируются через URI.
  - Без сохранения состояния (stateless): каждый запрос содержит всю необходимую информацию.
- **Преимущества:**
  - Простота и гибкость.
  - Масштабируемость за счет кеширования и балансировки нагрузки.
  - Широкая совместимость и использование стандартных протоколов.

RPC (Remote Procedure Call):

- **Описание:**
  - Протокол для выполнения вызовов процедур или методов на удаленном сервере.

    - Предполагает, что удаленные вызовы выглядят как локальные.
    - Использует различные протоколы передачи (JSON-RPC, XML-RPC, gRPC).
- **Преимущества:**
    - Семантика похожа на локальные вызовы функций.
    - Подходит для микросервисной архитектуры.

Сравнение:

- **REST** ориентирован на ресурсы и их представления, **RPC** — на действия и процедуры.
- REST использует стандартные HTTP методы, RPC может использовать собственные протоколы.
- REST лучше подходит для веб-приложений, RPC — для сервисов с жесткими требованиями к производительности.

2. Диспетчеризация запросов в веб-приложениях на Java. Интерфейс RequestDispatcher

RequestDispatcher:

- **Описание:**
    - Интерфейс в Servlet API для передачи запроса от одного сервлета к другому или для включения содержимого другого ресурса в ответ.
    - Предоставляет механизмы `forward` и `include`.
- **Методы:**
    - `forward(ServletRequest request, ServletResponse response)`:
        - Перенаправляет запрос к другому ресурсу на стороне сервера.
        - Клиент не знает о перенаправлении (URL в браузере не меняется).
    - `include(ServletRequest request, ServletResponse response)`:
        - Включает содержимое другого ресурса в ответ текущего сервлета.

- **Применение:**
  - Разделение логики обработки запросов и формирования ответов.
  - Повторное использование общих компонентов (например, хедеров и футеров страниц).

Пример использования:

```
RequestDispatcher dispatcher = request.getRequestDispatcher
("/otherServlet");
dispatcher.forward(request, response);
```

3. Написать правило на CSS, что у всех посещённых ссылок будет жёлтый фон, кроме тех, которые находятся внутри элемента с классом "news"

CSS правило:

```
a:visited {
    background-color: yellow;
}

.news a:visited {
    background-color: initial;
}
```

Пояснение:

- Первое правило устанавливает желтый фон для всех посещенных ссылок.
- Второе правило отменяет это свойство для ссылок внутри элементов с классом `news`.

Билет 11

1. Структура HTML документа

Основные элементы HTML-документа:

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>Заголовок страницы</title>
    <!-- Метаданные, стили, скрипты -->
</head>
<body>
    <!-- Содержимое страницы -->
</body>
</html>
```

Пояснения:

- `<!DOCTYPE html>`: Указывает браузеру, что документ использует HTML5.
- `<html>`: Корневой элемент HTML-документа.
- `<head>`: Содержит метаданные, такие как заголовок страницы, ссылки на стили и скрипты.
- `<body>`: Содержит видимое содержимое страницы.

2. Сервлеты — особенности, преимущества и недостатки относительно CGI, FastCGI

Сервлеты:

- **Особенности:**
  - Являются Java-классами, обрабатывающими HTTP-запросы.
  - Работают внутри сервлет-контейнера (например, Apache Tomcat).
  - Поддерживают многопоточность.
- **Преимущества:**
  - **Высокая производительность**: Не нужно создавать новый процесс на каждый запрос.

- **Безопасность**: Используют механизмы безопасности Java.
- **Масштабируемость**: Легко обрабатывать большое количество одновременных запросов.
- **Обширный API**: Доступ к библиотекам Java.

- **Недостатки относительно CGI/FastCGI:**
  - **Сложность настройки**: Требуется настройка сервлет-контейнера.
  - **Менее гибкие**: CGI-скрипты могут быть написаны на разных языках.

3. CSS правило: рисовать границу в 1 пиксель для картинок в блоках новостей (class="news") при наведении на них курсора

CSS правило:

```css
.news img:hover {
    border: 1px solid black;
}
```

Пояснение:

- Правило применяется к изображениям внутри элементов с классом `news`.
- При наведении курсора на изображение (`:hover`) отображается черная граница толщиной 1 пиксель.

Билет 13

1. Цикл жизни сервлета

Этапы жизненного цикла:

1. **Загрузка и инициализация:**
   - Сервлет-контейнер загружает класс сервлета и создает его экземпляр.

- Вызывается метод `init(ServletConfig config)` для инициализации сервлета.

2. **Обработка запросов:**
   - На каждый запрос вызывается метод `service(ServletRequest req, ServletResponse res)`.
   - `service()` определяет тип запроса и вызывает соответствующий метод (`doGet()`, `doPost()` и т.д.).
3. **Завершение работы:**
   - При остановке сервлета или сервера вызывается метод `destroy()`.
   - Освобождаются ресурсы, связанные с сервлетом.

Схема:

- `init()` → `service()` (многократно) → `destroy()`

2. Диалекты и процессоры Thymeleaf и стандартный диалект

Thymeleaf:

- **Диалекты:**
  - Наборы правил и синтаксиса для обработки шаблонов.
  - **Стандартный диалект** предоставляет базовые функциональности для создания веб-приложений.
- **Процессоры:**
  - Компоненты, обрабатывающие определенные атрибуты или теги в шаблоне.
  - Могут быть расширены для создания пользовательских диалектов.

Стандартный диалект включает:

- **Шаблонные атрибуты:**
  - `th:text`: Выводит текстовое значение.
  - `th:each`: Итерация по коллекциям.
  - `th:if`, `th:unless`: Условное отображение.

- `th:include`, `th:replace`: Включение фрагментов.

3. Написать сценарий, который будет считать количество слов «де-факто» во всех тегах `div` с классом `lecture`. Ещё надо обязательно использовать jQuery.

JavaScript с использованием jQuery:

```
$(document).ready(function() {
    var count = 0;
    $('div.lecture').each(function() {
        var text = $(this).text();
        var matches = text.match(/\\вде-факто\\b/gi);
        if (matches) {
            count += matches.length;
        }
    });
    console.log('Количество слов "де-факто": ' + count);
});
```

Пояснение:

- `$(document).ready()`: Убедимся, что DOM загружен.
- `$('div.lecture').each()`: Проходим по всем `div` с классом `lecture`.
- `$(this).text()`: Получаем текстовое содержимое элемента.
- `text.match(/\\вде-факто\\b/gi)`: Ищем все вхождения слова «де-факто» независимо от регистра.
- **Суммируем количество найденных вхождений.**

Билет 14

1. AJAX и DHTML — описание, сходства и различия

AJAX (Asynchronous JavaScript and XML):

- **Описание:**

- Технология для обмена данными с сервером асинхронно, без перезагрузки страницы.
- Позволяет обновлять части веб-страницы динамически.

DHTML (Dynamic HTML):

- **Описание:**
  - Комбинация HTML, CSS и JavaScript для создания интерактивных и динамических веб-страниц.
  - Позволяет изменять содержимое и стиль страницы на стороне клиента.

Сходства:

- Оба используются для создания динамичных веб-приложений.
- Используют JavaScript для манипуляции DOM.

Различия:

- **AJAX** фокусируется на общении с сервером для получения данных.
- **DHTML** фокусируется на изменении страницы на стороне клиента без взаимодействия с сервером.

2. Какие проблемы возникают при параллельной обработке запросов в JSP, как этого можно избежать?

Проблемы:

- **Потоконебезопасность:**
  - Использование общих или экземплярных переменных может привести к конфликтам при одновременном доступе из разных потоков.
- **Состояние сеанса:**
  - Неправильное управление данными сессии может привести к утечкам памяти или ошибкам.

Как избежать:

- **Избегать использования экземплярных переменных:**

    - Все переменные должны быть локальными или храниться в сессии.

- **Синхронизация доступа:**
    - Если необходимо использовать общие ресурсы, синхронизировать доступ к ним.
- **Использование потоко-безопасных коллекций и объектов:**
    - Например, `ConcurrentHashMap`.

3. Написать JS функцию, которая заменяет содержимое `<div>` с именем класса `nyan` на изображение по ссылке: `http://www.example.com/nyancat.gif`

JavaScript:

```
function replaceDivContent() {
    var divs = document.querySelectorAll('div.nyan');
    divs.forEach(function(div) {
        div.innerHTML = '<img src="<http://www.example.com/ny
ancat.gif>" alt="Nyan Cat">';
    });
}

// Вызов функции
replaceDivContent();
```

Пояснение:

- Находим все `div` с классом `nyan`.
- Заменяем их содержимое на тег `<img>` с указанной ссылкой.

Билет 16

1. JSP — элементы

Основные элементы JSP:

1. **Директивы JSP:**
   - `<%@ page %>`: Задает атрибуты страницы, такие как язык, кодировка, импорт пакетов.
   - `<%@ include %>`: Включает другой файл при компиляции страницы.
   - `<%@ taglib %>`: Подключает библиотеку тегов.
2. **Скриплеты:**
   - `<% ... %>`: Блоки Java-кода, выполняемые при обработке страницы.
3. **Выражения:**
   - `<%= ... %>`: Выражения Java, результаты которых вставляются в выходной поток.
4. **Декларации:**
   - `<%! ... %>`: Объявление полей и методов класса, в который компилируется JSP.
5. **Комментарии:**
   - **JSP комментарии (`<%-- ... --%>`):** Не выводятся в исходный код страницы и не компилируются.

2. CGI обработка запроса, преимущества и недостатки

CGI (Common Gateway Interface):

- **Преимущества:**
  - **Простота реализации:** Можно писать на любом языке программирования.
  - **Изолированность процессов:** Каждый запрос обрабатывается отдельным процессом.
- **Недостатки:**
  - **Низкая производительность:** Создание процесса на каждый запрос затратно.
  - **Плохая масштабируемость:** Не подходит для высоконагруженных приложений.

- **Отсутствие состояния:** Сложно сохранять данные между запросами.

3. Написать сервлет, который принимает из HTTP запроса параметр `name` и выводит его

Если параметр не обнаружен, то вывести "Anonymous user"

Код сервлета:

```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
import java.io.IOException;

@WebServlet("/hello")
public class HelloServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServ
letResponse response) throws IOException {
        String name = request.getParameter("name");
        if (name == null || name.trim().isEmpty()) {
            name = "Anonymous user";
        }
        response.setContentType("text/html; charset=UTF-8");
        response.getWriter().write("<h1>Hello, " + name + "!
</h1>");
    }
}
```

Пояснение:

- Получаем параметр `name` из запроса.
- Если параметр отсутствует или пустой, присваиваем значение "Anonymous user".
- Устанавливаем тип контента и выводим приветствие.

Билет 17

3. Если параметр не обнаружен, то вывести "Anonymous user"

Код сервлета:

```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
import java.io.IOException;

@WebServlet("/greet")
public class GreetServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServ
letResponse response) throws IOException {
        String name = request.getParameter("name");
        if (name == null || name.trim().isEmpty()) {
            name = "Anonymous user";
        }
        response.setContentType("text/html; charset=UTF-8");
        response.getWriter().write("<h1>Hello, " + name + "!
</h1>");
    }
}
```

Пояснение:

- Проверяем наличие параметра `name` в запросе.
- Если параметр отсутствует или пуст, устанавливаем значение `"Anonymous user"`.
- Выводим приветствие с полученным или установленным именем.

Билет 18

1. Преимущества и недостатки AJAX

AJAX (Asynchronous JavaScript and XML):

Преимущества:

- **Асинхронность:** Позволяет обновлять части веб-страницы без перезагрузки всей страницы, улучшая пользовательский опыт.
- **Снижение нагрузки на сервер:** Отправляются только необходимые данные, что экономит ресурсы сервера и трафик.
- **Интерактивность:** Обеспечивает более динамичные и отзывчивые интерфейсы.
- **Плавный UX:** Пользователь не замечает задержек, связанных с загрузкой страниц.

Недостатки:

- **Сложность отладки:** Асинхронный код сложнее тестировать и отлаживать.
- **Проблемы с SEO:** Контент, загружаемый через AJAX, может быть невидим для поисковых систем.
- **История браузера:** Действия AJAX не отражаются в истории браузера, затрудняя навигацию.
- **Безопасность:** Увеличивается риск уязвимостей, таких как XSS, если не проводить правильную валидацию данных.

2. Директива `page`: назначение, особенности, атрибуты

Директива `page` в JSP определяет атрибуты страницы, влияющие на ее поведение и свойства.

Назначение:

- Устанавливает параметры страницы, такие как кодировка, импорт классов, буферизация, обработка ошибок и т.д.

Особенности:

- Размещается в начале JSP-файла.
- Не выводит содержимое на страницу.
- Может содержать несколько атрибутов, разделенных пробелами.

Основные атрибуты:

- `language`: Язык программирования страницы (обычно `"java"`).
- `import`: Импорт Java-классов и пакетов.

```
<%@ page import="java.util.List, com.example.MyClass" %>
```

- `contentType`: MIME-тип ответа и кодировка.

```
<%@ page contentType="text/html; charset=UTF-8" %>
```

- `session`: Указывает, используется ли сессия на странице (`true`/`false`).
- `buffer`: Размер буфера вывода (например, `"8kb"`).
- `autoFlush`: Автоматическое очищение буфера (`true`/`false`).
- `isThreadSafe`: Потокобезопасность страницы (`true`/`false`).
- `errorPage`: Путь к странице обработки ошибок.
- `isErrorPage`: Указывает, является ли страница страницей обработки ошибок (`true`/`false`).
- `pageEncoding`: Кодировка страницы.

3. Написать конфигурацию сервлета (`org.xxx.MyServlet`) с помощью аннотации. Сервлет должен принимать все запросы от файлов `.html`, `.xhtml`

Код сервлета с аннотацией:

```
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;

@WebServlet(urlPatterns = {"*.html", "*.xhtml"})
public class MyServlet extends HttpServlet {
    // Реализация методов doGet(), doPost() и др.
}
```

Пояснение:

- Используется аннотация `@WebServlet` для конфигурации сервлета без `web.xml`.
- Параметр `urlPatterns` задает шаблоны URL, обрабатываемые сервлетом.
- Шаблоны `.html` и `.xhtml` позволяют сервлету обрабатывать все запросы к файлам с указанными расширениями.

Билет 19

1. HTTP методы

Основные HTTP методы:

- **GET**: Запрашивает представление ресурса. Используется для получения данных без изменения состояния сервера.
- **POST**: Передает данные на сервер для создания нового ресурса. Изменяет состояние сервера.
- **PUT**: Обновляет существующий ресурс или создаёт новый, если ресурса не было.
- **DELETE**: Удаляет указанный ресурс.
- **HEAD**: Запрашивает заголовки ресурса без тела ответа.
- **OPTIONS**: Запрашивает поддерживаемые сервером методы для ресурса.
- **PATCH**: Вносит частичные изменения в ресурс.
- **CONNECT**: Устанавливает туннель к серверу, определенному ресурсом.
- **TRACE**: Возвращает запрос, полученный сервером, для отладки.

Применение:

- **GET**: Получение данных (например, веб-страниц, API данных).
- **POST**: Отправка форм, загрузка файлов.
- **PUT/DELETE/PATCH**: Используются в RESTful API для изменения состояния ресурсов.

2. Thymeleaf. Особенности архитектуры. Отличия от FreeMarker.

Thymeleaf:

- **Особенности архитектуры:**
  - Шаблонизатор для Java-приложений, поддерживающий HTML, XML, JavaScript, CSS и текстовые файлы.
  - Концепция "естественных шаблонов" (Natural Templates), которые могут быть корректно отображены как в браузере, так и обработаны сервером.
  - Использует HTML5 атрибуты для внедрения логики в шаблоны.
- **Отличия от FreeMarker:**
  - **Синтаксис:**
    - **Thymeleaf**: Использует стандартные HTML-атрибуты с префиксом `th:` (например, `th:text`), что делает шаблоны валидными HTML документами.
    - **FreeMarker**: Использует специальные теги и директивы (`<#...>`, `${...}`), что может нарушать валидность HTML при просмотре в браузере.
  - **Природа шаблонов:**
    - **Thymeleaf**: Шаблоны могут быть просмотрены как статические HTML-страницы без предварительной обработки.
    - **FreeMarker**: Шаблоны не отображаются корректно без предварительной обработки сервером.
  - **Интеграция с фреймворками:**
    - **Thymeleaf** тесно интегрирован с Spring Framework, поддерживая Spring Expression Language (SpEL).
    - **FreeMarker** более универсален, но требует дополнительных настроек для интеграции.

3. Напишите JS функцию, которая заменит все текстовые поля на кнопки с тем же текстом

JavaScript:

```
function replaceTextFieldsWithButtons() {
    const textFields = document.querySelectorAll('input[type
="text"]');
    textFields.forEach(function(field) {
        const button = document.createElement('button');
        button.textContent = field.value;
        field.parentNode.replaceChild(button, field);
    });
}

// Вызов функции
replaceTextFieldsWithButtons();
```

Пояснение:

- Используем `querySelectorAll` для выбора всех текстовых полей (`<input type="text">`).
- Для каждого найденного поля создаем кнопку `<button>`, устанавливаем ее текст равным значению поля.
- Заменяем текстовое поле на кнопку в DOM.

Билет 20

1. Filter: постобработка и предобработка

Фильтры (Filters) в Java Servlet API:

Предобработка:

- Выполняется перед передачей запроса сервлету.
- Примеры использования:

- Проверка аутентификации и авторизации пользователя.
- Логирование входящих запросов.
- Установка кодировки запроса.
- Проверка и валидация параметров запроса.

Постобработка:

- Выполняется после того, как сервлет обработал запрос и сгенерировал ответ.
- Примеры использования:
  - Компрессия или шифрование выходных данных.
  - Добавление или изменение заголовков ответа.
  - Логирование исходящих ответов.
  - Кэширование ответов.

Реализация фильтра:

```
import javax.servlet.*;
import java.io.IOException;

public class MyFilter implements Filter {
    public void init(FilterConfig filterConfig) throws Servle
tException {
        // Инициализация фильтра
    }

    public void doFilter(ServletRequest request, ServletRespo
nse response, FilterChain chain) throws IOException, ServletE
xception {
        // Предобработка
        // Например, логирование запроса
        System.out.println("Request received at " + new java.
util.Date());
```

```
        // Передача управления следующему фильтру или сервлет
у
        chain.doFilter(request, response);

        // Постобработка
        // Например, логирование ответа
        System.out.println("Response sent at " + new java.uti
l.Date());
    }

    public void destroy() {
        // Освобождение ресурсов
    }
}
```

Регистрация фильтра:

- **Через аннотацию:**

```
@WebFilter("/*")
public class MyFilter implements Filter {
    // ...
}
```

- **Через `web.xml`:**

```
<filter>
    <filter-name>MyFilter</filter-name>
    <filter-class>com.example.MyFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>MyFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

2. Thymeleaf выражения. Стандартные выражения

Типы выражений в Thymeleaf:

1. **Variable Expressions (Вариабельные выражения):** `${...}`
   - Доступ к переменным модели.
   - Пример: `<span th:text="${user.name}">Имя пользователя</span>`
2. **Selection Variable Expressions (Селекционные выражения):** `{...}`
   - Используются внутри селекционного контекста (`th:object`).
   - Пример:

```
<form th:object="${user}">
    <input type="text" th:field="*{name}" />
</form>
```

3. **Message Expressions (Сообщения):** `#{...}`
   - Доступ к сообщениям из файлов локализации.
   - Пример: `<p th:text="#{welcome.message}">Добро пожаловать</p>`
4. **Link URL Expressions (URL выражения):** `@{...}`
   - Формирование URL с учетом контекста приложения.
   - Пример: `<a th:href="@{/login}">Вход</a>`
5. **Fragment Expressions (Фрагментные выражения):** `~{...}`
   - Используются для включения фрагментов шаблонов.
   - Пример: `<div th:insert="~{commons::header}"></div>`
6. **Literal Expressions (Литералы):**
   - Строковые: `'text'`
   - Числовые: `123`
   - Булевы: `true`, `false`
   - Null: `null`

7. **Logical and Arithmetic Operations (Логические и арифметические операции):**
   - Арифметические: `+`, `-`, `*`, `/`, `%`
   - Логические: `and`, `or`, `not`
8. **Conditionals (Условные выражения):**
   - Используются в `th:if`, `th:unless`.
   - Пример: `<div th:if="${user != null}">Привет, ${user.name}!</div>`

3. Написать FastCGI скрипт, который формирует форму с логином и паролем, которая при сабмите отправляет данные на `authenticate.fcgi` посредством SuperAgent, и если успешно, тогда перенаправить пользователя на `main.fcgi`

Серверная часть (`login.fcgi`):

```
import com.fastcgi.FCGIInterface;
import java.io.IOException;
import java.nio.charset.StandardCharsets;

public class LoginFastCGIServer {
    public static void main(String[] args) throws IOException
{
        FCGIInterface fcgiInterface = new FCGIInterface();
        while (fcgiInterface.FCGIaccept() >= 0) {
            String htmlForm = """
                <html>
                <body>
                    <form id="loginForm">
                        <label>Логин: <input type="text" name
="username"></label><br>
                        <label>Пароль: <input type="password"
name="password"></label><br>
                        <button type="submit">Войти</button>
                    </form>
```

```
                    <script src="<https://cdn.jsdelivr.net/np
m/superagent>"></script>
                    <script>
                        const form = document.getElementById
('loginForm');
                        form.addEventListener('submit', funct
ion(event) {
                            event.preventDefault();
                            const formData = new FormData(for
m);
                            const data = {
                                username: formData.get('usern
ame'),
                                password: formData.get('passw
ord')
                            };
                            superagent.post('/authenticate.fc
gi')
                                .send(data)
                                .end(function(err, res) {
                                    if (res && res.ok) {
                                        window.location.href
= '/main.fcgi';
                                    } else {
                                        alert('Ошибка авториз
ации');
                                    }
                                });
                        });
                    </script>
                </body>
                </html>
                """;
            String response = "Status: 200 OK\\r\\n" +
                    "Content-Type: text/html; charset=utf-8
\\r\\n" +
```

```
                    "Content-Length: " + htmlForm.getBytes(St
andardCharsets.UTF_8).length + "\\r\\n\\r\\n" +
                    htmlForm;
            FCGIInterface.request.outStream.write(response.ge
tBytes(StandardCharsets.UTF_8));
            FCGIInterface.request.outStream.flush();
        }
    }
}
```

Пояснение:

- **Форма логина:** Содержит поля для ввода имени пользователя и пароля.
- **SuperAgent:** Используется для отправки AJAX-запроса на `authenticate.fcgi`.
- **Обработка ответа:**
  - При успешной аутентификации происходит перенаправление на `main.fcgi`.
  - При ошибке выводится сообщение об ошибке.

Серверная часть (`authenticate.fcgi`):

- Должна обрабатывать POST-запрос, проверять учетные данные и возвращать соответствующий статус.

Билет 21

1. Коды состояния HTTP

Коды состояния HTTP — это трехзначные числа, которые сервер возвращает в ответ на запрос клиента. Они информируют о результате обработки запроса.

Основные категории кодов состояния:

- **1xx (Informational)** — Информационные сообщения:
  - **100 Continue**: Клиент может продолжать запрос.

    - **101 Switching Protocols**: Сервер переключается на другой протокол по запросу клиента.
- **2xx (Success)** — Успешные операции:
    - **200 OK**: Запрос успешно обработан.
    - **201 Created**: Создан новый ресурс.
    - **204 No Content**: Нет содержимого для возврата.
- **3xx (Redirection)** — Перенаправление:
    - **301 Moved Permanently**: Ресурс перемещен навсегда.
    - **302 Found**: Ресурс временно перемещен.
    - **304 Not Modified**: Ресурс не изменялся.
- **4xx (Client Error)** — Ошибки клиента:
    - **400 Bad Request**: Неверный запрос.
    - **401 Unauthorized**: Необходима аутентификация.
    - **403 Forbidden**: Доступ запрещен.
    - **404 Not Found**: Ресурс не найден.
- **5xx (Server Error)** — Ошибки сервера:
    - **500 Internal Server Error**: Внутренняя ошибка сервера.
    - **502 Bad Gateway**: Неверный шлюз.
    - **503 Service Unavailable**: Сервис недоступен.

Применение:

- Информирование клиента о статусе запроса.
- Помощь в обработке ошибок и перенаправлений.
- Улучшение взаимодействия между клиентом и сервером.

2. JSP Expression Language (EL): что это, зачем нужно, чем отличается от JSP, как обрабатывается веб-контейнером

JSP Expression Language (EL) — это язык выражений, встроенный в JSP, который упрощает доступ к данным, хранящимся в JavaBeans-компонентах, и других объектах.

Зачем нужна:

- Упрощение синтаксиса доступа к данным.
- Повышение читабельности кода.
- Избежание использования Java-кода внутри JSP.

Отличия от JSP скриплетов:

- **EL** предоставляет сокращенный синтаксис для доступа к данным, в то время как **скриплеты** позволяют вставлять произвольный Java-код.
- **EL** автоматически обрабатывается контейнером, обеспечивая безопасность и упрощение кода.

Как обрабатывается веб-контейнером:

- EL выражения распознаются и оцениваются контейнером во время выполнения страницы.
- Контейнер заменяет EL выражения результатами их оценки.
- Предоставляется доступ к различным областям видимости: `pageScope`, `requestScope`, `sessionScope`, `applicationScope`.

Примеры использования:

- **Доступ к атрибутам:**

```
${user.name}
```

- **Логические операции:**

```
${order.total > 100}
```

- **Выражения в атрибутах тегов:**

```
<c:if test="${not empty user}">
    <p>Привет, ${user.name}!</p>
</c:if>
```

3. Повернуть все картинки на 90 градусов в форме с `id="dhdhgd"`

CSS решение:

```
#dhdhgd img {
    transform: rotate(90deg);
}
```

Пояснение:

- Селектор `#dhdhgd img` находит все изображения внутри формы с `id="dhdhgd"`.
- Свойство `transform: rotate(90deg);` поворачивает изображения на 90 градусов по часовой стрелке.

Билет 22

1. CSS: назначение, правила, приоритеты

Назначение CSS:

- Описание внешнего вида HTML-документов.
- Разделение содержания и представления.
- Управление стилями элементов: цвет, шрифт, расположение и др.

Правила CSS:

- **Синтаксис:**

```
селектор {
    свойство: значение;
}
```

- **Селекторы:** Определяют, к каким элементам применяются стили (элементы, классы, идентификаторы).
- **Каскадность:** Правила применяются в порядке специфичности и источника (стили браузера, разработчика, пользователя).

Приоритеты:

- **Inline стили** (наиболее высокий приоритет).
- **ID селекторы** (`#id`).
- **Классовые селекторы** (`.class`), атрибутные селекторы, псевдоклассы.
- **Элементные селекторы** (`div`, `p`).
- **!important** переопределяет обычные приоритеты.

2. MVC: назначение, элементы, примеры реализации

Назначение MVC:

- Разделение приложения на три компонента для улучшения модульности и упрощения поддержки.

Элементы MVC:

1. **Model (Модель):** Логика приложения, управление данными.
2. **View (Представление):** Отображение данных пользователю.
3. **Controller (Контроллер):** Обработка пользовательского ввода, взаимодействие модели и представления.

Примеры реализации:

- **Java EE:** Сервлеты (контроллеры), JSP (представления), JavaBeans или EJB (модель).
- **Spring MVC:** Контроллеры с аннотацией `@Controller`, модели как сервисы и DAO, представления на основе шаблонизаторов (Thymeleaf, JSP).

3. Реализовать функцию на JavaScript, которая будет закрывать текущее окно, если в нем открыт `https://www.google.ru`

JavaScript функция:

```
if (window.location.href === '<https://www.google.ru/>' || wi
ndow.location.href === '<https://www.google.ru>') {
    window.close();
}
```

Замечание:

- Браузеры обычно запрещают скриптам закрывать окна, которые не были открыты скриптами.
- Функция `window.close()` сработает только если окно было открыто с помощью `window.open()`.

Билет 23

1. DOM и BOM

DOM (Document Object Model):

- Представляет структуру HTML или XML документа в виде дерева объектов.
- Позволяет JavaScript взаимодействовать и изменять содержимое страницы.
- Методы: `getElementById`, `querySelector`, `createElement` и др.

BOM (Browser Object Model):

- Предоставляет средства для взаимодействия с браузером.
- Содержит объекты: `window`, `navigator`, `location`, `history`, `screen`.
- Позволяет управлять окнами, навигацией, получать информацию о системе.

2. Управление сессиями. HttpSession

HttpSession:

- Интерфейс для хранения информации о пользователе между запросами.
- **Создание или получение сессии:**

```
HttpSession session = request.getSession();
```

- **Установка атрибутов:**

```
session.setAttribute("key", value);
```

- **Получение атрибутов:**

```
Object value = session.getAttribute("key");
```

- **Удаление атрибутов:**

```
session.removeAttribute("key");
```

- **Инвалидизация сессии:**

```
session.invalidate();
```

Механизмы идентификации сессии:

- **Cookies:** Идентификатор сессии хранится в cookie `JSESSIONID`.
- **URL Rewriting:** Идентификатор добавляется в URL, если cookies отключены.

3. Функция JavaScript, запрещающая для всех текстовых полей ввод символов, если они не латинские буквы или не цифры

JavaScript функция:

```
document.addEventListener('DOMContentLoaded', function() {
    var inputs = document.querySelectorAll('input[type="tex
t"]');
```

```
    inputs.forEach(function(input) {
        input.addEventListener('keypress', function(event) {
            var char = String.fromCharCode(event.which);
            if (!/[a-zA-Z0-9]/.test(char)) {
                event.preventDefault();
            }
        });
    });
});
```

Билет 24

1. ECMAScript

ECMAScript — стандарт языка программирования, на основе которого построен JavaScript.

Ключевые версии:

- **ES5 (2009):** Введение `strict mode`, улучшения для массивов, методы `Object`.
- **ES6 (ES2015):** Добавлены `let`, `const`, стрелочные функции, классы, шаблонные строки, промисы, модули.
- **ES7 (ES2016):** Оператор возведения в степень `*`, метод `Array.prototype.includes`.

Особенности:

- Улучшение синтаксиса и возможностей языка.
- Повышение производительности и удобства разработки.
- Поддержка современных фреймворков и библиотек.

2. Правила трансляции JSP

Этапы трансляции JSP:

1. **Парсинг JSP:** Разделение на статический и динамический контент.

2. **Генерация сервлета:** Преобразование JSP в Java-код сервлета.
3. **Компиляция сервлета:** Компиляция Java-кода в байт-код.
4. **Выполнение сервлета:** Обработка запросов и формирование ответов.

Правила:

- **Статический контент:** Преобразуется в вызовы `out.write()`.
- **Скриплеты:** Код внутри `<% ... %>` вставляется в метод `service()`.
- **Выражения:** `<%= ... %>` преобразуются в `out.print()`.
- **Директивы:** Обрабатываются при генерации сервлета (например, `import` пакетов).

3. HTML форма, которая отправляет ответ на вопрос из теста, при этом должен отправляться номер ответа (a,b,c...f) и номер вопроса

HTML форма:

```
<form action="/submitAnswer" method="post">
    <input type="hidden" name="questionNumber" value="1">
    <label>
        <input type="radio" name="answer" value="a"> Вариант
A
    </label><br>
    <label>
        <input type="radio" name="answer" value="b"> Вариант
B
    </label><br>
    <label>
        <input type="radio" name="answer" value="c"> Вариант
C
    </label><br>
    <!-- Дополнительные варианты -->
    <button type="submit">Отправить</button>
</form>
```

Пояснение:

- Используем скрытое поле для передачи номера вопроса.
- Радиокнопки для выбора варианта ответа.
- При отправке формы сервер получает `questionNumber` и `answer`.

Билет 25

1. Структура HTTP-запроса

Пример HTTP-запроса:

```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0
Accept: text/html
```

Составляющие:

1. **Стартовая строка:**
   - Метод (GET, POST и т.д.)
   - Путь к ресурсу
   - Версия протокола
2. **Заголовки запроса:**
   - Метаданные о запросе и клиенте.
   - Примеры: `Host`, `User-Agent`, `Accept`, `Cookie`.
3. **Тело запроса (для методов POST, PUT):**
   - Содержит данные, отправляемые на сервер.

2. Типы данных в PHP

Основные типы данных в PHP:

- **Scalar Types (Скалярные типы):**

    - **boolean**
    - **integer**
    - **float (double)**
    - **string**
- **Compound Types (Составные типы):**
    - **array**
    - **object**
    - **callable**
    - **iterable**
- **Special Types (Специальные типы):**
    - **resource**
    - **NULL**

Преобразование типов:

- Автоматическое преобразование при необходимости.
- Явное преобразование с помощью `(int)`, `(string)` и т.д.

3. Написать код сервлета, который будет перенаправлять все запросы на `https://google.com`

Код сервлета:

```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
import java.io.IOException;

@WebServlet("/*")
public class RedirectServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServ
letResponse response) throws IOException {
        response.sendRedirect("<https://google.com>");
```

```
    }

    protected void doPost(HttpServletRequest request, HttpSer
vletResponse response) throws IOException {
        response.sendRedirect("<https://google.com>");
    }
}
```

Пояснение:

- Аннотация `@WebServlet("/*")` заставляет сервлет обрабатывать все запросы.
- Метод `sendRedirect()` отправляет клиенту ответ с кодом 302 и новым URL.
- Клиент перенаправляется на `https://google.com`.

Билет 26

1. ECMAScript: преимущества 6 и 7 версий

ECMAScript 6 (ES6):

- **Блочная область видимости:** `let` и `const`.
- **Стрелочные функции:** Упрощенный синтаксис, не имеет своего `this`.
- **Классы и наследование:** Синтаксический сахар для прототипного наследования.
- **Шаблонные строки:** Вставка выражений и многострочные строки.
- **Деструктуризация:** Извлечение данных из массивов и объектов.
- **Промисы:** Работа с асинхронным кодом.
- **Модули:** `import` и `export`.

ECMAScript 7 (ES7):

- **Оператор экспоненты:** `*`.
- **Метод** `Array.prototype.includes`**:** Проверка наличия элемента в массиве.

Преимущества:

- Улучшение синтаксиса и читабельности.
- Поддержка современных подходов в программировании.
- Повышение производительности и удобства разработки.

2. GoF паттерны. Что это такое? Основные виды, примеры

GoF (Gang of Four) паттерны:

- Классические шаблоны проектирования, описанные в книге "Design Patterns: Elements of Reusable Object-Oriented Software" четырьмя авторами.

Основные категории:

1. **Порождающие (Creational):**
    - **Singleton:** Гарантирует единственный экземпляр класса.
    - **Factory Method:** Определяет интерфейс для создания объекта, но оставляет подклассам решение о том, какой класс инстанцировать.
    - **Abstract Factory:** Предоставляет интерфейс для создания семейств связанных объектов.
2. **Структурные (Structural):**
    - **Adapter:** Позволяет объектам с несовместимыми интерфейсами работать вместе.
    - **Decorator:** Динамически добавляет объектам новые обязанности.
3. **Поведенческие (Behavioral):**
    - **Observer:** Определяет зависимость "один ко многим" между объектами.
    - **Strategy:** Определяет семейство алгоритмов, инкапсулирует каждый из них и делает их взаимозаменяемыми.

Примеры:

- **Singleton в Java:**

```
public class Singleton {
    private static Singleton instance;

    private Singleton() {}

    public static Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }
}
```

3. Написать HTML страницу и сервлет, возвращающий странице количество сессий

Сервлет (SessionCounterServlet.java):

```
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import javax.servlet.annotation.WebListener;
import javax.servlet.http.*;

@WebListener
public class SessionCounterListener implements HttpSessionLis
tener {
    private static int activeSessions = 0;

    public void sessionCreated(HttpSessionEvent se) {
        activeSessions++;
    }

    public void sessionDestroyed(HttpSessionEvent se) {
        activeSessions--;
    }
```

```
    public static int getActiveSessions() {
        return activeSessions;
    }
}
```

Сервлет для отображения количества сессий:

```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
import java.io.IOException;

@WebServlet("/sessionCount")
public class SessionCountServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServ
letResponse response) throws IOException {
        int sessionCount = SessionCounterListener.getActiveSe
ssions();
        response.setContentType("text/html; charset=UTF-8");
        response.getWriter().write("<p>Активных сессий: " + s
essionCount + "</p>");
    }
}
```

HTML страница:

```
<!DOCTYPE html>
<html>
<head>
    <title>Счетчик сессий</title>
</head>
<body>
    <iframe src="/sessionCount" style="border:none;"></iframe
>
```

```
</body>
</html>
```

Билет 27

1. FastCGI: плюсы, минусы, отличия от CGI

CGI (Common Gateway Interface):

- **Описание:**
  - Интерфейс для взаимодействия веб-сервера с внешними программами.
  - При каждом запросе сервер создает новый процесс для выполнения CGI-скрипта.

FastCGI:

- **Описание:**
  - Улучшенная версия CGI, разработанная для повышения производительности.
  - FastCGI-приложения запускаются как постоянные процессы и могут обслуживать множество запросов.

Плюсы FastCGI:

- **Повышенная производительность:**
  - Отсутствие необходимости создавать новый процесс на каждый запрос.
- **Масштабируемость:**
  - Возможность обработки большего числа одновременных запросов.
- **Гибкость:**
  - Поддержка различных языков программирования, включая Java.

Минусы FastCGI:

- **Сложность настройки:**
  - Требует дополнительной конфигурации веб-сервера (Apache, Nginx).
- **Сложность отладки:**
  - Из-за постоянного процесса сложнее отлаживать ошибки.

Отличия от CGI:

- **Производительность:**
  - FastCGI быстрее благодаря постоянным процессам.
- **Ресурсоемкость:**
  - CGI более ресурсоемкий из-за постоянного создания и уничтожения процессов.
- **Управление состоянием:**
  - FastCGI позволяет сохранять состояние между запросами.

2. Суперглобальные массивы в PHP

Суперглобальные массивы — предопределенные массивы в PHP, доступные из любой области видимости без использования ключевого слова `global`.

Основные суперглобальные массивы:

- `$_GET`: Данные из URL-параметров запроса (метод GET).
- `$_POST`: Данные, отправленные через форму (метод POST).
- `$_REQUEST`: Объединяет `$_GET`, `$_POST` и `$_COOKIE`.
- `$_COOKIE`: Данные, хранящиеся в cookies.
- `$_SESSION`: Данные текущей сессии пользователя.
- `$_SERVER`: Информация о сервере и окружении исполнения скрипта.
- `$_FILES`: Информация о загруженных файлах.
- `$_ENV`: Переменные окружения.

Пример использования:

```php
<?php
// Получение данных из формы
$username = $_POST['username'];

// Доступ к информации о сервере
$scriptName = $_SERVER['SCRIPT_NAME'];

// Работа с сессиями
session_start();
$_SESSION['user'] = $username;
?>
```

3. JSP-страница, проверяющая наличие `jsessionid` в запросе и выводящая сообщение об ошибке, если его нет

Код JSP-страницы:

```
<%
    HttpSession session = request.getSession(false); // Не со
здавать новую сессию, если ее нет
    if (session == null) {
        response.setStatus(HttpServletResponse.SC_UNAUTHORIZE
D); // Устанавливаем статус 401
        out.println("<h1>Ошибка 401: Необходима аутентификаци
я</h1>");
    } else {
        // Продолжаем обработку, если сессия существует
        out.println("<h1>Добро пожаловать, " + session.getAtt
ribute("username") + "!</h1>");
    }
%>
```

Пояснение:

- Используем `request.getSession(false)` для получения текущей сессии без создания новой.

- Если сессия отсутствует (`session == null`), устанавливаем статус ответа `401 Unauthorized` и выводим сообщение об ошибке.
- Если сессия существует, продолжаем обработку и выводим приветствие.

Билет 28

1. Правила построения HTML-форм

Основные компоненты HTML-формы:

- `<form>`: Контейнер для элементов формы.
  - **Атрибуты:**
    - `action`: URL-адрес, на который отправляются данные формы.
    - `method`: Метод отправки данных (`GET` или `POST`).
    - `enctype`: Кодировка данных при отправке (например, для загрузки файлов `multipart/form-data`).
- **Элементы ввода данных:**
  - `<input>`: Различные типы полей (текст, пароль, чекбокс, радио-кнопка, файл).
  - `<textarea>`: Многострочное текстовое поле.
  - `<select>` и `<option>`: Выпадающий список.
  - `<label>`: Связывает текст с элементом формы для улучшения доступности.

Правила:

- **Семантика и доступность:**
  - Использовать теги `<label>` для всех полей ввода.
  - Группировать связанные элементы с помощью `<fieldset>` и `<legend>`.
- **Валидация:**
  - Использовать атрибуты `required`, `min`, `max`, `pattern` для клиентской валидации.

- **Безопасность:**
  - Экранировать пользовательский ввод на серверной стороне.
  - Использовать HTTPS для передачи данных.

Пример формы:

```
<form action="/submit" method="post">
    <label for="email">Электронная почта:</label>
    <input type="email" id="email" name="email" required><br>

    <label for="password">Пароль:</label>
    <input type="password" id="password" name="password" requ
ired><br>

    <button type="submit">Войти</button>
</form>
```

2. Конфигурация сервлетов. Файл `web.xml`

Файл `web.xml` — дескриптор развертывания веб-приложения Java, в котором настраиваются сервлеты, фильтры, слушатели и другие компоненты приложения.

Пример конфигурации сервлета:

```
<web-app>
    <servlet>
        <servlet-name>MyServlet</servlet-name>
        <servlet-class>org.xxx.MyServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>MyServlet</servlet-name>
        <url-pattern>/myServlet</url-pattern>
```

```
    </servlet-mapping>
</web-app>
```

Пояснение:

- `<servlet>`: Определяет сервлет, связывая имя с классом.
- `<servlet-mapping>`: Указывает, по какому URL доступен сервлет.

Дополнительные настройки:

- **Инициализационные параметры:**

```
<init-param>
    <param-name>paramName</param-name>
    <param-value>paramValue</param-value>
</init-param>
```

- **Задание загрузки при старте:**

```
<load-on-startup>1</load-on-startup>
```

3. Написать CSS, чтобы по клику все ссылки, кроме внутри `<h1>`, подчеркивались

CSS с использованием псевдокласса `:focus` и JavaScript для постоянного эффекта:

```
a.clicked {
    text-decoration: underline;
}

h1 a.clicked {
    text-decoration: none;
}
```

JavaScript для добавления класса при клике:

```
document.addEventListener('click', function(event) {
    if (event.target.tagName === 'A' && !event.target.closest
('h1')) {
        event.target.classList.add('clicked');
    }
});
```

Пояснение:

- **CSS:**
  - Класс `clicked` добавляет подчеркивание для ссылок.
  - Селектор `h1 a.clicked` отменяет подчеркивание для ссылок внутри `<h1>`.
- **JavaScript:**
  - При клике на ссылку, если она не внутри `<h1>`, добавляется класс `clicked`.

Билет 29

1. FastCGI взаимодействие с веб-сервером

Принцип взаимодействия:

- **Постоянный процесс:**
  - FastCGI-приложения работают как постоянные процессы или пулы процессов.
- **Обмен данными:**
  - Веб-сервер и FastCGI-приложение общаются через сокеты (TCP или Unix-сокеты).
  - Запросы и ответы передаются в виде бинарных сообщений по протоколу FastCGI.
- **Передача параметров:**

- Параметры запроса передаются через переменные окружения.
- Тело запроса (POST-данные) передается через стандартный ввод.

Настройка взаимодействия с Apache:

- **Модуль mod_fcgid:**
  - Apache передает запросы FastCGI-приложению, настроенному на определенный путь или расширение файлов.

Преимущества FastCGI:

- **Высокая производительность:**
  - Отсутствие затрат на запуск процесса для каждого запроса.
- **Масштабируемость:**
  - Возможность обработки большого количества одновременных запросов.

2. FreeMarker. Архитектура, принцип работы, использование в веб-приложениях

Архитектура:

- **Шаблоны (.ftl):**
  - Содержат разметку с директивами FreeMarker.
- **Модель данных:**
  - Набор данных (`Map`, JavaBeans), передаваемых в шаблон.
- **Конфигурация:**
  - Настройки для загрузки шаблонов и обработки данных.

Принцип работы:

1. **Создание конфигурации:**
   - Устанавливаем параметры загрузки шаблонов и другие настройки.
2. **Загрузка шаблона:**
   - Шаблон загружается из файловой системы или другого источника.

3. **Подготовка данных:**
    - Создаем модель данных для передачи в шаблон.
4. **Обработка шаблона:**
    - Генерируем выходной текст, объединяя шаблон и данные.

Использование в веб-приложениях:

- **Генерация динамического контента:**
    - Создание HTML-страниц на основе данных из базы данных или других источников.
- **Интеграция с фреймворками:**
    - Часто используется с Spring MVC и другими Java-фреймворками.

Отличия от других шаблонизаторов:

- **Свободный синтаксис:**
    - Гибкость в создании пользовательских директив и функций.
- **Высокая производительность:**
    - Быстрая обработка шаблонов.

3. Написать страницу на HTML и CSS, скрывающую содержимое от пользователя и показывающую сообщение "Разрешение не поддерживается", если разрешение экрана < 1400

HTML:

```
<!DOCTYPE html>
<html>
<head>
    <title>Проверка разрешения</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="content">
```

```
        <!-- Основное содержимое страницы -->
        <h1>Добро пожаловать на сайт</h1>
        <p>Здесь расположен основной контент.</p>
    </div>
    <div class="unsupported">
        <p>Разрешение не поддерживается</p>
    </div>
</body>
</html>
```

CSS (`style.css`):

```
.content {
    display: block;
}

.unsupported {
    display: none;
    text-align: center;
    font-size: 24px;
    color: red;
}

@media screen and (max-width: 1399px) {
    .content {
        display: none;
    }
    .unsupported {
        display: block;
    }
}
```

Пояснение:

- **Основное содержимое** отображается при ширине экрана от 1400 пикселей и выше.

- **Сообщение о неподдерживаемом разрешении** отображается при ширине экрана менее 1400 пикселей.

Билет 30

1. Реализация объектно-ориентированных программ в PHP

Объектно-ориентированное программирование (ООП) в PHP:

- **Классы и объекты:**
  - Определение классов с помощью ключевого слова `class`.
  - Создание объектов через оператор `new`.

Пример класса:

```
<?php
class Car {
    private $brand;
    private $model;

    public function __construct($brand, $model) {
        $this->brand = $brand;
        $this->model = $model;
    }

    public function getInfo() {
        return "Марка: $this->brand, Модель: $this->model";
    }
}

// Создание объекта
$car = new Car("Toyota", "Camry");
echo $car->getInfo();
?>
```

Особенности ООП в PHP:

- **Наследование:**
  - Классы могут наследовать свойства и методы другого класса с помощью `extends`.
- **Интерфейсы и абстрактные классы:**
  - Использование `interface` и `abstract class` для определения контрактов и абстрактного поведения.
- **Инкапсуляция:**
  - Модификаторы доступа `public`, `protected`, `private` контролируют доступ к свойствам и методам.
- **Полиморфизм:**
  - Возможность методов принимать объекты разных классов, реализующих общий интерфейс.

2. Предопределенные переменные JSP

Implicit Objects (Предопределенные объекты) в JSP:

- `request`: Объект `HttpServletRequest`, содержащий информацию о запросе.
- `response`: Объект `HttpServletResponse`, используемый для формирования ответа.
- `session`: Объект `HttpSession`, позволяющий сохранять данные между запросами.
- `application`: Объект `ServletContext`, общий для всего приложения.
- `out`: Объект `JspWriter`, используемый для вывода данных на страницу.
- `config`: Объект `ServletConfig`, содержащий параметры инициализации сервлета.
- `pageContext`: Объект `PageContext`, предоставляет доступ к различным областям видимости.
- `page`: Ссылка на текущий объект сервлета (аналог `this` в Java).
- `exception`: Объект `Throwable`, доступен на страницах ошибок.

Пример использования:

```
<%
    String userAgent = request.getHeader("User-Agent");
    session.setAttribute("userAgent", userAgent);
    out.println("Ваш браузер: " + userAgent);
%>
```

3. Код фильтра запросов, запрещающий доступ к приложению неавторизованным пользователям (отсутствует заголовок "X-Application-User")

Код фильтра:

```
import javax.servlet.*;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.*;
import java.io.IOException;

@WebFilter("/*")
public class AuthenticationFilter implements Filter {

    public void init(FilterConfig filterConfig) throws Servle
tException {
        // Инициализация фильтра (если необходимо)
    }

    public void doFilter(ServletRequest request, ServletRespo
nse response, FilterChain chain) throws IOException, ServletE
xception {
        HttpServletRequest httpReq = (HttpServletRequest) req
uest;
        HttpServletResponse httpRes = (HttpServletResponse) r
esponse;

        String userHeader = httpReq.getHeader("X-Application-
User");
```

```
        if (userHeader == null || userHeader.isEmpty()) {
            // Отправляем ошибку 401 Unauthorized
            httpRes.sendError(HttpServletResponse.SC_UNAUTHOR
IZED, "Доступ запрещен. Отсутствует заголовок X-Application-U
ser.");
            return;
        }

        // Продолжаем цепочку фильтров и сервлетов
        chain.doFilter(request, response);
    }

    public void destroy() {
        // Освобождение ресурсов (если необходимо)
    }
}
```

Пояснение:

- **Аннотация** `@WebFilter("/*")`**:** Фильтр применяется ко всем запросам.
- **Проверка заголовка:**
  - Извлекаем значение заголовка `X-Application-User`.
  - Если заголовок отсутствует или пуст, отправляем ответ с ошибкой `401 Unauthorized`.
- **Передача запроса дальше:**
  - Если заголовок присутствует, передаем управление следующему фильтру или сервлету.
