### Билет 1

1) Основные принципы JS

2) JSP Action

3) PHP делающий HTML с приветствием через GET

1) JS - Сценарий - код, включенный в состав веб-страницы. Клиентский сценарий выполняется на компьютере-клиенте, для этого необходим встроенный интерпретатор. Вставка сценария в веб-страницу происходит при помощи тега <script>.

JavaScript является объектно-ориентированным языком. JavaScript имеет ряд свойств, присущих функциональным языкам — функции как объекты первого класса, объекты как списки, карринг, анонимные функции, замыкания. Js имеет автоматическое приведение типов, автоматическая сборка мусора, анонимные функции, функции как объекты первого класса (т.е. мб сохранены в переменную, переданы в функцию как аргумент, созданы во время выполнения программы и т.п.).

Динамическая типизация: Позволяет не указывать тип, но при этом язык не определяет его сам. Типы переменных неизвестны до того момента, когда у них появляется конкретное значения при запуске.

Автоматическое управление памятью: Память автоматически выделяется при объявлении переменных, вызове некоторых функций/методов. Сборщик мусора освобождает память, когда больше не существует ссылок, позволяющих добраться до объекта от корня

Прототипное программирование: Стиль объектно-ориентированного программирования, при котором отсутствует понятие класса, а наследование производится путём клонирования существующего экземпляра объекта — прототипа.

Объекты первого класса

Объекты первого класса - это элементы, которые могут быть переданы как параметр, возвращены из функции или присвоены переменной. Определение функции (не вызов!) — это выражение, а значит оно возвращает значение, а именно — функцию, что доказывает, что функции в JS — объекты первого класса

Основные архитектурные черты: динамическая типизация; слабая типизация; автоматическое управление памятью; прототипное программирование; функции как объекты первого класса.

Типы данных

number -- целые, дробные числа, Infinity, NaN; String; boolean; null; undefined -- «значение не присвоено»; object -- для коллекций и более сложных сущностей

2) Действия JSP actions могут воздействовать на стандартный поток вывода, использовать, модифицировать и создавать объекты. actions используют конструкции с синтаксисом XML для управления сервлетом и позволяют, таким образом, динамически подключать файл, многократно использовать компоненты JavaBeans, направлять пользователя на другую страницу или генерировать HTML для Java plugin.

Синтаксис: <jsp:action\_name attribute="value" />

Существует набор стандартных действий, которые должны быть в обязательном порядке реализованы любым контейнером JSP, удовлетворяющим спецификации. Кроме этого, возможно создание новых действий с помощью директивы библиотеки тегов taglib.

Список стандартных действий.

jsp:include - Включает в страницу внешний файл во время обработки запроса

jsp:useBean - Добавляет на страницу экземпляр Java Bean с заданным контекстом

jsp:getProperty, jsp:setProperty Получение и установка свойств Java Bean

jsp:forward - Перенаправление на другую страницу

jsp:declaration – объявление, аналогично <%! … %>

jsp:scriptlet – скриптлет, аналогично <% … %>

jsp:expression – скриптлет, аналогично <%= … %>

jsp:text – вывод текста

3) PHP делающий HTML с приветствием через GET

<?php

if (isset($\_GET['name'])) {

$name = $\_GET['name'];

} else {

$name = "Гость";

}

$html = "<html>

<head>

<title>Приветствие</title>

</head>

<body>

<h1>Привет, " . htmlspecialchars($name) . "!</h1>

</body>

</html>";

echo $html;

?>

---

<?php

if (isset($\_GET['name'])) {

$name = htmlspecialchars($\_GET['name']);

echo "<h1>Привет, $name!</h1>";

} else {

echo "<h1>Добро пожаловать, Гость!</h1>";

}

?>

### Билет 2 (29)

1. Интеграция Fast-CGI скриптов с веб-сервером
2. Что такое FreeMarker template engine, механизм работы и особенности.
3. Написать фрагмент кода html и стили css, которые блокируют определенный контент при ширине экрана < 1024 пикселей.

1) Интеграция Fast-CGI скриптов с веб-сервером

Fast-CGI: Развитие технологии CGI. Все запросы могут обрабатываться одним процессом CGI-программы (в CGI-для каждого запроса запускается, после ответа завершается, а для Fast-CGI-процесс не завершается, а используется для последующих запросов). Веб-сервер взаимодействует с процессом через UNIX Domain Sockets или TCP/IP (а не через stdin и stdout).

Обработка запросов в однопоточном приложении FastCGI выполняется следующим образом:

1) Веб-сервер создает прикладные процессы FastCGI для обработки запросов. Процессы могут создаваться при запуске или по требованию.

2) Программа FastCGI инициализирует себя и ожидает нового подключения от веб-сервера.

3) Когда поступает запрос клиента, веб-сервер открывает соединение с процессом FastCGI. Сервер отправляет информацию о переменной среды CGI и стандартные входные данные по соединению.

4) Процесс FastCGI отправляет стандартные выходные данные и информацию об ошибках обратно на сервер по тому же соединению.

5) Когда процесс FastCGI закрывает соединение, запрос завершен. Затем процесс FastCGI ожидает другого подключения от веб-сервера.

1) Установите и настройте веб-сервер с поддержкой FastCGI, такой как Apache или Nginx.

2) Установите и настройте модуль FastCGI или обработчик для веб-сервера, такой как mod\_fcgid для Apache или fastcgi\_module для Nginx.

3) Настройте веб-сервер для обработки запросов к файлам скриптов (обычно с расширением .fcgi или .php) с помощью модуля или обработчика FastCGI.

4) Напишите скрипты FastCGI, используя соответствующий язык (например, PHP, Python и т.д.) и сохраните их с соответствующим расширением файла (например, .fcgi, .php и т.д.).

5) Протестируйте скрипты FastCGI, обратившись к ним через веб-сервер и убедившись, что они выполняются правильно и генерируют ожидаемый динамический контент.

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

FreeMarker - это  — компилирующий обработчик шаблонов, написанный на Java, один из инструментов, позволяющих отделить логику и данные от представления в духе концепции Model-view-controller. Используется преимущественно при разработке web-приложений с использованием Java-сервлетов, также может использоваться для вывода текста в других случаях: генерация CSS, исходного кода Java и т. д. В отличие от JSP FreeMarker не является зависимым от архитектуры сервлета или от протокола HTTP. Поддерживает: условия, циклы, присваивание значений переменным: <#assign name=value>, арифметику, операции со строками, форматирование, макросы и функции, подключение внешних шаблонов, экранирование символов

Шаблон может содержать следующие элементы:

● Статический HTML.

● Обращения к модели данных:

Welcome ${user}!

● Директивы:

<#if animals.python.price != 0>

Pythons are not free today!

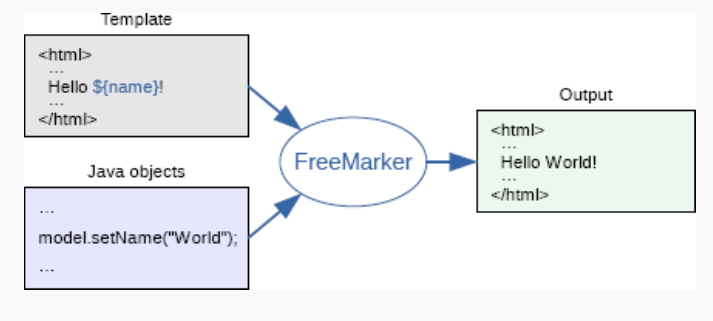
</#if>

● Вызовы встроенных функций:

animals?filter(it -> it.protected)

Принцип работы:

На вход подается шаблон, например html в котором есть специальные выражения, подготавливаются данные соответствующие этим выражением, а Freemarker динамически вставляет эти данные и получается динамически заполненный документ.



3) Написать фрагмент кода html и стили css, которые блокируют определенный контент при ширине экрана < 1024 пикселей.

HTML:

<!DOCTYPE html>

<html>

<head>

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

</head>

<body>

<div class="content">

<p>контент</p>

</div>

</body>

</html>

CSS (style.css):

.content {

display: block;

}

@media screen and (max-width: 1023px) {

.content {

display: none;

}

}

3) Для 29 Написать страницу на HTML и CSS, скрывающую содержимое от пользователя и показать сообщение "разрешение не поддерживается", если разрешение экрана < 1400.

HTML:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<link rel="stylesheet" href="styles.css">

<title>Проверка разрешения экрана</title>

</head>

<body>

<html>

<div class=”abobus”>Разрешение не поддерживается</div>

</body>

</html>

CSS:

.abobus {

display: none;

}

@media only screen and (max-width: 1399px) {

.abobus {

display: block;

}

}

### Билет 3

1) PHP особенности синтаксиса, использование в веб-приложениях

2) Long Polling и Websockets зачем нужны, сходства и различия

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

1) PHP особенности синтаксиса, использование в веб-приложениях:

PHP - это объектно-ориентированный скриптовый язык. Интерпретатор выполняет код, находящийся внутри блока <?php ?>

1. Имена переменных начинаются с символа “$”, а инструкции разделяются символом “;”, а из-за того, что PHP - язык с динамической типизацией, при объявлении тип не указывается:

$a = “Hello”;

$b = “World”;

$c = $a + “ ” + $b;

1. Типы данных: скалярные (integer, float, double, boolean, string и NULL, диапазон зависит от платформы), нескалярные (ресурс, массив и объект) и псевдотипы (mixed, number, callback и void)
2. PHP поддерживает операторы управления потоком, такие как условия (if, else, elseif), циклы
3. (for, while, foreach), операторы switch
4. Суперглобальные массивы - предопределенные массивы, доступные отовсюду:
   1. $\_GLOBALS - все глобальные переменные
   2. $\_SERVER - параметры, получаемые при запуске от ОС
   3. $\_ENV - переменные среды ОС
   4. $\_GET, $\_POST — параметры GET- и POST-запроса
   5. $\_FILES — сведения об отправленных методом
      * 1. POST файлах.
   6. $\_COOKIE - куки
   7. $\_REQUEST - состоит из элементов $\_GET, $POST, $COOKIE и $\_FILES
5. Поддержка ООП
   1. реализованы основные механизмы ООП: инкапсуляция, полиморфизм и наследование
   2. private (только внутри класса), public (по дефолту) и protected (в наследниках) поля
   3. final и abstract методы
   4. есть трейты, но нет множественного наследования
   5. объекты передаются по ссылке
   6. обращение к константам, стат. св-вам и методам - через “::”.

Вот некоторые из основных способов использования PHP в веб-приложениях:

1) Генерация динамических страниц: PHP позволяет создавать веб-страницы, которые могут генерироваться динамически на основе данных из баз данных, файлов или других источников.

2) Обработка форм: PHP используется для обработки данных, отправленных пользователем через веб-формы.

3) Работа с базами данных: PHP интегрируется с большинством систем управления базами данных (например, MySQL, PostgreSQL, SQLite), позволяя вам выполнять запросы, извлекать, обновлять и удалять данные из баз данных.

4) Аутентификация и авторизация: PHP позволяет создавать системы аутентификации и авторизации, что обеспечивает контроль доступа к различным частям вашего веб-приложения.

5) Обработка файлов

2) Long Polling и Websockets зачем нужны, сходства и различия

Long Polling — это технология, которая позволяет получать данные о новых событиях с помощью «длинных запросов». Сервер получает запрос, но отправляет ответ на него не сразу, а лишь тогда, когда произойдет какое-либо событие (например, придёт новое сообщение), либо истечет заданное время ожидания.

1. Клиент запрашивает страницу у сервера, используя обычный http
2. Запрошенная страница выполняет JavaScript, который запрашивает файл от сервера.
3. Сервер НЕ реагирует на запрошенную информацию и ждет, пока не появится новой информации
4. Когда появляется новая информация, сервер отсылает ее клиенту
5. Клиент получает новую информацию и сразу отсылает другой запрос серверу, запуская процесс ожидания на нем снова.

WebSocket — протокол полнодуплексной связи поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и вебсервером в режиме реального времени и не требует постоянного переподключения.

1. Клиент запрашивает страницу у сервера, используя обычный http
2. Открывается соединение с сервером
3. Сервер и клиент могут посылать друг другу сообщения

WebSocket, в отличие от Long Polling, имеет полнодуплексное соединение (по нему может отправлять запрос не только сервер, но и клиент), также оно имеет одно это соединение на множество запросов, в то время как Long Polling открывает и закрывает соединение на каждый новый запрос. Long Polling хорошо подходит для сценариев, где нагрузка на сервер относитель но невелика и задержка ответа не критична. Он был популярен до появления WebSockets.

Long Polling хорошо работает когда сообщения приходят относительно редко. В отличие от Long Polling соединение в сокетах устанавливается один и поддерживается до закрытия. Такой вариант лучше подходит если посылки между сервером и клиентом очень частые, ведь не нужно все время открывать новое соединение после очередного сообщения. Применение: WebSockets идеально подходят для приложений, где низкая задержка критична, таких как чаты в реальном времени, онлайн-игры, мониторинг и др.

Сходства:

• Оба метода позволяют создавать веб-приложения в реальном времени.

• Они обеспечивают двусторонний обмен данными между клиентом и сервером.

Различия:

• Long Polling использует HTTP и имеет больше накладных расходов из-за частых запросов клиента, в то время как WebSockets использует одно постоянное соединение, что делает его более эффективным.

• WebSockets поддерживает множество одновременных соединений на одном соединении TCP/IP, в то время как Long Polling для каждого запроса создает отдельное соединение.

• WebSockets обычно быстрее и обеспечивает низкую задержку в реальном времени, в то время как Long Polling может иметь более высокую задержку из-за периодической необходимости клиента делать запросы.  
Вот некоторые плюсы и минусы Long Polling и WebSockets относительно друг друга:

Long Polling:

Плюсы:

1. Простота реализации: Long Polling легче реализовать, так как он может быть осуществлен с использованием стандартных HTTP-серверов и браузеров без необходимости создания специальных серверных или клиентских компонентов.
2. Хорошо подходит для небольших проектов: Long Polling может быть хорошим выбором для простых проектов или проектов с ограниченными ресурсами, где не требуется максимальная эффективность передачи данных.

Минусы:

1. Задержка и накладные расходы: Long Polling может создавать задержки, так как клиент должен постоянно устанавливать новые HTTP-соединения, и это может вызвать дополнительные накладные расходы на уровне сети и сервера.
2. Ограниченная поддержка для двусторонней связи: Long Polling, по сути, односторонний метод, и его реализация для двусторонней связи может быть более сложной и менее эффективной.

WebSockets:

Плюсы:

1. Мгновенная передача данных: WebSockets обеспечивают мгновенную передачу данных между клиентом и сервером без постоянного пересоединения, что делает их идеальным выбором для приложений реального времени.
2. Более низкая задержка: WebSockets обеспечивают более низкую задержку по сравнению с Long Polling, что делает их предпочтительным выбором для интерактивных приложений.
3. Двунаправленная связь: WebSockets позволяют как клиенту, так и серверу инициировать передачу данных, что полезно для обоих сторон.

Минусы:

1. Большая сложность реализации: WebSockets требуют специального серверного и клиентского кода для управления соединением, что может сделать реализацию более сложной и требовательной к ресурсам.
2. Большее потребление ресурсов: WebSockets могут потреблять больше ресурсов сервера и сети, чем Long Polling, из-за постоянного поддержания соединения.

● WebSocket — протокол полнодуплексной связи поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и веб-сервером в режиме реального времени.

● Позволяет серверу отправлять данные браузеру без дополнительного запроса со стороны клиента.

● Обмен данными ведётся через отдельное TCP-соединение.

● Поддерживается всеми современными браузерами (даже IE).

● Альтернатива — AJAX + Long Polling.

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

<%@ page import="java.util.LinkedList" %>  
<%@ page import="java.util.ArrayList" %>  
<%@ page import="java.util.Iterator" %>  
  
<html>  
<head><title>asd</title></head>  
<body>  
<%  
 LinkedList<String> id = (LinkedList<String>) session.getAttribute("id");  
 LinkedList<Long> time = (LinkedList<Long>) session.getAttribute("time");  
  
 if (id == null) {  
 id = new LinkedList<>();  
 time = new LinkedList<>();  
 session.setAttribute("id", id);  
 session.setAttribute("time", time);  
 }  
  
 if (!id.contains(session.getId())) {  
 id.add(session.getId());  
 time.add(System.currentTimeMillis());  
 }  
  
 if (id.size() != 0 && time.size() != 0 && id.size() == time.size()) {  
 ArrayList<Integer> indexes = new ArrayList<>();  
  
 for (int i = 0; i < time.size(); i++) {  
 if ((System.currentTimeMillis() - time.get(i)) >= 60 \* 1000)  
 indexes.add(i);  
 }  
  
 Iterator<Integer> iterator = indexes.iterator();  
 while (iterator.hasNext()) {  
 int i = iterator.next();  
 id.remove(i);  
 time.remove(i);  
 }  
  
 response.getWriter().println(id.size());  
 } else {  
 response.getWriter().println("error");  
 }  
%>  
</body>  
</html>

Страница для количества запросов, а не пользователей:  
<%@ page contentType=”text/html; charset=UTF-8” %>

<%@ page import=”java.util.Map, java.util.concurrent.ConcurrentHashMap” %>

<!DOCTYPE html>

<html>

<head>

<title>Request statistics for the last 60 seconds</title>

</head>

<body>

<h1>Request statistics for the last 60 seconds</h1>

<%

Map<String, Long> requestStats = (ConcurrentHashMap<String, Long>), request.getAttribute(”requestStats”);

long currentTime = System.currentTimeMillis();

int count = 0;

for (Long requestTime : requestStats.values()) {

if (currentTime - requestTime <= 60000) {

count++;

}

}

%>

<p><%= count %> requests have been executed in the last 60 seconds.</p>

</body>

### Билет 4

1) Библиотека JQuery, её назначение и основные API

2) Архитектура Model 1 и Model 2 - особенности, отличия, сферы применения

3) Шаблон и код инициализации контекста Thymeleaf, формирующие HTML-страницу, показывающую текущие курсы валют (доллара и евро) относительно рубля, динамику их изменения за последнюю торговую сессию на валютной бирже

1) 1 - jQuery - библиотека js, помогающая легко получить доступ к любому элементу DOM и манипулировать ими, предоставляет API для работы с AJAX.

jQuery включается в страницу как внешний файл.

<script src="jquery-2.2.2.min.js">

Вся работа с jQuery ведется с помощью функции $. Работу с jQuery можно разделить на 2 типа:

* Получение jQuery-объекта с помощью функции $().
* Вызов глобальных методов у объекта $.

Основные назначения jQuery:

1. Выбор элементов: jQuery предоставляет мощные селекторы CSS, которые позволяют выбирать элементы на веб-странице. Например, $("p") выберет все абзацы на странице.
2. Манипуляция элементами: Вы можете изменять содержимое элементов, их атрибуты, стили и многое другое, используя методы jQuery. Например, $("p").text("Новый текст") изменит текст всех абзацев на странице.
3. jQuery упрощает выполнение AJAX-запросов, что позволяет загружать данные асинхронно с сервера без перезагрузки страницы.

Основные методы и функции в jQuery включают:

1. Выборка элементов:
   * $() или jQuery(): Селекторы, позволяющие выбирать элементы на веб-странице. Например, $('div') выбирает все элементы div.
2. Манипуляция DOM:
   * addClass(), removeClass(), toggleClass(): Управление классами элементов.
   * append(), prepend(), after(), before(): Добавление содержимого внутрь или рядом с элементами.
   * html(), text(): Установка или получение HTML-содержимого элементов.
   * attr(), prop(), data(): Управление атрибутами и данными элементов.
3. События:
   * on(), off(): Привязка и удаление обработчиков событий.
   * click(), hover(), и другие методы для удобной обработки событий.
4. Анимация:
   * animate(): Создание анимаций для элементов.
   * fadeIn(), fadeOut(), slideDown(), slideUp(): Анимация видимости элементов.
5. AJAX:
   * $.ajax(), $.get(), $.post(): Выполнение асинхронных HTTP-запросов.
   * $.getJSON(), $.getScript(): Загрузка JSON данных и скриптов.
6. Утилиты:
   * $.each(): Перебор элементов в коллекции.
   * $.extend(): Объединение объектов.
   * $.trim(): Удаление пробелов из строки.

2) Архитектура Model 1 и Model 2 - особенности, отличия, сферы применения

Веб приложение можно разбить на 3 зоны ответственности: Модель (Model), Представление (View) и Контроллер (Controller). Модель - это объектная модель бизнес-логики и часть взаимодействия с базами данных. Контроллер - то, что расшифровывает запрос, вызывает соответствующие ф-ции модели и отдает оформленные данные представлению.

Model 1 Предназначена для проектирования небольших веб-приложений с довольно простой бизнес логикой. За обработку данных и представления отвечает один и тот же компонент (JSP, Servlet и т. п.). Использование данной модели оправдано, когда необходимо быстро сделать небольшое приложение и выгоднее объединить данные и представление в кучу, чтобы не тратить много времени на проектирование обособленных друг от друга частей и их связывание.

● Предназначена для проектирования приложений небольшого масштаба и сложности.

● За обработку данных и представления отвечает *один и тот же* компонент (сервлет или JSP).

Изображение выглядит как текст, диаграмма, зарисовка, снимок экрана

Автоматически созданное описание

Model 2 сохраняет это разделение. Model 1 же соединяет Представление и Контроллер в одном компоненте (чаще JSP (общается с моделью через бины), может быть сервлет).

Когда масштаб приложения растет удобнее использовать Model 2. В Model 2 за представление и логику отвечают разные компоненты (разные сервлеты, разные JSP и т. п.).  Model 2 использует шаблон MVC - Model-View-Contoller.

● Предназначена для проектирования достаточно сложных веб-приложений.

● За обработку и представление данных отвечают *разные* компоненты (сервлеты и JSP).

Изображение выглядит как текст, диаграмма, План, линия

Автоматически созданное описание

Таким образом, Model 1 проще разрабатывать, однако есть недостатки:

1. Сложнее поддерживать приложение ввиду смешения уровней представления и контроллера
2. сеть компонентов децентрализована, так что изменение URI компонента нарушит цепочку вызовов, это будет сложно отследить

Но и у Model 2 есть недостаток - компоненты Model 1, часто представляющие JSP, можно менять без пересборки приложения.

Преимущества: простая, быстрая разработка, подходящая для мелкомасштабной разработки

Недостатки: бизнес-логика и логика представления не абстрагируются и не разделяются на странице JSP, а нагрузка JSP слишком велика. Следовательно, это не способствует повторному использованию и модификации бизнеса прикладной системы, и его нелегко поддерживать.

Преимущества: он обладает преимуществами компонентной компоновки, облегчая разработку и управление крупномасштабными системами. Распределение обязанностей ясно.

Недостатки: не подходит для разработки небольших проектов. Сложность разработки: Model 2 может потребовать больше усилий при начальной разработке из-за необходимости четкого разделения обязанностей между компонентами.

3) Шаблон и код инициализации контекста Thymeleaf, формирующие HTML-страницу, показывающую текущие курсы валют (доллара и евро) относительно рубля, динамику их изменения за последнюю торговую сессию на валютной бирже

<!DOCTYPE html>

<html lang="en" xmlns:th="http://www.thymeleaf.org">

<head>

<meta charset="UTF-8">

<title>билет</title>

</head>

<body>

<h1>Currency Exchange Rates</h1>

<table>

<tr>

<th>Валюта</th>

<th>Курс</th>

<th>Изменение</th>

</tr>

<tr th:each="currency : ${currencies}">

<td th:text="${currency.name}"></td>

<td th:text="${currency.rate}"></td>

<td th:text="${currency.change}"></td>

</tr>

</table>

</body>

</html>

@Controller

public class CurrencyController {

@Autowired

CurrencyService currencyService;

@GetMapping("/currency")

public String getCurrency(Model model) {

List<Currency> currencies = currencyService.getCurrencies();

model.addAttribute("currencies", currencies);

return "currency";

}

}

### Билет 5 (12)

1) Jakarta EE архитектура, компоненты, контейнер

Jakarta EE - это переименованная старшая версия Java EE. Приложения состоят из компонентов под управлением контейнеров.

Используются принципы

1. Inversion of Control (IoC). Суть в особой роли контейнера - он управляет жизненным циклом компонента (можно вешать обработчики) и обеспечивает взаимодействие между компонентами (первый компонент помещает данные в контейнер, второй получает их по определенным приметам).
2. Contexts & Dependency Injection (CDI). Суть в том, что мы зависим от API контейнера через аннотации, которые описывают компоненты, чтобы контейнер понял, что с ними делать. В противном случае нам пришлось бы делать пометки через реализацию интерфейсов и использование конкретных методов контейнера.

Jakarta EE (ранее известная как Java Platform, Enterprise Edition или Java EE) - это набор спецификаций и стандартов для разработки и выполнения распределенных приложений Архитектура Jakarta EE: Архитектура Jakarta EE включает в себя следующие ключевые компоненты

• Клиенты

• Компоненты приложения такие как сервлеты, EJB (Enterprise JavaBeans), JMS (Java Message Service), CDI (Contexts and Dependency Injection), и другие

• Серверный контейнер: Jakarta EE приложение работает внутри серверного контейнера, который предоставляет среду выполнения для компонентов приложения. Серверный контейнер обеспечивает множество служб, таких как управление жизненным циклом компонентов, безопасность, управление транзакциями, пулы соединений и другие.

• Инфраструктура: Jakarta EE также предоставляет инфраструктуру, включая JDBC (Java

Database Connectivity) для доступа к базам данных, JNDI (Java Naming and Directory Interface) для доступа к ресурсам и EIS (Enterprise Information Systems), JTA (Java Transaction API) для управления транзакциями.

Компоненты Jakarta EE:

• Сервлеты

• EJB (Enterprise JavaBeans)

• JMS (Java Message Service)

• CDI (Contexts and Dependency Injection)

• JPA (Java Persistence API)

• JSP (JavaServer Pages)

• JCA (Java Connector Architecture)

2) Синтаксис шаблонов, модель данных FreeMarker

Описано выше в билете 2

1. [заменить все гиперссылки на текстовые поля со значением ссылок.](https://pastebin.com/CyrCWwRh)

<html>

<head>

<script>

function replaceLinks() {

let linksEls = document.getElementsByTagName("a");

while (linksEls.length != 0) {

let e = linksEls[0];

let text = document.createElement("p");

text.innerText = e.getAttribute("href");

e.parentNode.replaceChild(text, e);

}

}

</script>

</head>

<body>

<a href="https://www.google.com">Hello</a><br>

<a href="https://www.google.com">Another hello</a><br>

window.onload = replaceLinks;

</body>

</html>

### Билет 6 (15)

1) конфигурация PHP, способы интеграции PHP с веб-приложением

Интерпретатор PHP - это программа, которая получает запрос и отдает некоторый html ответ, направлять ей запрос с портов должен некоторый сервер. Можно использовать PHP через CGI и FastCGI, однако тут есть свои минусы. Апаче же имеет свой апи взаимодействия с PHP, поэтому, встроенный в веб-сервер, php интерпретатор работает быстрее.

Конфигурация PHP:

Настройки PHP хранятся в файле php.ini. Вы можете найти этот файл в зависимости от вашей операционной системы и установки PHP. В этом файле различные параметры PHP, такие как максимальный размер загружаемых файлов, настройки даты и времени, уровень вывода ошибок и многое другое.

Способы интеграции:

• Использование модуля PHP: Наиболее распространенным способом является интеграция PHP с веб-сервером через модуль PHP. Например, для Apache, вы можете использовать модуль mod\_php, а для Nginx - модуль ngx\_http\_php\_module. Это позволяет обрабатывать PHP-скрипты непосредственно внутри веб-сервера.

• Использование FastCGI: FastCGI - это более современный и гибкий способ интеграции PHP. Вы можете настроить PHP как FastCGI-процесс, который взаимодействует с веб-сервером через протокол FastCGI. Для Apache, вы можете использовать модуль mod\_fcgid или mod\_fastcgi. Для Nginx, используйте FastCGI-прокси.

• Docker: Если вы используете контейнеризацию с Docker, вы можете создать контейнер, включающий в себя PHP и веб-сервер (например, Apache или Nginx), и настроить их взаимодействие

---

PHP-интерпретатор читает содержимое файла .php переданного ему от сервера и обрабатывает код от . Всё, что за пределами (HTML-теги и текст) PHP-интерпретатор возвращает серверу так как есть.

Конфигурационные параметры хранятся в файле php.ini. Можно подключать дополнительные модули, расширяющие возможности языка (например, добавляющие поддержку взаимодействия с СУБД)

Способы использования интерпретатора PHP:

- С помощью SAPI / ISAPI (например, mod\_php для Apache).

- С помощью CGI / FastCGI.

- Через интерфейс командной строки.

2) ServletContext - что это и для чего применяется

ServletContext - это интерфейс, через который идет взаимодействие с контейнером. У всех сервлетов приложения он общий (если приложение не распределенное), поэтому в него можно помещать общую для всех сервлетов информацию вызовами get/setAttribute, через него можно вести логи, а также есть возможность получения параметров конфигурации сервера

Применение:

• Хранение и обмен данных: ServletContext может использоваться для хранения данных, которые должны быть доступны всем компонентам приложения. Это позволяет разным частям приложения обмениваться информацией.

• Конфигурация приложения: Вы можете использовать ServletContext для хранения параметров конфигурации приложения, таких как параметры базы данных, настройки безопасности и другие.

• Загрузка ресурсов: ServletContext предоставляет методы для доступа к ресурсам внутри приложения, таким как файлы, изображения и другие ресурсы, без прямого обращения к файловой системе.

• Управление сессиями

• Логирование и слежение

• Взаимодействие с другими компонентами

3) [Написать JSP страничку, которая отображает корзину покупателя. Один объект этой корзины представляет класс ShoppingItem, который содержит что-то там о товаре. Коллекция этих объектов хранится в отдельном managed bean.](http://pastebin.com/pBA7f4Zt)

@ManagedBean

public class ShoppingCartBean {

private List<ShoppingItem> items = new ArrayList<>();

public List<ShoppingItem> getItems() {

return items;

}

public void addItem(ShoppingItem item) {

items.add(item);

}

}

JSP:

<\%@ page language=”java” contentType=”text/html; charset=UTF-8” pageEncoding=”UTF-8” \%>

<\%@ taglib prefix=”c” uri=”http://java.sun.com/jsp/jstl/core” \%>

<!DOCTYPE html>

<html>

<head>

<title>Shopping Cart</title>

</head>

<body>

<h1>Shopping Cart</h1>

<table>

<tr>

<th>Name</th>

<th>Price</th>

</tr>

<c:forEach items=”${shoppingCartBean.items}” var=”item”>

<tr>

<td>${item.name}</td>

<td>${item.price}</td>

</tr>

</c:forEach>

</table>

</body>

### Билет 7 (17)

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

LESS, SASS и SCSS - это препроцессоры CSS, которые предоставляют дополнительные функции и улучшают структуру и организацию стилей в сравнении с обычным CSS.

LESS: Синтаксис: LESS использует синтаксис, который напоминает обычный CSS, но добавляетнекоторые расширения, такие как переменные, миксины, операторы и вложенность.

Взаимодействие с браузером: LESS-файлы требуют компиляции в обычный CSS перед отправкой на клиентский браузер. Это обычно делается с использованием компиляторов LESS, которые могут быть интегрированы в сборочный процесс.

SASS (Syntactically Awesome Stylesheets):

Синтаксис: SASS предлагает два синтаксиса: SASS (с более компактным синтаксисом, без фигурных скобок и точек с запятой) и SCSS (синтаксис, близкий к обычному CSS с фигурными скобками и точками с запятой). Оба синтаксиса поддерживают переменные, миксины и другие расширения.

Взаимодействие с браузером: SASS-файлы также требуют предварительной компиляции в CSS перед использованием в браузере. Существует множество инструментов для компиляции SASS, включая командную строку и среды разработки

SCSS (Sassy CSS):

Синтаксис: SCSS является одним из синтаксических вариантов SASS, который ближе к синтаксису CSS, что делает его более привычным для разработчиков CSS. SCSS также поддерживает все функции SASS, включая переменные и миксины.

Взаимодействие с браузером: Как и в случае с LESS и SASS, SCSS-файлы должны быть предварительно скомпилированы в CSS перед отправкой на клиентский браузер.

Фишки языков:

1. константы:
   1. в less @название: значение
   2. в sass и scss $название: значение
2. примеси - некий кусочек кода с именем, который потом по этому имени можно куда-то встроить.
   1. в less внутри блока расположить идентификатор(). например, .name();
   2. в scss @mixin название {св-ва} (в sass нет фигурных скобок), включить миксин можно так: @include название
3. вложенные стили - можно делать иерархическую структуру правил (делается вложением внутрь элементов согласно базовому синтаксису)
4. Импорт стилей. Можно делать в разных файлах иерархические вложенные друг в друга таблички стилей. Позволяет их грамотно структурировать. Во всех языках можно использовать синтаксис @import “название”
5. Наследование. Важно понимать, что это не классическое наследование, в том смысле, что у нас есть правило и оно будет расширено.Это же наследование на уровне синтаксического сахара. То есть фактически при трансляции SCSS в CSS правило .message будет встроено, скопировано внутрь .success. На уровне браузера это все равно будет плоский CSSsas
   1. в less: list ids :extend(.first, #second, …) {}
   2. в sass и scss: @extend .smth
6. Арифметика. В css есть только основы, которых часто не хватает. Можно применять повсюду.
7. Логика

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

JSTL - расширение JSP, добавляющее библиотеку тегов, позволяющие решать типовые задачи.

Примеры задач:

● Условная обработка.

● Создание циклов, вывод массивов / коллекций.

● Поддержка интернационализации.

● Рекомендуется использовать их вместе с EL вместо скриптлетов.

Для подключения библиотеки тегов **JSTL** используются следующие выражения:

// Основные теги создания циклов, определения условий, вывода информации на страницу и т. д.

<%@ taglib prefix="c"

uri="http://java.sun.com/jsp/jstl/core" %>

// Теги для работы с XML-документами

<%@ taglib prefix="x"

uri="http://java.sun.com/jsp/jstl/xml" %>

// Теги для работы с базами данных

<%@ taglib prefix="s"

uri="http://java.sun.com/jsp/jstl/sql" %>

// Теги для форматирования и интернационализации информации (i10n и i18n)

<%@ taglib prefix="f"

uri="http://java.sun.com/jsp/jstl/fmt" %>

JSTL основные теги:

• Контрольные теги (<c:if>, <c:choose>, <c:forEach>): Позволяют выполнять условные операции и итерации.

• Теги вывода (<c:out>, <c:import>, <c:param>): Используются для вывода данных на странице и включения других ресурсов.

• Теги для работы с коллекциями (<c:forEach>, <c:forTokens>, <c:forEach>): Позволяют выполнять итерации по коллекциям данных.

• Теги форматирования (<fmt:formatDate>, <fmt:formatNumber>): Используются для форматирования дат, чисел и валют.

3) Написать структуру HTTP запроса, отправляющего логин и пароль пользователя

GET <https://site.ru/signin?login=Petya%20Vasechkin&password=qwerty123> HTTP/1.1  
Host: site.ru  
Referer: <https://site.ru/signin>  
Cookie: income=1

POST /login HTTP/1.1

Host: example.com

Content-Type: application/x-www-form-urlencoded

Content-Length: 27

username=johndoe&password=secret

3) Написать сервлет, который принимает из http запроса параметр name и выводит его. Если параметр не обнаружен то вывести Anonymous user

public class MyServlet extends HttpServlet {

public void service(HttpServletRequest req, HttpServletResponse res) {

var out = res.getWriter();

var param = req.getParameter(“name”);

if (param != null) {

out.println(req.getParameter(“name”));

else

out.println(“Anonymous user”);

}

}

### Билет 8

1) SuperAgent - это API для реализации AJAX при помощи простых методов.

SuperAgent умеет много полезных вещей: работать с CORS запросами, посылать form-data, задать кол-во попыток отправки запроса, сделать редирект.

request

.post('/api/pet')

.send({ name: 'Manny', species: 'cat' })

.set('X-API-Key', 'foobar')

.set('Accept', 'application/json')

.then(res => {

alert('yay got ' + JSON.stringify(res.body));

});

.catch(err => {

// err.message, err.response

});

Здесь методом post мы указываем, что хотим отправить post-запрос на ресурс указанный в аргументе (по аналогии есть методы get, head и т. д.). Далее методом send мы отправляем непосредственно тело запроса, методом set назначаем нужные HTTP-заголовки. Метод then принимает функцию, которая будет вызвана при успешном получении результата. Также существует множество других методов, например catch, позволяющий нам отловить и обработать ошибку в случае возникновения таковой. Есть метод serialize() позволяющий автоматически сериализовать объекты в JSON, XML и так далее. Есть метод retry() позволяющий повторно отправить запрос нужное количество раз при возникновении ошибки. В общем это удобный и многофункциональный API который избавляет нас от огромного количество стандартного AJAX-кода через XMLHttpRequest.

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

Шаблон проектирования (Design Pattern):

• Описание: Шаблон проектирования - это повторяемое решение для типичных проблем в рамках конкретного контекста разработки. Это абстрактный шаблон, который может быть адаптирован и применен к разным ситуациям. Шаблоны проектирования предоставляют практики и структуры для создания гибких и расширяемых систем.

• Применение: Шаблоны проектирования применяются на уровне кода. Они предлагают решения для конкретных проблем, таких как создание объектов, управление зависимостями, реализация паттернов поведения и др. Примеры включают Singleton, Factory, Observer, Strategy и многие другие.

• Пример: Паттерн Singleton обеспечивает, что в системе существует только один экземпляр класса и предоставляет глобальную точку доступа к этому экземпляру.

Архитектурный шаблон (Architectural Pattern):

• Описание: Архитектурный шаблон - это повторяемая архитектурная конструкция для организации компонентов и подсистем в рамках приложения. Он определяет структуру приложения и способы взаимодействия между его частями. Архитектурные шаблоны обычно связаны с высокоуровневой структурой системы.

• Применение: Архитектурные шаблоны применяются на более высоком уровне и охватывают более широкие аспекты системы. Они определяют общую структуру, включая слои, компоненты и взаимодействие между ними. Примеры включают клиент-сервер, MVC (Model-ViewController), микросервисную архитектуру и др.

• Пример: Архитектурный шаблон MVC (Model-View-Controller) определяет разделение системы на три основных компонента: модель (хранение данных), представление (отображение данных пользователю) и контроллер (управление взаимодействием между моделью и представлением).

Отличия между шаблонами проектирования и архитектурными шаблонами:

• Уровень абстракции: Шаблоны проектирования применяются на уровне кода и предлагают решения для конкретных задач внутри приложения. Архитектурные шаблоны определяют структуру и организацию приложения на более высоком уровне.

• Область применения: Шаблоны проектирования применяются для решения локальных задач, таких как создание объектов, управление состоянием и т. д. Архитектурные шаблоны определяют общую структуру приложения и взаимодействие между его компонентами.

• Цель: Шаблоны проектирования помогают решать конкретные проблемы внутри кода, обеспечивая его гибкость и повторное использование. Архитектурные шаблоны определяют общую структуру системы и облегчают разделение обязанностей между компонентами.

3) Написать AJAX запрос который получает JSON и выводит его элементы

<!DOCTYPE html>

<html>

<head>

<title>AJAX JSON Request</title>

</head>

<body>

<h1>JSON Data</h1>

<div id=”json-data”></div>

<script>

fetch(’https://jsonplaceholder.typicode.com/posts/1’)

.then(response => response.data)

.then(data => {

const jsonDataElement = document.getElementById(’json-data’);

jsonDataElement.innerHTML = JSON.stringify(data, null, 4);

})

.catch(error => {

console.error(’Error: ’ + error);

});

</script>

</body>

### Билет 9

1) Как сервлеты обрабатывают HTTP запрос

1 - Для обработки запроса в HttpServlet определен ряд методов, которые мы можем переопределить в классе сервлета:

* **doGet**: обрабатывает запросы GET (получение данных)
* **doPost**: обрабатывает запросы POST (отправка данных)
* **doPut**: обрабатывает запросы PUT (отправка данных для изменения)
* **doDelete**: обрабатывает запросы DELETE (удаление данных)
* **doHead**: обрабатывает запросы HEAD

Каждый метод обрабатывает определенный тип запросов HTTP, и мы можем определить все эти методы, но, зачастую, работа идет в основном с методами doGet и doPost.

1. Когда движок сервлетов создает объект сервлета, у сервлета вызывается метод **init()**.

Этот метод вызывается только один раз - при создании сервлета. Мы можем переопределить этот метод, чтобы определить в нем некоторую логику инициализации.

1. Когда к сервлету приходит запрос, движок сервлетов вызывает метод **service()** сервлета. А этот метод, исходя из типа запроса (GET, POST, PUT и т.д.) решает, какому методу сервлета (doGet, doPost и т.д.) обрабатывать этот запрос.

Этот метод также можно переопределить, однако в этом нет смысла. В реальности для обработки запроса переопределяются методы onGet, onPost и т.д., которые обрабатывают конкретные типы запросов.

1. Если объект сервлета долгое время не используется (к нему нет никаких запросов), или если происходит завершение работы движка сервлетов, то движок сервлетов выгружает из памяти все созданные экземпляры сервлетов. Однако до выгрузки сервлета из памяти у сервлета вызывается метод **destroy()**.

2) Многоуровневая архитектура: элементы, зачем нужна

Этот архитектурный подход разделяет комплекс ПО на уровни. Архитектура может иметь n уровней, разделяющих ответственности между поставщиком данных и потребителем.

**Двухуровневая система**

Изображение выглядит как снимок экрана, текст, Мультимедийное программное обеспечение, дизайн

Автоматически созданное описание

Так выглядит двухуровневая архитектура

Эта система состоит из двух физических машин в качестве сервера и клиента. Она обеспечивает изоляцию операций управления данными, обработки данных и операций представления.

* **Клиент**содержит слои презентации, бизнес-логики и передачи данных.
* **Сервер**включает хранилища и базы данных.

**Трехуровневая и n-уровневая системы**



Так выглядит трехуровневая архитектура

Такие архитектуры обладают высокой масштабируемостью как по горизонтали, так и по вертикали. Реализация n-уровневой системы, как правило, обходится дороже, но обеспечивает высокую производительность. Поэтому она обычно применяется в крупных и комплексных программных решениях.

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

<#list grades?sort\_by("time") as grade>

<tr>

<td>${grade.student}</td>

<td>${grade.time}</td>

<td>${grade.value}</td>

</tr>

</#list>

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Страница с оценками студентов</title>

</head>

<body>

<h1>Оценки студентов</h1>

<#assign myObject = {

'item1': {'name': One', 'grade': 1, 'time': '2023-10-30 19:00:00'},

'item2': {'name': Two', 'grade': 2, 'time': '2023-10-30 10:30:00'},

'item3': {'name': Three', 'grade': 3, 'time': '2023-10-30 11:15:00'}

}/>

<#macro displayObjects>

<ul>

<#list myObject?values?sort\_by("time") as item>

<li>${item.name} - Grade: ${item.grade} - Time: ${item.time}</li>

</#list>

</ul>

</#macro>

<@displayObjects />

</body>

</html>

### Билет 10

1) Rest и RPC

REST (Representational State Transfer) и RPC (Remote Procedure Call) представляют два различных подхода к взаимодействию между компонентами программного обеспечения, такими как клиенты и серверы. Давайте рассмотрим их основные характеристики:

1. REST (Representational State Transfer):
   * REST - это архитектурный стиль для проектирования сетевых приложений, основанный на принципах, которые способствуют масштабируемости, удобству использования и управляемости.
   * REST использует стандартные HTTP методы, такие как GET, POST, PUT и DELETE, для выполнения операций над ресурсами, представленными в виде URL (Uniform Resource Locator).
   * Каждый ресурс имеет свое уникальное URI, и клиенты могут взаимодействовать с ресурсами с использованием HTTP запросов.
   * REST подчеркивает использование статуса и представления (например, JSON или XML) для обмена данными между клиентами и серверами.
   * RESTful API часто считают более гибкими и расширяемыми, чем традиционные RPC-подобные подходы.
2. RPC (Remote Procedure Call):
   * RPC - это метод вызова удаленных процедур, который позволяет клиентскому коду вызывать функции или методы на удаленном сервере, как если бы они были локальными.
   * RPC скрывает детали взаимодействия с сетью и позволяет клиенту вызывать удаленные методы, как если бы они были частью локального API.
   * В RPC-подобных системах обычно создаются клиентские и серверные прокси, которые предоставляют интерфейсы для вызова удаленных методов.
   * Протоколы, такие как gRPC, используются для реализации RPC-подобных систем и обычно сериализуют данные в бинарном формате.

REST - публичный веб (сайты магазинов, блоги)

RPC - информационные, интерпрайзовые системы.

---

*Representational State Transfer* (передача состояния

представления) – подход к архитектуре сетевых протоколов,

обеспечивающих доступ к информационным ресурсам.

● Основные концепции:

● Данные должны передаваться в виде небольшого числа стандартных форматов (HTML, XML, JSON).

● Сетевой протокол должен поддерживать кеширование, не должен зависеть от сетевого слоя, не должен сохранять информацию о состоянии между парами «запрос-ответ».

● Антипод REST – подход, основанный на вызове удаленных процедур (*Remote Procedure Call* – *RPC*).

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

RequestDispatcher - интерфейс, предоставляющий методы для делегирования запросов другим компонентам (сервлетам, JSP-страницам и т.д).

Получить реализацию этого интерфейса можно через контексты сервлета или запроса:

(**context** или **request**)**.getRequestDispatcher(“resource-url”)**

В первом случае, URL ресурса должен быть абсолютным - начиная с корня приложения, во втором случае URL может быть как абсолютным, так и относительным.

Интерфейс RequestDispatcher определяет два метода для перенаправления запросов:

forward(ServletRequest request, ServletResponse response): Этот метод используется для перенаправления запроса от одного сервлета к другому или к другому ресурсу в рамках текущего веб-приложения. include(ServletRequest request, ServletResponse response): Этот метод также используется для перенаправления запроса, но в отличие от forward, он не завершает выполнение текущего сервлета. Вместо этого результат обработки включенного ресурса (например, JSP-страницы) добавляется к текущему ответу. Это позволяет включать содержимое других ресурсов в текущий ответ.

3) Написать правило на CSS, что у всех посещённых ссылок будет жёлтый фон, кроме тех, кто лежит в news

a:visited {

background-color: yellow;

}

.news a:visited {

background-color: initial;

}

### Билет 11

1) структура HTML документа

HTML документ состоит из элементов, которые могут:

1. Быть пустыми: <br>
2. Иметь тело: <body>тело</body>
3. Иметь атрибуты: <a href=”...”>
4. Иметь вложенные элементы:

<div>

<p>Текст</p>

</div>

Сам HTML документ может иметь объявление версии HTML, но сейчас это необязательно:  
 <!DOCTYPE HTML>

База - это парный тег <html>, в котором заключаются парные теги <head> и <body>. <head> содержит мета-информацию страницы, <body> - содержание документа, которое отрисовывается браузером (если не указан соответствующий атрибут.

Документ состоит из элементов, начало и конец которых обозначаются тегами. Некоторые теги не могут содержать текст (например, <br> — перенос строки, <img> — картинка, <input> — элемент ввода в форме). Их не нужно закрывать:

В HTML5 введены семантические теги <header>, <footer>, <section>, которые аналогичны <div>, но указывают на логическую структуру.

2) Сервлеты - особенности, преимущества и недостатки относительно CGI, FastCGI

Пакеты javax.servlet и javax.servlet.http обеспечивают интерфейсы и классы для создания сервлетов.

**Cервлет** — это Java-класс, который наследуется обычно от класса HttpServlet и переопределяет часть методов:

* doGet — если мы хотим, чтобы сервлет реагировал на GET запрос.
* doPost — если мы хотим, чтобы сервлет реагировал на POST запрос.
* doPut, doDelete — если мы хотим, чтобы сервлет реагировал на PUT и DELETE запрос (есть и такие в HTTP). Эти методы реализуются крайне редко, т.к. сами команды тоже очень редко встречаются.
* init, destroy — для управления ресурсами в момент создания сервлета и в момент его уничтожения.

Сервлет vs CGI

• Сервлеты запускаются в одном процессе (HTTP-сервер с дополнительными функциями, который называется Serlet Container), и они существуют до тех пор, пока этот процесс существует.

• CGI каждый раз создает новый экземпляр процесса для обслуживания запроса. Это убийца перфоманса.

• Поскольку для каждого запроса существует новый процесс, это означает, что CGI не может агрегировать данные из нескольких запросов в памяти.

Сервлет vs FastCGI

• При использовании сервлетов веб-сервер может напрямую вызвать приложение.

Преимущества сервлетов по сравнению с CGI и FastCGI:

• Производительность: Сервлеты более эффективны и быстрее, чем CGI, так как они работают в рамках JVM и переиспользуются.

• Многозадачность: Сервлеты поддерживают многозадачность, что позволяет одновременно обрабатывать множество запросов.

• Более легкая интеграция: Сервлеты легко интегрируются с другими Java-технологиями и фреймворками.

• Более низкая нагрузка на сервер: Переиспользование сервлетов уменьшает нагрузку на сервер по сравнению с созданием новых процессов CGI для каждого запроса.

Недостатки сервлетов по сравнению с CGI и FastCGI:

• Сложность разработки: Разработка сервлетов может быть более сложной и требовательной к навыкам разработчика, чем создание CGI-скриптов.

• Язык программирования: Сервлеты требуют знания Java, что может ограничить выбор языка программирования по сравнению с CGI, который поддерживает различные языки.

• Настройка и развертывание: Настройка сервера для работы с сервлетами может потребовать дополнительных шагов по сравнению с CGI.

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

.news img:hover {

border: 1px solid blue;

}

### Билет 13

1) Цикл жизни сервлета



За вызовы методов цикла жизни отвечает веб-контейнер

* В случае отсутствия сервлета в контейнере.

1. Класс сервлета загружается контейнером.
2. Контейнер создает экземпляр класса сервлета.
3. Контейнер вызывает метод init().

* Обслуживание клиентского запроса. Каждый запрос обрабатывается в своем отдельном потоке. Контейнер вызывает метод service() для каждого запроса. Этот метод определяет тип пришедшего запроса и распределяет его в соответствующий этому типу метод для обработки запроса. Разработчик сервлета должен предоставить реализацию для этих методов. Если поступил запрос, метод для которого не реализован, вызывается метод родительского класса и обычно завершается возвращением ошибки инициатору запроса.
* В случае если контейнеру необходимо удалить сервлет, он вызывает метод destroy(), который снимает сервлет из эксплуатации. Подобно методу init(), этот метод тоже вызывается единожды за весь цикл сервлета.

2) Диалекты и процессоры Thymeleaf и стандартный диалект

Thymeleaf – это компилирующий обработчик шаблонов, написанный на Java.

Особенности Thymeleaf:

● «Выходной» формат – XML, XHTML и HTML5. Также поддерживаются JS, CSS и Plain Text.

● Не привязан к Servlet API – может работать как «онлайн», так и «оффлайн».

● Построен на модульной системе. Модули называются диалектами (dialects).

● Поддержка возможностей i18n и l10n.

● Есть встроенный кеш скомпилированных шаблонов.

Процессоры и диалекты

● Thymeleaf – модульный движок. Модуль Thymeleaf называется диалектом (dialect).

● Диалект состоит из одного или нескольких процессоров (processor).

● Процессор – объект, который применяет некоторую логику к формируемому на основе шаблона артефакту.

● «Из коробки» Thymeleaf содержит стандартный диалект (Standard Dialect), которого достаточно для решения большинства типовых задач.

1. Стандартный диалект (Standard Dialect): Этот диалект предоставляет базовую функциональность для создания шаблонов. Он включает в себя выражения для вставки данных, итерации по коллекциям, условные операторы и многое другое.

Однако Thymeleaf также поддерживает различные диалекты, которые предоставляют дополнительные функции и синтаксис для конкретных задач. Например, Thymeleaf Layout Dialect используется для создания макетов и шаблонов, Thymeleaf Spring Dialect облегчает интеграцию Thymeleaf с фреймворком Spring, Thymeleaf Security Dialect позволяет контролировать доступ к элементам страницы на основе прав доступа и т. д.

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

var objects = $(“div.lecture”);

var wc = 0;

for (let el of objects) {

wc += el.innerText.match(/де-факто/g)z.length;

}

console.log(wc - 1);

—---------------------------------------------------

$(“div.lecture : contains(‘де-факто’)”).length;

### Билет 14

1) AJAX и DHTML - описание, сходства и различия

Dynamic HTML — способ создания интерактивного веб-сайта, использующий

сочетание:

● статичного языка разметки HTML;

● выполняемого на стороне клиента скриптового языка JavaScript;

● CSS (каскадных таблиц стилей);

● DOM (объектной модели документа).

Описание: AJAX - это набор технологий и подходов, который позволяет веб-страницам обмениваться данными с сервером без перезагрузки всей страницы. Он использует JavaScript для асинхронных HTTP-запросов, обработки данных и обновления содержания страницы.

Сходства:

1. Интерактивность: И Ajax, и DHTML позволяют создавать интерактивные веб-страницы, которые могут обновляться без полной перезагрузки страницы.
2. Использование JavaScript: Обе технологии требуют использования JavaScript для управления динамическими элементами на странице.
3. Манипуляция DOM: И Ajax, и DHTML позволяют изменять DOM (Document Object Model) страницы, что позволяет динамически изменять содержимое и структуру страницы.

Различия:

1. Фокус:
   * Ajax: Основное внимание в технологии Ajax уделяется асинхронным запросам к серверу. С помощью Ajax можно отправлять запросы на сервер и обновлять содержимое страницы без перезагрузки.
   * DHTML: DHTML сконцентрирован на динамическом изменении элементов на странице без необходимости отправки запросов на сервер. Это включает в себя анимацию, изменение стилей, и т.д.
2. Загрузка данных:
   * Ajax: Ajax позволяет загружать данные с сервера асинхронно. Это полезно, например, при динамическом обновлении содержимого страницы, загрузке данных из базы данных или обмене данными с сервером без перезагрузки страницы.
   * DHTML: DHTML не фокусируется на загрузке данных с сервера. Его цель - создание динамических эффектов и анимаций на клиентской стороне.

Одно из основных различий между стандартным HTML и DHTML заключается в том, что после загрузки страницы в стандартном HTML она не изменится, пока не получит другой запрос от сервера, однако в DHTML элементам разрешено изменять в любое время без необходимости возврата к серверу.

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

Как мы знаем, сами запросы выделяются в отдельные потоки, однако вызов обработки сервлета (service) совершается от одного объекта сервлета (по крайней мере такое может быть). И тогда, если используется общий ресурс, может наблюдаться состояние гонки, которое можно разрешить двумя способами:

1. стандартными методами - блоками синхронизации, методами синхронизации, использованием volatile atomic переменных и т.п.
2. установкой директивы isTreadSafe в значение false, чтобы запретить параллельную обработку запросов

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

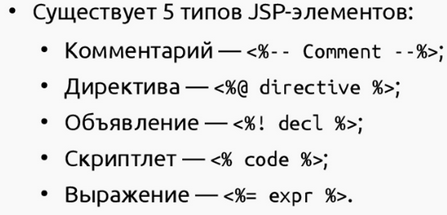
function insertNyancat() {

document.querySelector('div.nyan').innerHTML = '<img src="http://www.example.com/nyancat.gif">';

}

### Билет 16

1) JSP - элементы



html комментарий <!-- --> будет виден на клиенте

java комментарий <% /\*\*/ %> будет виден в сген. сервлете

jsp комментарий <%-- --%> будет виден только в jsp-файле

Директивы. Для передачи аргументов и параметров контейнеру. <%@ атрибут="значение" %>

Директива <%@ page %>:

<%@ page language=”java”contentType=”text/html; charset=UTF-8”%> - устанавливает язык (Java) и тип контента для JSP-страницы.

Директива <%@ include %>:

<%@ include file=”header.jsp”%> - включает содержимое другой JSP-страницы в текущую.

Объявления. Объявления используются для определения используемых в программе конструкций Java (задавать переменные, методы, внутренние классы).

<%! private int accessCount = 0; %>

Количество обращений к странице с момента загрузки сервера: <%= ++accessCount %>

Cкриптлеты. Скриплеты JSP дают возможность вставить любой код в метод сервлета, который будет создан при обработке страницы, позволяя использовать большинство конструкций Java.

<%

String queryData = request.getQueryString();

out.println("Дополнительные данные запроса: " + queryData);

%>

Выражения. Выражения JSP применяются для того, чтобы вставить значения Java непосредственно в вывод. Выражения Java вычисляются, конвертируются в строку и вставляются в страницу.

Текущее время: <%= new java.util.Date() %>

Имя вашего хоста: <%= request.getRemoteHost() %>

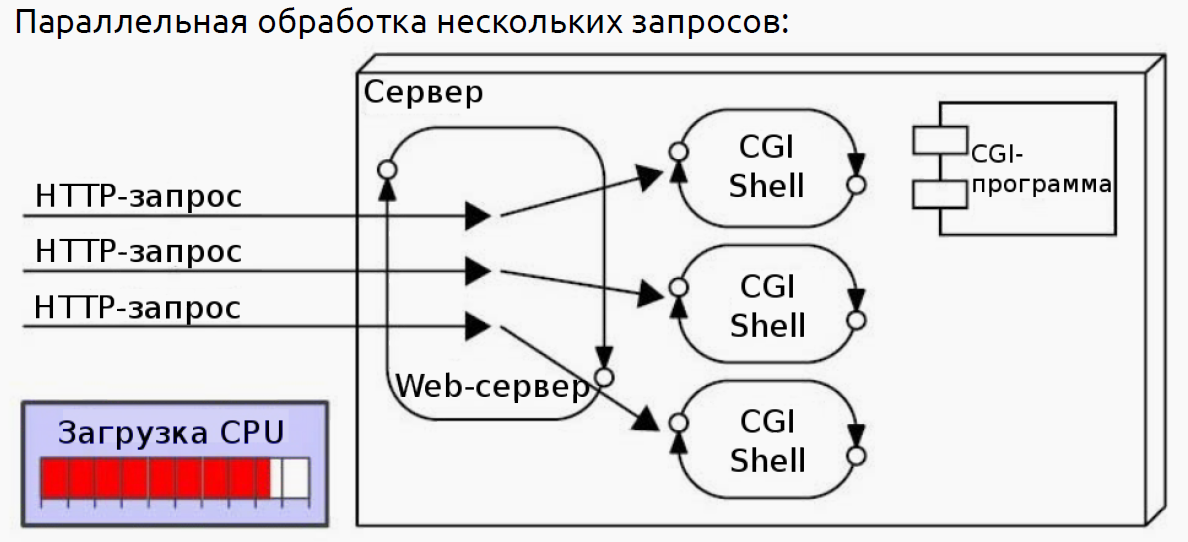
2) CGI обработка запроса, преимущества и недостатки

● CGI — простейший механизм вызова пользователем программ на стороне сервера.

● Данные отправляются программе посредством HTTP-запроса, формируемого веб-браузером.

● То, какая именно программа будет вызвана, обычно определяется URL запроса.

● Каждый запрос обрабатывается отдельным процессом CGI-программы.

● Взаимодействие программы с веб-сервером осуществляется через stdin и stdout. 

Достоинства:

● Программы могут быть написаны на множестве языков программирования.

● «Падение» CGI-сценария не приводит к «падению» всего сервера.

● Исключены конфликты при параллельной обработке нескольких запросов.

● Хорошая поддержка веб-серверами.

Недостатки:

● Высокие накладные расходы на создание нового процесса.

● Плохая масштабируемость.

● Слабое разделение уровня представления и бизнес-логики.

● Могут быть платформо-зависимыми.

3) Написать сервлет, который принимает из http запроса параметр name и выводит его.

public class MyServlet extends HttpServlet {

public void service(HttpServletRequest req, HttpServletResponse res) {

var out = res.getWriter();

out.println(req.getParameter(“name”));

}

}

### Билет 18

1) Преимущества и недостатки ajax

Преимущества:

● экономия трафика;

● уменьшение нагрузки на сервер;

● ускорение реакции интерфейса;

Недостатки:

● отсутствие интеграции со стандартными инструментами браузера;

● динамически загружаемое содержимое недоступно поисковикам;

● старые методы учёта статистики сайтов становятся неактуальными;

● усложнение проекта;

● требуется включенный JavaScript в браузере.

2) Директива page: назначение, особенности, атрибуты

● Позволяет задавать параметры, используемые контейнером при управлении жизненным циклом страницы.

● Обычно расположена в начале страницы.

● На одной странице может быть задано несколько директив page с разными указаниями контейнеру.

● Синтаксис: <%@ page attribute="value" %>

**Атрибут Для чего нужен**

buffer Задаёт параметры буферизации и размер буфера для потока вывода сервлета.

autoFlush Указывает, автоматически ли выгружается содержимое буфера при его переполнении.

contentType Позволяет задать Content Type и кодировку страницы.

errorPage Позволяет задать страницу, на которую будет осуществлено перенаправление при возникновении Runtime Exception.

isErrorPage Указывает, является ли текущая страница Error Page.

extends Позволяет задать имя родительского класса, от которого будет наследоваться сервлет.

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

@WebServlet(name = ”MyServlet”, urlPatterns = { ”\*.html”, ”\*.xhtml” })

public class MyServlet extends HttpServlet {}

### Билет 19

1) HTTP методы

GET - запрос содержимого ресурса

HEAD - аналог GET, но в ответе отсутствует тело

POST - передача данных ресурсу

PUT - загрузка содержимого запроса на указанный URI

PATCH: Аналогичен PUT, но применяется к фрагменту ресурса

DELETE: Удаляет определенный ресурс

TRACE: Возвращает полученный запрос, таким образом есть возможность увидеть информацию, добавленную промежуточными серверами

CONNECT: Преобразует соединение запроса в прозрачный TCP/IP-туннель

2) Thymeleaf. Особенности архитектуры. Отличия от FreeMаrker..

Компилирующий обработчик шаблонов.

● «Выходной» формат – XML, XHTML и HTML5. Также поддерживаются JS, CSS и Plain Text.

● Не привязан к Servlet API – может работать как «онлайн», так и «оффлайн».

● Построен на модульной системе. Модули называются *диалектами (dialects)*.

● Поддержка возможностей i18n и l10n.

● Есть встроенный кеш скомпилированных шаблонов.

● Thymeleaf – модульный движок. Модуль Thymeleaf называется *диалектом (dialect)*.

● Диалект состоит из одного или нескольких *процессоров (processor)*.

● Процессор – объект, который применяет некоторую логику к формируемому на основе шаблона артефакту.

● «Из коробки» Thymeleaf содержит стандартный диалект (Standard Dialect), которого достаточно для решения большинства типовых задач.

● В стандартном диалекте Thymeleaf реализована поддержка 5 видов выражений:

● ${...} – Variable expressions. - Позволяют обращаться к переменной из контекста Thymeleaf.

В случае использования совместно со Spring – позволяют обращаться к атрибутам модели:

<span th:text="**${book.author.name}**">

**((Book)context.getVariable("book"))**

**.getAuthor()**

**.getName()**

● \*{...} – Selection expressions.- Позволяют обратиться к выбранному ранее объекту

вместо обращения к контексту.

<div th:object="**${book}**">

…

<span th:text="**\*{title}**">...</span>

…

</div>

{

**final Book selection =**

**(Book) context.getVariable("book");**

**output(selection.getTitle());**

}

● #{...} – Message (i18n) expressions. - Позволяют обращаться к сообщениям из файлов локализации

**#{main.title}**

● @{...} – Link (URL) expressions. - Позволяют формировать URL внутри контекста приложения

<a th:href="**@{/order/list}**">...</a>

● ~{...} – Fragment expressions. - Позволяют компоновать шаблоны из фрагментов <div th:insert="**~{commons :: main}**">

...

</div>

Литералы/Literals:

* + **Текст**: 'one text', 'Another one!',...
  + **Число**: 0, 34, 3.0, 12.3,...
  + **Boolean**: true, false
  + **Null**: null
  + **Токены**: one, sometext, main,...
* Текст:
  + **Соединение строк**: +
  + **Подстроки**: |The name is ${name}|
* Арифметика:
  + **Binary**: +, -, \*, /, %
  + **Минус (unary operator)**: -
* Boolean:
  + **Binary**: and, or
  + **Boolean отрицание (unary operator)**: !, not
* Сравнение и равенство:
  + **Сравнение**: >, <, >=, <= (gt, lt, ge, le)
  + **Равенство**: ==, != (eq, ne)
* Условные:
  + **If-then**: (if)? (then)
  + **If-then-else**: (if)? (then): (else)
  + **Default**: (value) ?: (defaultvalue)
* Специальные токены:
  + **No-Operation**: \_

Ключевые отличия Thymeleaf и FreeMarker

1. Поддержка специальных синтаксических конструкций. Хотя Thymeleaf и FreeMarker имеют некоторые отличий, одним из ключевых является поддержка специальных синтаксических конструкций.

2. Ограниченный продукт. Thymeleaf ограниченный продукт, который может не подойти для разработки сложных веб-приложений или облачных проектов. FreeMarker, напротив, предлагает большой выбор возможностей, которые делают его идеальным выбором для разработчиков, работающих в облачной среде.

3. Поддержка HTML5. Thymeleaf полностью поддерживает новый стандарт HTML5, в то время как FreeMarker не имеет уникальных инструментов, чтобы поддерживать этот формат.

4. Наследование шаблонов. Thymeleaf позволяет иметь вложенные шаблоны для более эффективной структуризации кода и улучшения удобства работы. Это очень полезно, когда вам нужно иметь несколько файлов шаблонов для разных блоков на сайте. FreeMarker также поддерживает наследование шаблонов, но его использование не так явно прописано, как в Thymeleaf.

3) Напишите js функ. которая заменит все текстовые поля на кнопки с тем же текстом

function replaceText() {

var objects = document.getElementsByTagName(“p”);

for (let a of objects) {

a. innerHTML= "<button>" + a.innerText + "</button>";

}

}

----

function replaceTextFields() {

Array.prototype.slice.call(document.querySelectorAll('input[type=text]')).forEach(function(textField) {

textField.setAttribute('type', 'submit');

});

}

### Билет 20

1) Filter пост предобработка

Для создания Фильтра нужно реализовать интерфейс Filter, а также замапиться на определенный URL (или даже их набор). У него ключевой и единственный метод на реализацию - doFilter(). FilterChain - это цепочка фильтров, которая представляет собой их последовательность.

Сначала вызывается первый, он делает **предобработку**, и если запрос удовлетворяет условиям, вызывает chain.doFilter, чтобы вызвался следующий фильтр. После последнего фильтра управление передается сервлету.

После chain.doFilter можно добавить некоторый код - **постобработку**. Он уже не будет определять, пойдет ли запрос дальше. Там можно расположить логику связанную с логами, сбором статистики и т.п.

● Фильтры позволяют осуществлять пред- и постобработку запросов до и после передачи их ресурсу (сервлету, JSP или HTML-странице).

● Пример предобработки — допуск к странице только авторизованных пользователей.

● Пример постобработки — запись в лог времени обработки запроса.

● Реализуют интерфейс javax.servlet.Filter.

● Ключевой метод — doFilter.

● Метод doFilter класса FilterChain передаёт управление следующему фильтру или целевому ресурсу; таким образом, возможна реализация последовательностей фильтров, обрабатывающих один и тот же запрос.

2) Thymeleaf выражения. Стандартные выражения  
● Значения атрибутов присваиваются путём вычисления выражений (expressions).

● Выражения, поддерживаемые стандартным диалектом, называются стандартными выражениями.

● В стандартном диалекте Thymeleaf реализована поддержка 5 видов выражений:

● ${...} – Variable expressions. - Позволяют обращаться к переменной из контекста Thymeleaf.

В случае использования совместно со Spring – позволяют обращаться к атрибутам модели:

<span th:text="**${book.author.name}**">

**((Book)context.getVariable("book"))**

**.getAuthor()**

**.getName()**

● \*{...} – Selection expressions.- Позволяют обратиться к выбранному ранее объекту

вместо обращения к контексту.

<div th:object="**${book}**">

…

<span th:text="**\*{title}**">...</span>

…

</div>

{

**final Book selection =**

**(Book) context.getVariable("book");**

**output(selection.getTitle());**

}

● #{...} – Message (i18n) expressions. - Позволяют обращаться к сообщениям из файлов локализации

**#{main.title}**

● @{...} – Link (URL) expressions. - Позволяют формировать URL внутри контекста приложения

<a th:href="**@{/order/list}**">...</a>

● ~{...} – Fragment expressions. - Позволяют компоновать шаблоны из фрагментов <div th:insert="**~{commons :: main}**">

...

</div>

Литералы/Literals:

* + **Текст**: 'one text', 'Another one!',...
  + **Число**: 0, 34, 3.0, 12.3,...
  + **Boolean**: true, false
  + **Null**: null
  + **Токены**: one, sometext, main,...
* Текст:
  + **Соединение строк**: +
  + **Подстроки**: |The name is ${name}|
* Арифметика:
  + **Binary**: +, -, \*, /, %
  + **Минус (unary operator)**: -
* Boolean:
  + **Binary**: and, or
  + **Boolean отрицание (unary operator)**: !, not
* Сравнение и равенство:
  + **Сравнение**: >, <, >=, <= (gt, lt, ge, le)
  + **Равенство**: ==, != (eq, ne)
* Условные:
  + **If-then**: (if)? (then)
  + **If-then-else**: (if)? (then): (else)
  + **Default**: (value) ?: (defaultvalue)
* Специальные токены:
  + **No-Operation**: \_

3) Написать PHP скрипт который формирует форму с логиным и паролем, которая при сабмите отправляет данные на authenticate.php посредством SuperAgent, и если успешно тогда перенаправить пользователя на main.php

<!DOCTYPE html>

<html>

<head>

<titleВход></title>

</head>

<body>

<form id=”login-form”>

<label for=”username”>Login:</label>

<input type=”text” id=”username” name=”username” required><br>

<label for=”password”>Password:</label>

<input type=”password” id=”password” name=”password” required><br>

<button type=”submit”>Log in</button>

</form>

<script src=”https://cdn.jsdelivr.net/npm/superagent”></script>

<script>

document.getElementById(’login-form’).addEventListener(’submit’,function (e) {

e.preventDefault();

const username = document.getElementById(’username’).value;

const password = document.getElementById(’password’).value;

superagent.post(’authenticate.php’)

.send({ username, password })

.end(function (err, res) {

if (res.status === 200) {

window.location.href = ’main.php’;

} else {

alert(’Error’);

}

});

});

</script>

</body>

</html>

<?php

if ($\_SERVER[’REQUEST\_METHOD’] === ’POST’) {

$username = $\_POST[’username’];

$password = $\_POST[’password’];

if ($username === ’login’ && $password === ’password’) {

http\_response\_code(200);

exit;

} else {

http\_response\_code(401);

exit;

}

}

?>

---

<form onsubmit=”auth()”>

<input type=”text” id=”login”>

<input type=”password” id=”password”>

</form>

JS:

function auth() {

let login = document.querySelector(“#login”).value;

let password = document.querySelector(“#password”).value;

request.post(“authenticate.php”)

.send({ login, password })

.then(res => {

localStorage.setItem(“token”, res.token);

location.href = “main.php”;

})

.catch(err => {

alert(“Произошла ошибка”)

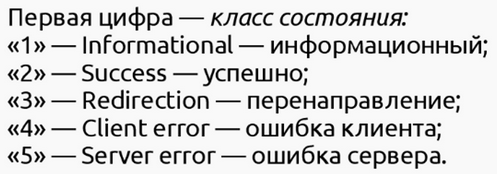
});

}

### Билет 21

1) Коды состояния HTTP

Это значение, кратко охарактеризовывающее результат обработки запроса



Две цифры не описаны, разработчик может выбирать их на свое усмотрение. Однако же ничего не мешает, как и в случае с методом HTTP, указать не тот класс состояния, что делать все же не стоит.

2) JSP Expression Language, что такое, зачем нужно, чем отличается от JSP, как обрабатывается веб контейнером

● Расширение JSP, позволяющее удобно работать с JavaBeans-компонентами без написания кода на Java.

● Позволяет использовать на страницах арифметические и логические выражения.

● Поддерживается «из коробки», можно отключить в настройках конкретной страницы и / или приложения.

Expression Language предоставляет компактный синтаксис для обращения к массивам, коллекциям, объектам и их свойствам внутри страницы jsp.

EL обрабатывается веб-контейнером (например, Apache Tomcat или WildFly) следующим образом:

1. При обработке JSP-страницы, содержащей выражения EL, веб-контейнер анализирует страницу и находит выражения в **${}**.
2. Веб-контейнер выполняет вычисления, необходимые для вычисления каждого выражения. Это может включать доступ к соответствующим Java-объектам, вызов методов и вычисление значений.
3. Результаты вычислений вставляются обратно в JSP страницу вместо самих выражений EL.
4. Итоговая JSP страница, уже с вставленными данными из EL, отправляется клиенту (браузеру) в виде HTML.

JSP (JavaServer Pages): Это технология для создания динамических веб-страниц, позволяющая вставлять Java-код в HTML-разметку.

JSP Expression Language (EL): Это инструмент для упрощения доступа к данным в JSP-страницах без необходимости вставлять Java-код, используя синтаксис ${...}.

JSP EL отличается от обычных JSP-скриплетов (Java-кода внутри JSP-страниц) тем, что он

предоставляет более простой и чистый способ вставки данных в веб-страницы, не требуя явного написания Java-кода. Он обрабатывается веб-контейнером автоматически при выполнении JSP-страницы, и его результаты вставляются в HTML-код страницы перед отправкой клиенту.

JSP EL - это мощное средство для создания динамических веб-страниц, и его использование способствует улучшению читаемости и поддерживаемости кода JSP.

Он просматривает контексты на предмет наличия бина с указанным именем в следующем порядке:

1. Контекст страницы (данные сохраняются в PageContext)
2. Контекст запроса
3. Контекст сессии
4. Контекст приложения

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

#dhdhgd img {

transform: rotate(90deg);

}

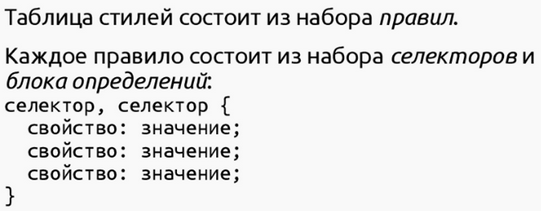
### Билет 22

1) CSS : назначение, правила, приоритеты

Cascade StyleSheets (CSS) – язык описания внешнего вида HTML-документа. CSS - технология описания внешнего вида документа, написанного языком разметки. Это цвет, шрифт, форма и т.п. Можно указывать стили в качестве атрибута, но это смешивает представление и содержание документа. CSS решает эту проблему.

● Используется для задания цветов, шрифтов и других аспектов представления документа.

● Основная цель — разделение содержимого документа и его представления.



● Если к одному элементу «подходит» сразу несколько стилей, применён будет наиболее приоритетный.

● Приоритеты рассчитываются таким образом (от большего к меньшему):

1. свойство задано при помощи !important;

2. стиль прописан напрямую в теге;

3. наличие идентификаторов (#id) в селекторе;

4. количество классов (.class) и псевдоклассов (:pseudoclass) в селекторе;

5. количество имён тегов в селекторе.

● Имеет значение относительный порядок расположения свойств — свойство, указанное позже, имеет приоритет.

Основные виды селекторов:

\* - любые элементы

div элементы с тегом div

#id - элемент по id

.class - элементы с классом class

[name="value"] - селекторы по атрибуту

:visited - псевдоклассы

div p - элементы p, являющиеся потомками div

div > p – только непосредственные потомки

div ~ p – правые соседи: все p на том же уровне вложенности, которые идут после div

div + p – первый правый сосед: p на том же уровне вложенности, который идёт сразу после div

2) MVC : назначение, элементы, примеры реализации

**Model-View-Controller** — схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

* Модель (Model) предоставляет данные и методы работы с ними: запросы в базу данных, проверка на корректность (не зависит от представления (не знает как данные визуализировать) и контроллера (не имеет точек взаимодействия с пользователем). Примеры реализации: Модель может быть классом или набором классов, которые представляют объекты и операции, связанные с этими объектами.
* Представление (View) отвечает за получение необходимых данных из модели и отправляет их пользователю (не обрабатывает введённые данные пользователя). Примеры реализации: Представление может быть HTML-шаблоном в веб-приложении, окном или формой в приложении с графическим интерфейсом.
* Контроллер (Controller) обеспечивает «связи» между пользователем и системой. Контролирует и направляет данные от пользователя к системе и наоборот. Примеры реализации: В веб-приложении контроллер может быть обработчиком маршрута, который определяет, какой метод модели вызвать и какое представление отобразить в ответ на запрос пользователя.

// Контролеер

class CalcController : IController {

public void OnClick( int number ) {

view.Total = model.SetInput(number).ToString();

}

public void OnAdd() {

model.ChangeToAddState();

}

}

// Представление

public class frmCalcView : Form, ICalcView{

public string Total{

get{

return textBox1.Text;

}

set{

textBox1.Text = value;

}

}

}

// Модель

class CalculatorModel : ICalcModel{

public int SetInput ( int number ) {

if (state == States.NoOperation) {

currentValue = number;

}

else if (state == States.Add){

currentValue = Add(currentValue , number );

}

return currentValue;

}

}

Примеры Реализации MVC: MVC широко используется в разработке веб-приложений, десктопных приложений и мобильных приложений. Вот некоторые примеры реализации MVC:

(a) Веб-приложения: Веб-приложения используют MVC для разделения логики на сервере (контроллер), представления на клиентской стороне (представление) и базы данных (модель).

(b) Десктопные приложения: Для десктопных приложений MVC разделяет графический интерфейс (представление), бизнес-логику (контроллер) и данные (модель).

Изображение выглядит как диаграмма, снимок экрана, графическая вставка, интернет

Автоматически созданное описание

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

function closeWindowIfGoogle() {

if (location.href == “<https://www.google.ru>/”) {

window.close();

}

}

### Билет 23

1) DOM и BOM

**DOM** - программный интерфейс для HTML и XML документов, описывающий структурированное представление документа и определяющий, как это структура может быть доступна из программ, которые могут изменять ее содержимое.

Согласно DOM документ может быть представлен в виде объектов, с которыми можно производить манипуляции:

1. генерация и добавление узлов
2. получение узлов
3. изменение узлов
4. изменение связей между ними
5. удаление узлов

**BOM** - объектная модель браузера. Основное предназначение — управление окнами браузера и обеспечение их взаимодействия. BOM специфична для каждого браузера. Каждое из окон браузера представляется объектом window. Может создавать системные диалоги, управлять информацией о параметрах монитора и браузера и т.д.

Обращение к узлам происходит с помощью элементов document (DOM) или window (BOM).

2) Управление сессиями. HttpSession

Сеанс (сессия) – соединение между клиентом и сервером, устанавливаемое на определенное время, за которое клиент может отправить на сервер сколько угодно запросов. Сеанс устанавливается непосредственно между клиентом и Web-сервером. Каждый клиент устанавливает с сервером свой собственный сеанс.

Чтобы открыть новый сеанс, используется метод getSession() интерфейса HttpServletRequest. Метод извлекает из переданного в сервлет запроса объект сессии класса HttpSession, соответствующий данному пользователю.

Чтобы сохранить значения переменной в текущем сеансе, используется метод setAttribute(), прочесть – getAttribute(), удалить – removeAttribute(). Список имен всех переменных, сохраненных в текущем сеансе, можно получить, используя метод Enumeration getAttributeNames().

Есть 3 способа отслеживания сессии: cookies, переопределяемый URL (используется response.encodeURL() для каждой ссылки, который вставляет идентификатор сессии в каждый URL.), cкрытые поля форм.

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

<textarea onkeyup="validArea(this)"></textarea>

<script>

function validArea(area) {

area.value = area.value.replace(/[^0-9a-zA-Z]/g, '');

}

</script>

--- если наоборот нельзя

<textarea oninput="validArea(this)"></textarea>

<script>

function validArea(area) {

area.value = area.value.replace(/[0-9a-zA-Z]/g, '');

}

</script>

### Билет 24

1) ECMAscript

ECMAScript — стандарт языков, не имеющий средств ввода-вывода язык программирования, используемый в качестве основы для построения других скриптовых языков.

ECMAScript - ядро JavaScript.

Ключевые особенности ES:

1. Встраиваемый - не используемый самостоятельно, встраиваемый во что-то
2. Расширяемый, не имеющий средств ввода/вывода
3. 5 примитивных типов данных: Number, String, Boolean, Null, Undefined
4. Объектный тип данных – Object
5. 15 различных видов инструкций.
6. Блок не ограничивает область видимости (можно объявить переменную внутри цикла и вызвать её вне цикла)
7. Можно объявить переменные, которые видны везде
8. Функция — это тоже объект

ES6

Это обновление добавило новый синтаксис для написания классов и модулей, добавились итераторы и циклы for/of, Python-style генераторы, двоичные данные, лямбда-выражения, типизированные массивы, коллекции, обещания (promises), рефлексию и прокси, усовершенствовали числа и математику. Добавлено ключевое слово let (которое помогает объявить переменной область видимости - блок) и const.

ES7

Добавлена операция возведения в степень (\*\*), Array.prototype.includes().

2) Правила трансляции jsp

Статический HTML транслируется в out.write. Это происходит на этапе трансляции .jsp в .java.

JSP страницу сервер транслирует в сервлет, как правило, при первом обращении к ней. Трансляция осуществляется один раз, выполнение кода результирующего сервлета – каждый раз в ответ на запрос.

При загрузке в веб-контейнер страницы JSP транслируются компилятором (jasper) в сервлеты.

Во время фазы трансляции контейнер локализует или создаёт класс реализации JSP-страницы, соответствующий данной JSP-странице. Этот процесс определяется семантикой JSP-страницы. Контейнер интерпретирует имеющиеся на этой странице стандартные директивы и элементы, ссылающиеся на библиотеки тэгов. Библиотека тэгов может по выбору представлять метод проверки корректности использования на JSP-странице данной библиотеки.

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

<!DOCTYPE html>

<html>

<head>

<title>Test</title>

</head>

<body>

<form action=”обработчик.php” method=”post”>

<h2>Q 1</h2>

<p>What is HTML?</p>

<label for=”q1a”><input type=”radio” name=”question1” value=”a” id=”q1a”> a) </label><br>

<label for=”q1b”><input type=”radio” name=”question1” value=”b” id=”q1b”> b) </label><br>

<label for=”q1c”><input type=”radio” name=”question1” value=”c” id=”q1c”> c) </label><br>

…

<h2>Q 2</h2>

<p>What is CSS?</p>

<label for=”q2a”><input type=”radio” name=”question2” value=”a” id=”q2a”> a) </label><br>

<label for=”q2b”><input type=”radio” name=”question2” value=”b” id=”q2b”> b) </label><br>

<label for=”q2c”><input type=”radio” name=”question2” value=”c” id=”q2c”> c) </label><br>

…

<input type=”submit” value=”Submit”>

</form>

</body>

### Билет 25

1) Структура http-запроса

HTTP запрос состоит из трех основных частей, которые идут в нем именно в определенном порядке:

1. строка запроса (Request Line) - указывает метод передачи, URL-адрес, к которому нужно обратиться и версию протокола HTTP.
2. заголовки (Message Headers) - описывают тело сообщений, передают различные параметры и др. сведения и информацию.
3. пустая строка (разделитель)
4. тело сообщения (Entity Body) – необязательный параметр - данные, которые передаются в запросе.

● Стартовая строка:

Метод URI HTTP/Версия

GET /spip.html HTTP/1.1

● Заголовки:

Host: se.ifmo.ru

User-Agent: Mozilla/5.0 (X11; U; Linux i686; ru; rv:1.9b5)

Gecko/2008050509 Firefox/3.6

Accept: text/html

Connection: close

● Тело сообщения

Заголовки HTTP:

● Формат:

ключ:значение

● 4 группы:

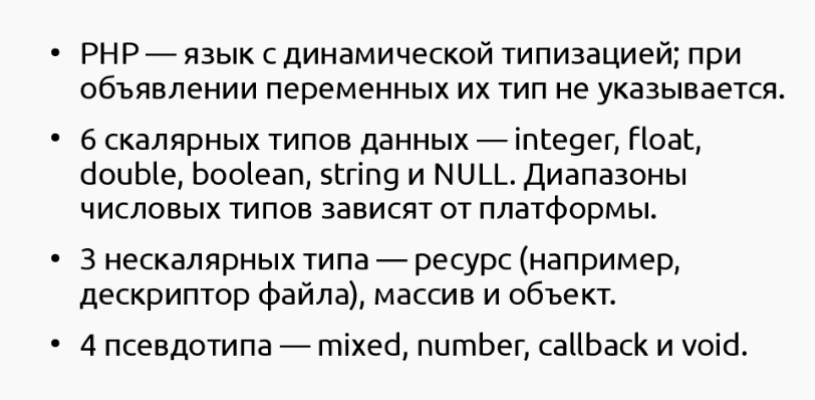
● General Headers — могут включаться в любое сообщение клиента и сервера. Пример — Cache-Control.

● Request Headers — используются только в запросах клиента. Пример — Referer.

● Response Headers — используются только в запросах сервера. Пример — Allow.

● Entity Headers — сопровождают любую сущность сообщения. Пример — Content-Language.

2) Типы данных в PHP



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

@WebServlet("/\*")

class Abobus extends HttpServlet {

public void service(HttpServletRequest req, HttpsServletResponse res) {

res.sendRedirect(“https://google.com”);

}

}

### Билет 26

1) ESMA script преимущества 6 и 7 версий

1. Ограничение области видимости - let и const (запрет изм.)
2. Шаблоны литералов:

let username=”me”;

console.log(`Hello, ${username}`);

1. Деструктуризация:

1. из массива:

let [first, , third, , firth] = array;

2. обмен значениями:

[a, b] = [b, a];

3. нескольких возвращаемых значений:

const {first, second, third} = getOneTwoThree();

4. объекта, передаваемого в кач-ве аргумента:

function hello({name, surname});

user = {name: “miron”, surname: “znanie”};

hello(user);

5. объекта:

const {name, surname} = hello(user);

1. Синтаксический сахар для классов. По сути это ф-ция с полем prototype, в значении которой лежит “родитель”. Доступ к родителю - через ключевое слово super.  
   (e.g. class Pig extends Animal).
2. Синтаксический сахар промисов вместо колбэков. После выполнения некоторого события будет асинхронно вызвано другое событие
3. Стрелочные ф-ции:

(aboba) => aboba.action()

1. Цикл с итератором: for (const a of array)
2. Параметры по умолчанию: function f(name, surname = “none”)
3. Rest-параметры (aka varargs в java): function f(format, …params)
4. Развернуть массив: [...array1, …array2, value]
5. В ES7 добавили Array.includes (names.includes(“check”) и возведение в степень \*\*

---

ES6

Это обновление добавило новый синтаксис для написания классов и модулей, добавились итераторы и циклы for/of, Python-style генераторы, двоичные данные, лямбда-выражения, типизированные массивы, коллекции, обещания (promises), рефлексию и прокси, усовершенствовали числа и математику. Добавлено ключевое слово let (которое помогает объявить переменной область видимости - блок) и const.

ES7

Добавлена операция возведения в степень (\*\*), Array.prototype.includes().

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

● Описаны в книге 1994 г. *«Design Patterns: Elements of Reusable Object-Oriented Software»* («Приёмы объектно-ориентированного проектирования. Паттерны проектирования»).

● Авторы — *Эрих Гамма* (Erich Gamma), *Ричард Хелм* (Richard Helm), *Ральф Джонсон* (Ralph Johnson), *Джон Влиссидс* (John Vlissides) — Gang of Four (GoF, «Банда Четырёх»).

● В книге описаны 23 классических шаблона проектирования.

Порождающие GoF-паттерны:

● Abstract Factory — Абстрактная фабрика.

● Builder — Строитель.

● Factory Method — Фабричный метод.

● Prototype — Прототип.

● Singleton — Одиночка.

Структурные GoF-паттерны

● Adapter — Адаптер.

● Bridge — Мост.

● Composite — Компоновщик.

● Decorator — Декоратор.

● Facade — Фасад.

● Flyweight — Приспособленец.

● Proxy — Заместитель.

Поведенческие GoF-паттерны

● Chain of responsibility — Цепочка обязанностей.

● Command — Команда.

● Interpreter — Интерпретатор.

● Iterator — Итератор.

● Mediator — Посредник.

● Memento — Хранитель.

● Observer — Наблюдатель.

● State — Состояние.

● Strategy — Стратегия.

● Template — Шаблонный метод.

● Visitor — Посетитель.

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

<!DOCTYPE html>

<html>

<head>

<title>Session Count</title>

</head>

<body>

<h1>Session Count: <span id=”sessionCount” ></span></h1>

<script>

fetch(”/SessionCountServlet”)

.then(response => response.text())

.then(data => {

document.getElementById(”sessionCount”).textContent = data;

});

</cript>

</body>

</html>

@WebServlet(”/SessionCountServlet”)

public class SessionCountServlet extends HttpServlet {

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

HttpSession session = request.getSession();

ServletContext application = getServletContext();

Integer sessionCount = (Integer) application.getAttribute(”sessionCount”);

(sessionCount == null) {

sessionCount = 0;

}

sessionCount++;

application.setAttribute(”sessionCount”, sessionCount);

response.setContentType(”text/plain”);

response.getWriter().write(sessionCount.toString());

}

}

### Билет 27

1) FastCGI. Плюсы, минусы, отличия от CGI

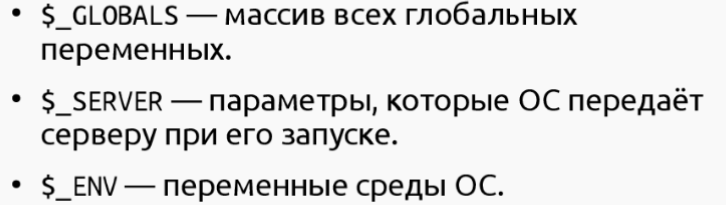
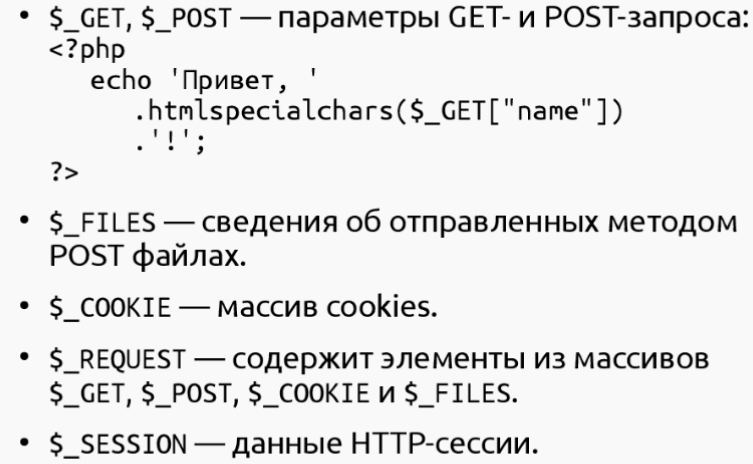
В CGI на каждый запрос форкается процесс. В случае с FastCGI механика другая: это задача вас самих написать приложение, которое будет что-то слушать на HTTP`шном порту. И мы можем сконфигурировать веб-сервер на ретрансляцию запроса с определенного урла на TCP-шный порт локальной или удаленной машины-сервера. А что дальше будет происходить с обработкой - то уже зависит от программиста.

---

CGI — устаревшая технология, позволяющая взаимодействовать веб-серверу с сервером приложений. Для каждого запроса запускается процесс с интерпретатором PHP, после возвращения ответа он завершается. Поскольку это очень неэффективно, был создан FastCGI, в котором процесс интерпретатора не завершается, а используется для последующих запросов. В этом есть и плюсы и минусы. К плюсам можно отнести Поддержка долгоживущих процессов, Улучшенная производительность, Снижение нагрузки на сервер, к минусам то, что если что-то сломается, то в CGI упадет один процесс и (наверное) этого никто не заметит, в FastCGI упадет все. Кроме того, эта штука работает не через stdin/stdout, а что-то другое. Веб-сервер взаимодействует с процессом через UNIX Domain Sockets или TCP/IP (а не через stdin и stdout).

2) Суперглобальные массивы в PHP (SuperGlobal massive)

Есть предопределенные (не надо явно указывать global $var) массивы (и переменные), которые доступны в любом месте скрипта.

* 
* 

3) Страница JSP, проверяющая есть ли jsessionid в запросе и если нету - выводящая сообщение об ошибке с кодом ошибки.

<%@ page language=”java” contentType=”text/html; charset=UTF-8” pageEncoding=”UTF-8” %>

<!DOCTYPE html>

<html>

<head>

<meta charset=”UTF-8”>

<title>jsessionid check</title>

</head>

<body>

<%

String jsessionid = request.getParameter(”jsessionid”);

if (jsessionid == null || jsessionid.isEmpty()) {

response.setStatus(HttpServletResponse.SC\_BAD\_REQUEST);

out.println(”<h1>Error: no jsessionid</h1>”);

} else {

out.println(”<h1>jsessionid in request: ” + jsessionid + ”</h1>”);

}

%>

</body>

</html>

### Билет 28

1. Правила построения html-форм

2. Конфигурация сервлетов. Файл web.xml

3. Написать css чтоб по клику все ссылки кроме внутри h1 подчеркивались

1) HTML-формы

● Предназначены для обмена данными между пользователем и сервером.

● Документ может содержать любое число форм, но одновременно на сервер может быть отправлена только одна из них.

● Вложенные формы запрещены.

● Границы формы задаются тегами <form>...</form>.

● Метод HTTP задаётся атрибутом method тега <form>:

<form method="GET" action="URL">...</form>

Форма предназначена для обмена данными между пользователем и сервером.

Задается с помощью тега <form> и могут содержать в себе атрибуты: action, содержищий URI обработчика формы (обязательный атрибут), method (по умолчанию, GET), enctype (тип кодирования), accept (MIME-типы для загрузки файлов), name, onsubmit/onreset (обработчик события submit/reset для скриптов), accept-character.

Виды полей:

* Кнопка <input>. Типы кнопок <submit>, <image>, <reset>, <button>.
* Checkbox.
* Radio.
* Select.
* Text и многострочный textarea.
* Password.
* Hidden (скрытое поле).
* File.

2)

Java веб-приложения используют файл дескриптора развертывания web.xml для определения какие URL будут передаваться определенному сервлету, какие URL требуют аутентификации. Дескриптор развертывания веб-приложений описывает классы, ресурсы и конфигурацию приложения, а так же как сервер будет использовать их для выполнения веб-запросов.

<servlet>

<servlet-name>redteam</servlet-name>

<servlet-class>mysite.server.TeamServlet</servlet-class>

<init-param>

<param-name>teamColor</param-name>

<param-value>red</param-value>

</init-param>

</servlet>

Oпределяет соответствие между путями URL и сервлетами:

<servlet-mapping>

<servlet-name>redteam</servlet-name>

<url-pattern>/red/\*</url-pattern>

</servlet-mapping>

---

1. web-app: Корневой элемент XML-файла ‘web.xml‘, который определяет структуру и пара-

метры веб-приложения.

2. servlet: Элемент, который определяет конфигурацию сервлета. Внутри него могут быть ука-

заны имя сервлета, класс сервлета и другие настройки.

3. servlet-mapping: Элемент, который связывает URL-шаблон с определенным сервлетом, опре-

деляя, какой сервлет будет обрабатывать запросы, соответствующие этому URL.

4. servlet-name: Элемент, содержащий имя сервлета, которое используется для связывания с

сервлетом, определенным в элементе ‘servlet‘.

39

5. url-pattern: Элемент, определяющий URL-шаблон, который будет использоваться для мап-

пинга сервлета на конкретные URL-адреса.

6. init-param: Элемент, позволяющий определить параметры и значения, которые могут быть

использованы сервлетом при инициализации.

7. load-on-startup: Элемент, который указывает, что сервлет должен быть инициализирован

при запуске приложения, а не при первом запросе.

8. filter: Элемент, используемый для определения фильтра и его параметров.

9. filter-mapping: Элемент, связывающий фильтр с URL-шаблоном.

10. listener: Элемент, определяющий слушателя событий, который реагирует на события в веб-

приложении.

3) CSS:

a:active {

text-decoration: underline;

}

h1 a:active {

text-decoration: none;

}

### Билет 30

1. Реализация объектно-ориентированных программ в PHP

2. Предопределенные переменные JSP

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

1) ● Полная поддержка появилась в PHP5.

● Реализованы все основные механизмы ООП —инкапсуляция, полиморфизм и наследование.

● Поля и методы могут быть приватными (private), публичными (public) и защищёнными (protected).

● Можно объявлять финальные и абстрактные методы и классы (аналогично Java).

● Множественное наследование не поддерживается, но есть интерфейсы и механизм особенностей (traits).

● Объекты передаются по ссылке.

● Обращение к константам, статическим свойствам и методам класса осуществляется с помощью конструкции «::».

class ClassName {

public $publicName;

private $privateName;

protected $protectedName;

const CONST\_VAL = 'val';

public function getPrivateName() {

return $this->$privateName; // $this -- ссылка на сам объект, $parent - на родительский.

}

};

echo ClassName::CONST\_VAL; // для обращения к константам

$classname = new ClassName();

$className->publicName; // доступ к переменной

PHP поддерживает все три основных механизма ООП — инкапсуляцию, полиморфизм подтипов и наследование (с помощью extend). Поддерживаются интерфейсы (с помощью implements). Есть абстрактные и final методы и классы. Множественное наследование не поддерживается, но класс может реализовывть несколько интерфейсов или с помощью механизма особенностей (trait), который имеет средства для разрешения конфликтов.

Методы: \_\_construct() -- конструктор \_\_destruct() -- для деинициализации объекта \_\_get(), \_\_set() \_\_sleep(), \_\_wakeup() \_\_clone()

2) В процессе трансляции контейнер добавляет в метод \_jspService ряд объектов, которые можно использовать в скриптлетах и выражениях:

* request : Эта переменная предоставляет доступ к объекту HttpServletRequest, который содержит информацию о текущем HTTP-запросе от клиента. HttpServletRequest -- позволяет обращаться к параметрам запроса (через метод getParameter), типу запроса (GET, POST, HEAD) и входящим HTTP заголовкам (cookies, Referer)
* response : Переменная response предоставляет доступ к объекту HttpServletResponse, который используется для управления HTTP-ответом, отправляемым клиенту.
* out : PrintWriter -- Переменная out предоставляет доступ к объекту JspWriter, который используется для
* вывода данных на веб-страницу session : HttpSession
* application : ServletContext С помощью переменной application можно получить доступ к объекту ServletContext,
* предоставляющему информацию и ресурсы, общие для всего веб-приложения
* config : Переменная config предоставляет доступ к объекту ServletConfig, который содержит информацию о настройках сервлета, связанного с JSP-страницей
* Page - API для доступа к экземпляру класса сервлета, в который транслируется JSP
* PageContext - контекст JSP-страницы
* Exception — используется только на страницах-перенаправлениях с информацией об ошибках (Error Pages).

3.

public class TestFilter implements Filter {

@Override

public void init(FilterConfig filterConfig) throws ServletException {

}

@Override

public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {

HttpServletRequest httpServletRequest = (HttpServletRequest) servletRequest;

if(httpServletRequest.getHeader("x-application-user") == null){

httpServletRequest.getRequestDispatcher("/").forward(servletRequest, servletResponse);

}

filterChain.doFilter(servletRequest, servletResponse);

}

@Override

public void destroy() {

}

}

31 Билет

1. HTTP, реализация и особенности

2. Жизненный цикл JSP

3. Написать калькулятор на PHP

1. HTTP — протокол прикладного уровня передачи данных. Основой HTTP является технология «клиент-сервер».

HTTP запрос состоит из трех основных частей, которые идут в нем именно в определенном порядке:

строка запроса (Request Line) - указывает метод передачи, URL-адрес, к которому нужно обратиться и версию протокола HTTP.

заголовки (Message Headers) - описывают тело сообщений, передают различные параметры и др. сведения и информацию.

пустая строка (разделитель)

тело сообщения (Entity Body) – необязательный параметр - данные, которые передаются в запросе.

Методы запросов:

GET запрашивает представление ресурса, может только извлекать данные.

HEAD как GET, только без тела ответа.

POST используется для отправк сущностей определенному ресурсу. Может изменять данные.

PUT создает новый ресурс или заменят предсавление целевого ресурса (в отличае от POST для идентичных наборов двнных будет иметь одинаковый результат).

DELETE удаляет ресурм.

CONNECT устанавливает "туннель" к серверу, определенному по ресурсу.

OPTIONS для описания параметров соединения с ресурсом

TRACE вызов возвращаемого текстового сообщения

PATCH частичное изменение ресурса

Коды ответов:

1хх: Informational

2xx: Success

3xx: Redirection (перенаправление)

4xx: Client Error

5xx: Server Error

2. Конвертацией JSP страниц в HTML код занимается контейнер.

Жизненный цикл:

Translation – JSP контейнер проверяет код JSP страницы, парсит ее для создания кода сервлета.

Compilation – JSP контейнер компилирует исходный код jsp класса и создает класс на этой фазе.

Class Loading – контейнер загружает классы в память на этой фазе.

Instantiation – внедрение конструкторов без параметров созданных классов для инициализации в памяти классов.

Initialization – в контейнере вызывается init метод объекта JSP класса и инициализируется конфигурация сервлета с init параметрами, которые указаны в дескрипторе развертывания (web.xml).

Request Processing – длительный жизненный цикл обработки запросов клиента JSP страницей. Обработка является многопоточной и аналогична сервлетам — для каждого запроса создается новая нить, создаются объекты ServletRequest и ServletResponse и происходит внедрение сервис методов JSP.

Destroy – последняя фаза жизненного цикла JSP на которой JSP класс удаляется из памяти. Обычно это происходит при выключении сервера или андеплое приложения. Методы: jspInit() , \_jspService(), jspDestroy()

3.

<!DOCTYPE html>

<html>

<head>

<title>Calculator</title>

</head>

<body>

<h2>Simple calculator in PHP</h2>

<form method="post">

<input type="text" name="num" placeholder="Number" >

<select name="operation">

<option value="add">+</option>

<option value="subtract">-</option>

<option value="multiply">\*</option>

<option value="divide">/</option>

</select>

<input type="text" name="num" placeholder="Number" >

<input type="submit" value="Calculate">

</form>

<p>Result:</p>

<?php

if ($\_SERVER['REQUEST\_METHOD'] === 'POST') {

$num1 = $\_POST['num'];

$num2 = $\_POST['num'];

$operation = $\_POST['operation'];

if (is\_numeric($num1) && is\_numeric($num2)) {

switch ($operation) {

case 'add':

$result = $num1 + $num2;

break;

case 'subtract':

$result = $num1 - $num2;

break;

case 'multiply':

$result = $num1 \* $num2;

break;

case 'divide':

if ($num2 != 0) {

$result = $num1 / $num2;

} else {

$result = 'Division by zero';

}

break;

default:

$result = 'Invalid operation';

break;

}

echo $result;

} else {

echo 'Please enter numbers to calculate.';

}

}

?>

</body>

Возможные вопросы

2. Java SE, Java EE, Java ME. Архитектура приложений Java EE, понятие о контейнере, компоненте.

Java SE — основные инструменты и библиотеки языка

Java EE — платформа для enterprise серверных приложений (JSP, JSF, сервлеты и т.д.)

Java ME — урезанная платформа для мобильных устройств (в основном использовалась до появления iOS/Android)

Архитектура приложений Java EE, понятие о контейнере, компоненте.

Сервлеты — серверные сценарии, жизненным циклом которых управляет веб-контейнер. Существует несколько веб-контейнеров (Apache Tomcat, GlassFish, Jetty, др.); они все реализуют определенный стандарт Java EE (6/7/8).

При запуске веб-контейнер читает манифест (web.xml) приложения и создает для него ServletContext. Для каждого класса сервлета и фильтра из манифеста создается объект. Контекст и объекты фильтров/сервлетов будут жить на протяжении работы приложения, т.е. одна и та же инстанция будет использоваться для всех запросов и сессий.

При поступлении HTTP запроса веб-контейнер создает новые объекты HttpServletRequest и HttpServletResponse и передает их фильтрам, потом сервлету.

Компоненты - классы, написанные по определённым правилам, которые дают возможность создания инструментов.

12. Сервлеты.

Пакеты javax.servlet и javax.servlet.http обеспечивают интерфейсы и классы для создания сервлетов.

Cервлет — это Java-класс, который наследуется обычно от класса HttpServlet и переопределяет часть методов:

doGet — если мы хотим, чтобы сервлет реагировал на GET запрос.

doPost — если мы хотим, чтобы сервлет реагировал на POST запрос.

doPut, doDelete — если мы хотим, чтобы сервлет реагировал на PUT и DELETE запрос (есть и такие в HTTP). Эти методы реализуются крайне редко, т.к. сами команды тоже очень редко встречаются.

init, destroy — для управления ресурсами в момент создания сервлета и в момент его уничтожения.

Сервлет vs CGI

Сервлеты запускаются в одном процессе (HTTP-сервер с дополнительными функциями, который называется Serlet Container), и они существуют до тех пор, пока этот процесс существует.

CGI каждый раз создает новый экземпляр процесса для обслуживания запроса. Это убийца перфоманса.

Поскольку для каждого запроса существует новый процесс, это означает, что CGI не может агрегировать данные из нескольких запросов в памяти.

Сервлет vs FastCGI

При использовании сервлетов веб-сервер может напрямую вызвать приложение.

JS-функция, открывающая в новом окне браузера сайт <http://www.google.com>

window.open("http://google.com", "newwin", "width=1200,height=600");

JS-функция, удаляющая со страницы все элементы

c классом “MarkedToRemove”

Array.prototype.slice.call(document.getElementsByTagName("div")).forEach(function (e) {

if (e.getAttribute("class") === "MarkedToRemove"){

e.parentNode.removeChild(e);

}

});

css правило, выравнивающее все блоки

внутри формы с id=”sampleForm” по правому краю

#sampleForm div{

text-align:right

}