**Вопросы к защите лабораторной работы:**

1. Java-сервлеты. Особенности реализации, ключевые методы, преимущества и недостатки относительно CGI и FastCGI.
2. Контейнеры сервлетов. Жизненный цикл сервлета.
3. Диспетчеризация запросов в сервлетах. Фильтры сервлетов.
4. HTTP-сессии - назначение, взаимодействие сервлетов с сессией, способы передачи идентификатора сессии.
5. Контекст сервлета - назначение, способы взаимодействия сервлетов с контекстом.
6. JavaServer Pages. Особенности, преимущества и недостатки по сравнению с сервлетами, область применения.
7. Жизненный цикл JSP.
8. Структура JSP-страницы. Комментарии, директивы, объявления, скриптлеты и выражения.
9. Правила записи Java-кода внутри JSP. Стандартные переменные, доступные в скриптлетах и выражениях.
10. Bean-компоненты и их использование в JSP.
11. Стандартные теги JSP. Использование Expression Language (EL) в JSP.
12. Параметры конфигурации JSP в дескрипторе развёртывания веб-приложения.
13. Шаблоны проектирования и архитектурные шаблоны. Использование в веб-приложениях.
14. Архитектура веб-приложений. Шаблон MVC. Архитектурные модели Model 1 и Model 2 и их реализация на платформе Java EE.

Оглавление

[**Java-сервлеты. Особенности реализации, ключевые методы, преимущества и недостатки относительно CGI и FastCGI.** 2](#_Toc148387359)

[Сервлет vs FastCGI 3](#_Toc148387360)

[**Контейнеры сервлетов. Жизненный цикл сервлета.** 4](#_Toc148387361)

[**Как обрабатываются запросы** 6](#_Toc148387362)

[**Портлеты** 6](#_Toc148387363)

[**Диспетчеризация запросов в сервлетах. Фильтры сервлетов.** 6](#_Toc148387364)

[Диспетчеризация запросов в сервлетах 6](#_Toc148387365)

[Фильтр сервлетов 7](#_Toc148387366)

[HTTP-сессии - назначение, взаимодействие сервлетов с сессией, способы передачи идентификатора сессии 9](#_Toc148387367)

[**Контекст сервлета - назначение, способы взаимодействия сервлетов с контекстом.** 9](#_Toc148387368)

[Структура JSP-страницы. Комментарии, директивы, объявления, скриптлеты и выражения. 11](#_Toc148387369)

[Комментарии 11](#_Toc148387370)

[Директивы 11](#_Toc148387371)

[Cкриптлеты 12](#_Toc148387372)

[Выражения 12](#_Toc148387373)

[Правила записи Java-кода внутри JSP. Стандартные переменные, доступные в скриптлетах и выражениях. 13](#_Toc148387374)

[JavaServer Pages. Особенности, преимущества и недостатки по сравнению с сервлетами, область применения. 14](#_Toc148387375)

[Bean-компоненты и их использование в JSP. 16](#_Toc148387376)

[Bean и JSP 16](#_Toc148387377)

[Жизненный цикл JSP. 17](#_Toc148387378)

[Стандартные теги JSP. Использование Expression Language (EL) в JSP. 18](#_Toc148387379)

[**Архитектура веб-приложений. Шаблон MVC. Архитектурные модели Model 1 и Model 2 и их реализация на платформе Java EE.** 19](#_Toc148387380)

**Maven** — инструмент для управления и сборки проектов — настоящий помощник Java-программиста.

# **Java-сервлеты. Особенности реализации, ключевые методы, преимущества и недостатки относительно CGI и FastCGI.**

**Java Servlet API** — стандартизированный API, предназначенный для реализации на сервере и работе с клиентом по схеме запрос-ответ.

**Сервлет** — это класс, который умеет получать запросы от клиента и возвращать ему ответы. Да, сервлеты в Java — именно те элементы, с помощью которых строится клиент-серверная архитектура.

**Сервлет** – серверный сценарий написанный на java

Жизненным циклом сервлетов управляет веб-контейнер

Чтобы **из обычного класса сделать http-сервлет**, его нужно унаследовать от класса HttpServlet. Над классом указываем аннотацию @WebServlet(), в которой привязываем (мапим) сервлет к конкретному пути (“/hello”).

Чтобы **обрабатывать GET-запросы**, переопределяем метод doGet(). Обрати внимание на аргументы метода — HttpServletRequest и HttpServletResponse. С объекта HttpServletRequest мы можем взять всю необходимую информацию о запросе, в HttpServletResponse можем записать наш ответ и установить необходимые заголовки.

Ключевые методы

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

Различия сервлетов от CGI и FastCGI

* Сервлеты запускаются в одном процессе (HTTP-сервер с дополнительными функциями, который называется Serlet Container), и они существуют до тех пор, пока этот процесс существует.
* В отличие от CGI, запросы обрабатываются в отдельных потоках(а не процессах) на веб-контейнере.

Сервлеты работают многопоточно. Для каждого запроса создается свой поток — отдельно выполняющаяся последовательность действий. Потоки независимы друг от друга и разделены на уровне системы. Такой подход позволяет серверу работать быстрее и обрабатывать несколько запросов одновременно.

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

### [Сервлет vs FastCGI](https://github.com/EgorMIt/ITMO/blob/master/2%20-%20%D0%92%D0%B5%D0%B1-%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5/Lab2.md#%D1%81%D0%B5%D1%80%D0%B2%D0%BB%D0%B5%D1%82-vs-fastcgi)

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

# **Контейнеры сервлетов. Жизненный цикл сервлета.**

**Контейнер сервлетов - это компонент сервера приложений, который отвечает за обработку и управление сервлетами. Он предоставляет среду выполнения для сервлетов, а также реализует механизмы жизненного цикла и управления состоянием сервлетов.**

Сервлет размещается на сервере, однако чтобы сервер мог использовать сервлет для обработки запросов, сервер должен поддерживать движок или контейнер сервлетов (servlet container/engine).

**Кроме маршрутизации запросов, контейнер сервлетов выполняет и другие функции:**

1. Динамически генерирует HTML-страницы с JSP-файлов.
2. Зашифровывает/расшифровывает HTTPS-сообщения.
3. Предоставляет разграниченный доступ для администрирования сервлетов.

Контейнер сервлетов обеспечивает следующие функции:

- Загрузка, инициализация и уничтожение сервлетов

- Управление потоками выполнения сервлетов

- Управление сессиями и куками

- Предоставление доступа к ресурсам, таким как базы данных или файловые системы

- Обработка запросов и ответов HTTP

**Контейнер сервлетов** является частью сервера приложений и может быть реализован как веб-сервер, standalone-приложение или как часть другого приложения

**Жизненный цикл сервлета**

Для каждого сервлета движок сервлетов создает только одну копию. Вне зависимости от того, сколько запросов будет отправлено сервлету, все запросы будут обрабатываться толькой одной копией сервлета.

Объект сервлета создается либо при запуске движка сервлетов, либо когда сервлет получает первый запрос.

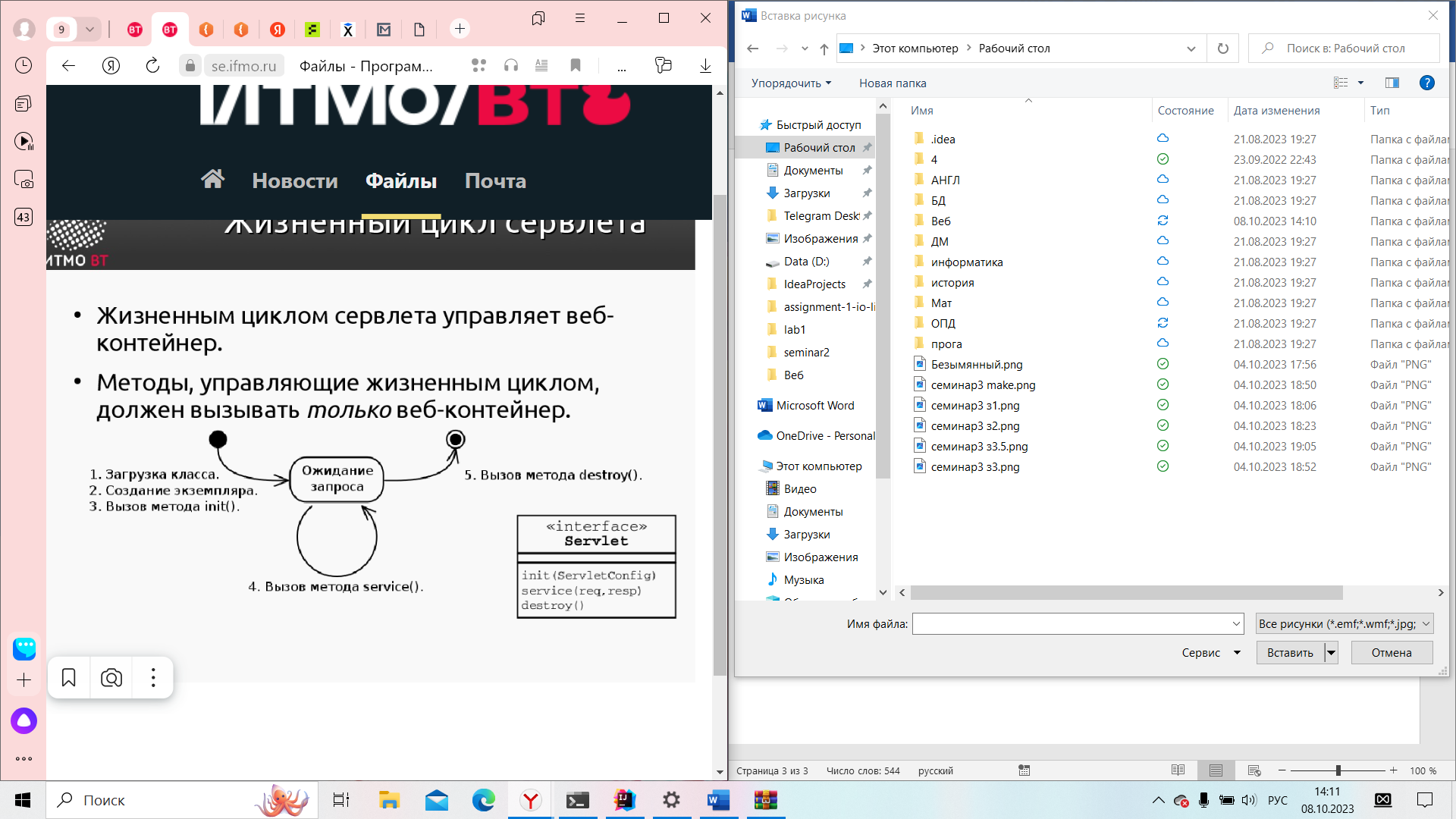
Затем для каждого запроса запускается поток, который обращается к объекту сервлета.

При работе с сервлетом движок сервлетов вызывает у класса сервлета ряд методов, которые определены в родительском абстрактном классе HttpServlet.

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

2.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.

3. 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()**.



Жизненный цикл сервлета состоит из следующих шагов:

* 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.

## **Как обрабатываются запросы**

* Если у сервлета есть метод для такого запроса, он выполняется — программа делает то, что должна.
* Если разработчик не прописал метод для этого типа запроса, сервлет пользуется стандартным — тем, который для таких запросов описан в родительском классе. Чаще всего после этого он возвращает ошибку исполнения.

# **Портлеты**

 Для них есть свои контейнеры, они тоже размещаются на сервере и выполняют определенные задачи. Более того: портлеты основаны на сервлетах и расширяют их возможности.

Основное отличие — портлетов на одной странице может быть несколько, и каждый из них отвечает за свою задачу. А сервлет на странице всегда один.

Это происходит, потому что сервлет должен работать с веб-документом целиком, а портлет — только с его частью. Поэтому портлеты можно представить как «кирпичики», из которых складывается функциональность страницы.

# **Диспетчеризация запросов в сервлетах. Фильтры сервлетов.**

### [Диспетчеризация запросов в сервлетах](https://github.com/EgorMIt/ITMO/blob/master/2%20-%20%D0%92%D0%B5%D0%B1-%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5/Lab2.md#%D0%B4%D0%B8%D1%81%D0%BF%D0%B5%D1%82%D1%87%D0%B5%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F-%D0%B7%D0%B0%D0%BF%D1%80%D0%BE%D1%81%D0%BE%D0%B2-%D0%B2-%D1%81%D0%B5%D1%80%D0%B2%D0%BB%D0%B5%D1%82%D0%B0%D1%85)

* Сервлеты могут делегировать обработку запросов другим ресурсам (сервлетам, JSP и HTML-страницам).
* Диспетчеризация осуществляется с помощью реализаций интерфейса javax.servlet.RequestDispatcher.
* Два способа получения RequestDispatcher — через ServletRequest (абсолютный или относительный URL) и ServletContext (только абсолютный URL).

Два способа делегирования обработки запроса — forward и include

**forward** — делегирует обработку запроса другому сервлету на сервере, клиент при этом не задействуется. Для этого в метод doGet() нового сервлета нужно добавить такой код:

getServletContext().getRequestDispatcher("/hello").forward(req, resp);

**redirect** — возвращает клиенту адрес, по которому нужно обратиться для обработки его запроса. Большинство браузеров переходит на переданную ссылку автоматически. Для реализации редиректа нужно добавить этот код:

resp.sendRedirect(req.getContextPath() + "/hello");

### [Фильтр сервлетов](https://github.com/EgorMIt/ITMO/blob/master/2%20-%20%D0%92%D0%B5%D0%B1-%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5/Lab2.md#%D1%84%D0%B8%D0%BB%D1%8C%D1%82%D1%80-%D1%81%D0%B5%D1%80%D0%B2%D0%BB%D0%B5%D1%82%D0%BE%D0%B2)

* 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 передает управление следующему фильтру или целевому ресурсу; таким образом возможна реализация последовательностей фильтров, обрабатывающих один и тот же запрос

Сервлетные фильтры могут:

* перехватывать инициацию сервлета прежде, чем сервлет будет инициирован;
* определить содержание запроса прежде, чем сервлет будет инициирован;
* модифицировать заголовки и данные запроса, в которые упаковывается поступающий запрос;
* модифицировать заголовки и данные ответа, в которые упаковывается получаемый ответ;
* перехватывать инициацию сервлета после обращения к сервлету.

Основой для формирования фильтров служит интерфейс javax.servlet.Filter, который реализует три метода:

void init (FilterConfig config) throws ServletException; - вызывается при создании филтра

void destroy (); - удаление филтра

void doFilter (ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException; - вызывается для обработки каждого запроса

# HTTP-сессии - назначение, взаимодействие сервлетов с сессией, способы передачи идентификатора сессии

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

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

* HTTP – stateless – протокол
* Javax.servlet.HttpSession – интерфейс, позволяющий идентифицировать конкретного клиента (браузер) при обработке множества HTTP -запросов от него
* Экземпляр **HttpSession** создается при первом обращении клиента к приложению и сохранятеся некоторое (настраемое) время после последнего обращения
* **Идентификатор сессии** либо помещается в куки, либо добавляется к URL. Если удалить этот идентификатор, то сервер не сможет опознать клиента и создаст новую сессию
* В экземпляр HttpSession можно помещать общую для этой сессии информацию (методы getAttribute и setAttribute)

# **Контекст сервлета - назначение, способы взаимодействия сервлетов с контекстом.**

**Контекст сервлета** - API, с помощью которого сервлет может взаимодействовать со своим контейнером.

* Содержит информацию о конфигурации сервлетов, параметры и атрибуты контекста, а также другие ресурсы, доступные веб-приложению.
* Доступ к методам осуществляется через интерфейс javax.servlet.ServletContext
* У всех сервлетов внутри приложения общий контекст
* В контексте можно помещать общую для всех сервлетов информацию (методы getAttribute и setAttribute)
* Если приложение – распределенное, то на каждом экземпляре JVM контейнером создается свой контекст.

Информация о контексте сервера доступна через метод  getServletContext() объекта ServletConfig (этот объект передается сервлету во время инициализации).

Для доступа к контексту сервлета используется объект ServletContext, который можно получить из объекта ServletConfig, переданного в метод инициализации сервлета

# Структура JSP-страницы. Комментарии, директивы, объявления, скриптлеты и выражения.

* 2 варианта синтаксиса – на базе HTML и XML
* Обозначаются тегами <% %>
* Существует 5 типов JSP – элементов:
  1. Комментарий - <%-- Comment --%>
  2. Директива - <%@ Directive %>
  3. Объявляение - <%! Decl %>
  4. Скриплет – <% code %>
  5. Выражение – <%= exp %>

### [Комментарии](https://github.com/EgorMIt/ITMO/blob/master/2%20-%20%D0%92%D0%B5%D0%B1-%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5/Lab2.md#%D0%BA%D0%BE%D0%BC%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D1%80%D0%B8%D0%B8)

В JSP-страницах комментарии можно разделить на две группы:

* комментарии исходного кода JSP
* комментарии HTML-разметки.
* Java - комментарии

Комментарии исходного кода JSP отмечаются специальной последовательностью символов: <%-- в начале и --%> в конце комментария. Данный вид комментариев удаляется на этапе компиляции JSP-страницы.

Комментарии HTML-разметки оформляются в соответствии с правилами языка HTML. Данный вид комментариев рассматривается JSP-компилятором как статический текст и помещается в выходной HTML-документ. JSP-выражения внутри HTML-комментариев исполняются.

<!-- Дата создания страницы: <%= new java.util.Date() %> -->

### [Директивы](https://github.com/EgorMIt/ITMO/blob/master/2%20-%20%D0%92%D0%B5%D0%B1-%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5/Lab2.md#%D0%B4%D0%B8%D1%80%D0%B5%D0%BA%D1%82%D0%B8%D0%B2%D1%8B)

Управляют процессом трансляции в сервлет

Директивы используются для указания информации о странице, такой как импортируемые классы, используемые теги и другие настройки.

Все директивы начинаются с <%@, затем следует название директивы и атрибуты со значениями, и заканчиваются %>

Примеры:

1**. page**: используется для указания настроек страницы, таких как язык программирования, тип контента и кодировка страницы

2. **include:** используется для включения содержимого другого файла в текущую страницу. Пример: <%@ include file="header.jsp" %>

3. **import:** используется для импорта классов Java на страницу. Пример: <%@ import="java.util.List, java.util.ArrayList" %>

4.**session**: используется для указания, что текущая страница будет использовать сессию. Пример: <%@ page session="true" %>

[Объявления](https://github.com/EgorMIt/ITMO/blob/master/2%20-%20%D0%92%D0%B5%D0%B1-%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5/Lab2.md#%D0%BE%D0%B1%D1%8A%D1%8F%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F)

Объявления JSP позволят вам задавать переменные, методы, внутренние классы и так далее. Объявления используются для определения используемых в программе конструкций Java.

<%! private int accessCount = 0; %>

### [Cкриптлеты](https://github.com/EgorMIt/ITMO/blob/master/2%20-%20%D0%92%D0%B5%D0%B1-%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5/Lab2.md#c%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BB%D0%B5%D1%82%D1%8B)

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

### [Выражения](https://github.com/EgorMIt/ITMO/blob/master/2%20-%20%D0%92%D0%B5%D0%B1-%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5/Lab2.md#%D0%B2%D1%8B%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F)

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() %>

# Правила записи Java-кода внутри JSP. Стандартные переменные, доступные в скриптлетах и выражениях.

*Предопределенные переменные:*

- в процессе трансляции контейнер добавляет в метод jspService ряд объектов, которые можно использовать в скриплетах и выражениях

- **out**: объект класса JspWriter, который используется для вывода текста на страницу.

**- request:** объект класса HttpServletRequest, который представляет текущий HTTP-запрос.

- **response:** объект класса HttpServletResponse, который представляет текущий HTTP-ответ.

- **session:** объект класса HttpSession, который представляет текущую сессию.

- **application:** объект класса ServletContext, который представляет контекст приложения.

- **pageContext:** объект класса PageContext, который представляет контекст текущей страницы JSP. С помощью этого объекта можно получить доступ к другим стандартным переменным и атрибутам страницы.

JSP Actions:

* XML – элементы, позволяющие управлять поведением сервлета
* Синтаксис : <jsp:action\_name attribute=”value” />

**<jsp:include>** - включение содержимого из другой JSP страницы или HTML файла.

**<jsp:forward>** - перенаправление запроса на другую страницу.

**<jsp:useBean>** - создание экземпляра JavaBean.

**<jsp:setProperty>** - установка свойств JavaBean.

**<jsp:getProperty>** - получение значения свойства JavaBean.

# JavaServer Pages. Особенности, преимущества и недостатки по сравнению с сервлетами, область применения.

* Страницы JSP – это текстовый файлы, содержащие статический HTML и JSP – элементы
* JSP – элементы позволяют формировать динамическое содержимое
* При загрузке в веб-контейнер страницы JSP транслируются компилятором (jasper) в сервлеты
* Позволяют отделить бизнес-логику от уровня представления

Динамическая часть заключается в специальные теги "<% %>"

#### [Преимущества:](https://github.com/EgorMIt/ITMO/blob/master/2%20-%20%D0%92%D0%B5%D0%B1-%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5/Lab2.md#%D0%BF%D1%80%D0%B5%D0%B8%D0%BC%D1%83%D1%89%D0%B5%D1%81%D1%82%D0%B2%D0%B0)

● Высокая производительность — транслируются в сервлеты. (сервлеты работают быстрее, чем интерпретируемый код JSP)

● Более простая интеграция с другими технологиями Java.

● Простые для понимания — структура похожа на обычный HTML.

● Более высокий уровень абстракции, что упрощает разработку приложений.

#### [Недостатки:](https://github.com/EgorMIt/ITMO/blob/master/2%20-%20%D0%92%D0%B5%D0%B1-%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5/Lab2.md#%D0%BD%D0%B5%D0%B4%D0%BE%D1%81%D1%82%D0%B0%D1%82%D0%BA%D0%B8)

● Трудно отлаживать, если приложение целиком основано на JSP.

● Большинство JSP-страниц содержат много логики, что может привести к проблемам с поддержкой кода.

● Возможны конфликты при параллельной обработке нескольких запросов.

*Область применения.*

JSP используется для создания динамических веб-страниц, которые могут изменяться в зависимости от действий пользователя или других факторов.

Основные отличия между JSP и сервлетами:

1. Синтаксис: JSP использует комбинацию HTML-кода и Java-кода, в то время как сервлеты пишутся на чистом Java.

2. Уровень абстракции: JSP представляет собой более высокоуровневый подход, который позволяет разработчикам создавать веб-страницы, используя теги и другие элементы, а сервлеты являются более низкоуровневым подходом, который требует более глубоких знаний Java.

3. Цель использования: JSP предназначены для создания веб-страниц с динамическим содержимым, в то время как сервлеты используются для обработки запросов и генерации ответов на сервере.

4. Жизненный цикл: JSP компилируются в сервлеты при первом запросе, а затем работают как сервлеты. Сервлеты же компилируются в байт-код Java и загружаются в память сервера при запуске приложения.

# Bean-компоненты и их использование в JSP.

**JavaBeans** — классы в языке Java, написанные по определённым правилам. Они используются для объединения нескольких объектов в один для удобной передачи данных.

- это объекты Java, которые используются для хранения и передачи данных между JSP страницами и сервлетами.

Чтобы класс мог работать как bean, он должен соответствовать определённым соглашениям *об именах методов, конструкторе и поведении*. Эти соглашения дают возможность создания инструментов, которые могут использовать, замещать и соединять JavaBeans.

Правила описания:

* Класс должен иметь конструктор без параметров, с модификатором доступа public. Такой конструктор позволяет инструментам создать объект без дополнительных сложностей с параметрами.
* Свойства класса должны быть доступны через get, set и другие методы (так называемые методы доступа), которые должны подчиняться стандартному соглашению об именах. Это легко позволяет инструментам автоматически определять и обновлять содержание bean’ов.
* Класс должен быть сериализуем. Это даёт возможность надёжно сохранять, хранить и восстанавливать состояние bean независимым от платформы и виртуальной машины способом.
* Класс должен иметь переопределенные методы equals(), hashCode() и toString().

### [Bean и JSP](https://github.com/EgorMIt/ITMO/blob/master/2%20-%20%D0%92%D0%B5%D0%B1-%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5/Lab2.md#bean-%D0%B8-jsp)

Устанавить связь между JSP и соответствующим ей бин-компонентом

<jsp:useBean id="\_loginJSPBean" class="lbm.examples.LoginJSPBean" scope="session"/>

Передать значения всех полей формы

# Жизненный цикл JSP.

Жизненный цикл JSP начинается с создания запроса от клиента к серверу.

Затем сервер определяет, что запрос связан с JSP и передает его контейнеру сервлетов.

Контейнер сервлетов определяет, существует ли уже скомпилированный класс JSP. Если класс не существует:

* 1. Контейнер сервлетов транслирует JSP в код сервлета
  2. Компилирует сервлет
  3. Загрузка класса сервлета в память
  4. Создание экземпляра сервлета
  5. Вызов метода jspinit() - инициализируется конфигурация сервлета с init параметрами, которые указаны в дескрипторе развертывания (web.xml).
  6. Dызывает метод \_jspService() для обработки запроса
  7. Вызов метода jspDestroy()

Жизненный цикл:

1. Translation – JSP контейнер проверяет код JSP страницы, парсит ее для создания кода сервлета.
2. Compilation – JSP контейнер компилирует исходный код jsp класса и создает класс на этой фазе.
3. Class Loading – контейнер загружает классы в память на этой фазе.
4. Instantiation – внедрение конструкторов без параметров созданных классов для инициализации в памяти классов.
5. Initialization – в контейнере вызывается init метод объекта JSP класса и инициализируется конфигурация сервлета с init параметрами, которые указаны в дескрипторе развертывания (web.xml).
6. 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.
7. 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()

# Стандартные теги JSP. Использование Expression Language (EL) в JSP.

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" %>

К стандартным тегам JSP можно отнести:

<c:out>,<c:set>,<c:remove>,<c:if>,<c:choose>,<c:forEach>,<c:import>,<c:catch>

**Expression Language или сокращенно EL** предоставляет компактный синтаксис для обращения к массивам, коллекциям, объектам и их свойствам внутри страницы jsp. Он довольн прост. Вставку окрывает знак $, затем в фигурные скобки {} заключается выводимое значение:

${attribute}

${object.property}

Например, выражение ${user.name} будет эквивалентно <jsp:getProperty name="user" property="name" />. EL также позволяет выполнение операций, например ${num1 + num2} или ${string1.toUpperCase()}.

Пример использования тега <c:forEach> для итерации по коллекции:

jsp

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

<c:forEach items="${myList}" var="item">

<p>${item}</p>

</c:forEach>

# Параметры конфигурации JSP в дескрипторе развёртывания веб-приложения.

Java веб-приложения используют **файл дескриптора** развертывания для определения какие URL будут передаваться определенному сервлету, какие URL требуют аутентификации и др.

Этот дескриптор развертывания называется web.xml и находится в WAR приложения в WEB-INF/ директории. Web.xml – часть стандарта сервлета для веб-приложений.

jsp-config>

<jsp-property-group>

<url-pattern>\*.jsp</url-pattern>

<page-encoding>UTF-8</page-encoding>

<scripting-invalid>true</scripting-invalid>

<trim-directive-whitespaces>true</trim-directive-whitespaces>

</jsp-property-group>

</jsp-config>

В этом примере мы определяем группу свойств для JSP, которые будут применяться к файлам с расширением .jsp. Мы задаем кодировку страницы, запрещаем использование скриптового кода, а также устанавливаем опцию удаления пробелов из директив.

# Шаблоны проектирования и архитектурные шаблоны. Использование в веб-приложениях.

* **Шаблоны проектирования или паттерн** – повторимая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста
* **это универсальные решения для типовых задач**, которые **возникают в процессе разработки** программного обеспечения. Они позволяют повысить качество и ускорить процесс разработки, а также улучшить сопровождаемость и расширяемость приложения.
* Позволяют избежать типовых ошибок при разработке типовых решений
* Легче поддерживать код

**Архитектурные шаблоны:**

* Более высокий уровень по сравнению с шаблонами проектирования
* Описывают архитектуру всей системы или приложения(организацией структуры приложения и взаимодействием его компонентов.)
* Имеют дело не с отдельными классами, а с целыми компонентами или модулями
* **Компоненты и модули могут быть построены с использованием различных шаблонов**

Основные архитектурные шаблоны:

* Многоуровневая архитектура
* Каналы и фильтры
* Клиент — сервер
* Модель — представление — контроллер
* Управляемая событиями архитектура

В веб-приложениях широко используются такие шаблоны проектирования, как MVC (Model-View-Controller), Front Controller, Intercepting Filter и другие.

Архитектурные шаблоны, такие как SOA (Service-Oriented Architecture), REST (Representational State Transfer)

Архитектурный шаблон REST определяет принципы построения веб-сервисов, которые используют протокол HTTP для передачи данных. Он предполагает использование уникальных идентификаторов ресурсов и ограниченного набора операций над ними.

Архитектурный шаблон SOA предполагает построение приложения из независимых сервисов, которые могут быть использованы другими приложениями. Это позволяет повысить гибкость и масштабируемость приложения.

Сервисы представляют собой независимые компоненты, которые могут быть использованы в различных приложениях и взаимодействовать друг с другом через стандартизированные протоколы и интерфейсы.

# **Архитектура веб-приложений. Шаблон MVC. Архитектурные модели Model 1 и Model 2 и их реализация на платформе Java EE.**

Архитектура веб-приложений:

- описывает структуру и организацию компонентов, которые используются для создания приложения, работающего в сети Интернет. В общем случае, веб-приложение состоит из клиентской и серверной частей.

*Клиентская часть* веб-приложения представляет **интерфейс пользователя и отвечает за взаимодействие с пользователем.** Она может быть реализована с помощью HTML, CSS и JavaScript

*Серверная часть* веб-приложения **отвечает за обработку запросов от клиента и предоставление данных**. Она может быть реализована с помощью различных технологий, таких как PHP, Java, Python и Node.js. Серверная часть может использовать базы данных, такие как MySQL или MongoDB, для хранения данных.

1. Клиент – отображение данных пользователю

2. Бизнес-логика – за обработку данных и логику приложению

3. Данные – взаимодействие с базой данных и их получение

***Архитектура Model 1.***

- предназначени для проектирования приложений небольшого масштаба и сложности. Она представляет собой простую модель, где все компоненты приложения (модель, представление и контроллер) находятся в одном слое.

- за обработку данных и представления отвечает один и тот же компонент (сервлет или JSP)

В архитектуре Model 1 все запросы от клиента направляются напрямую к сервлету, который является контроллером. Контроллер обрабатывает запрос и возвращает ответ напрямую клиенту. Модель и представление находятся в том же слое, что и контроллер

**Сервлеты** отвечают за обработку запросов от клиента и взаимодействие с моделью**. JSP** используются для отображения данных на странице.

Пример реализации архитектуры Model 1 на платформе Java EE:

- Создание сервлета, который будет обрабатывать запросы от клиента.

- Создание класса модели, который будет отвечать за обработку данных.

- Создание JSP страницы, которая будет отображать данные на странице.

- В сервлете вызывается метод модели для получения данных и передачи их на JSP страницу.

- JSP страница использует полученные данные для отображения на странице.

***Архитектура Model 2.***

Она представляет собой более сложную модель, в которой компоненты приложения разделены на три слоя: модель, представление и контроллер.

В архитектуре Model 2

1.Все запросы от клиента направляются к контроллеру

2.Контроллер обрабатывает запрос и передает данные модели.

3.Модель обрабатывает данные и возвращает результаты контроллеру.

4.Контроллер передает результаты представлению, которое отображает данные на странице и отправляет ответ клиенту.

Реализация архитектуры Model 2 на платформе Java EE может быть выполнена с использованием сервлетов, JSP и JavaBeans. **Сервлеты** отвечают за обработку запросов от клиента и взаимодействие с контроллером. **JSP** используются для отображения данных на странице, а **JavaBeans** - для работы с моделью.

Пример реализации архитектуры Model 2 на платформе Java EE:

- Создание сервлета, который будет обрабатывать запросы от клиента.

- Создание класса контроллера, который будет отвечать за обработку запросов и взаимодействие с моделью и представлением.

- Создание класса модели, который будет отвечать за обработку данных.

- Создание JSP страницы, которая будет отображать данные на странице.

- Создание JavaBean класса, который будет взаимодействовать с моделью и передавать данные контроллеру.

- В сервлете вызывается метод контроллера для обработки запроса и получения данных из модели через JavaBean.

- Контроллер передает данные на JSP страницу для отображения.

***Шаблон MVC***

* VC — это набор архитектурных идей и принципов для построения сложных информационных систем с пользовательским интерфейсом;
* MVC — это аббревиатура, которая расшифровывается так: Model-View-Controller.

**Model.** Первая компонента/модуль — так называемая модель. Это компонент, который отвечает за обработку данных и бизнес-логики приложения. Он представляет собой набор классов и методов, которые работают с данными и обеспечивают их целостность и безопасность.

**View.** Вторая часть системы — вид. Данный модуль отвечает за отображение данных пользователю. Все, что видит пользователь, генерируется видом.

**Controller.** Третьим звеном данной цепи является контроллер. В нем хранится код, который отвечает за обработку действий пользователя (любое действие пользователя в системе обрабатывается в контроллере), вызывают соответствующие методы модели и передают данные на представление для отображения

Модель — самая независимая часть системы. Настолько независимая, что она не должна ничего знать о модулях Вид и Контроллер. Модель настолько независима, что ее разработчики могут практически ничего не знать о Виде и Контроллере.

Основное предназначение Вида — предоставлять информацию из Модели в удобном для восприятия пользователя формате. Основное ограничение Вида — он никак не должен изменять модель.

Основное предназначение Контроллера — обрабатывать действия пользователя. Именно через Контроллер пользователь вносит изменения в модель. Точнее в данные, которые хранятся в модели.

