Java Servlets и Java Server Pages

...

Jakarta EE, трёхзвенная архитектура приложений, контейнеры сервлетов

Java Enterprise Edition (EE) = Jakarta EE (начиная с 2018)

набор спецификаций (с документами) для языка Java, описывающий **архитектуру серверной платформы для задач средних и крупных предприятий.**

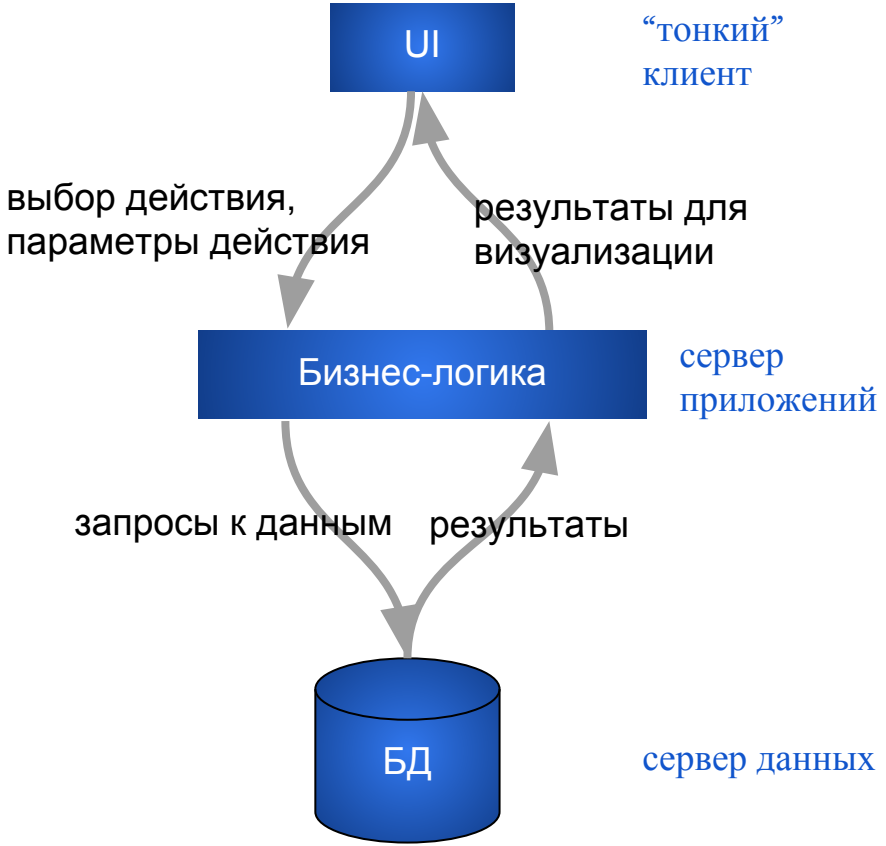
Jakarta EE
что внутри?

Детали:
https://ru.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition

EJB (Enterprise JavaBeans) спецификация технологии серверных компонентов, содержащих бизнес-логику
JPA (Java Persistence API)управление постоянством и объектно-реляционное отображение
Servlet Обслуживание запросов веб-клиентов
JSP (JavaServer Pages) динамическая генерация веб-страниц на стороне сервера
JAX-WS Java API for XML Web Services — создание веб-сервисов
JAX-RS Java API for RESTful Web Services — создание RESTful веб-сервисов
JSON-P Java API for JSON Processing — разбор и генерация JSON
JSON-B Java API for JSON Binding — преобразование Java объектов в/из JSON
JNDI Java Naming and Directory Interface — служба каталогов
JavaMail Получение и отправка электронной почты
JACC Java Authorization Contract for Containers
.....

Трехзвенная архитектура

или 3-tier, multitier



“Тонкий” клиент

- интерфейсный (обычно графический) компонент
- не должен иметь прямых связей с базой данных (по требованиям безопасности)
- не имеет основной бизнес-логики (по требованиям масштабируемости). Допустимые варианты бизнес-логики: интерфейс авторизации, алгоритмы шифрования, проверка вводимых значений на допустимость и соответствие формату, несложные операции (сортировка, группировка, подсчет значений) с данными, уже загруженными на терминал
- не хранит состояние приложения (по требованиям надежности)

Сервер приложений

- бОльшая часть бизнес-логики. Вне его остаются фрагменты, экспортируемые на терминалы (см. “тонкий” клиент), а также погруженные в третий уровень хранимые процедуры и триггеры
- контроль безопасности

Сервер данных

- это СУБД с нужными базами данных, в которых есть

  - таблицы
  - хранимые процедуры (значительно уменьшают нагрузку на каналы связи и улучшают безопасность)
  - триггеры

Сервер приложений и Jakarta EE

Java-сервер приложений ведет себя как расширенная виртуальная машина для запуска приложений, прозрачно управляя соединениями с базой данных с одной стороны и соединениями с веб-клиентом с другой.

Java-сервера приложений

Sun GlassFish (интегрирована с NetBeans IDE),

IBM WebSphere,

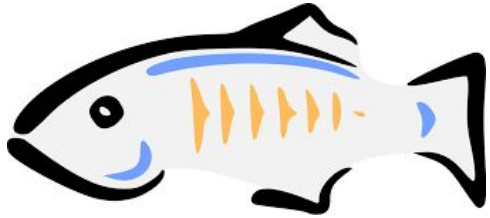
RedHat JBoss Application Server,

Apple WebObjects,

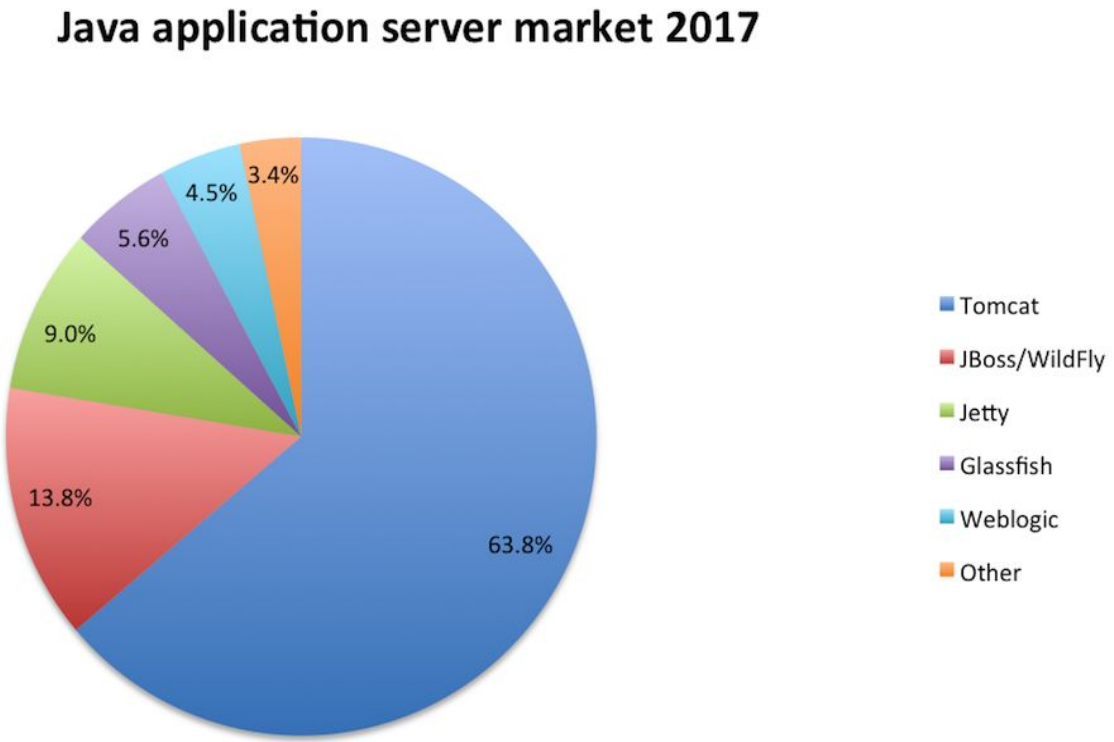
Tomcat/TomEE

Jetty (интегрирована с Eclipse IDE)

Oracle Weblogic

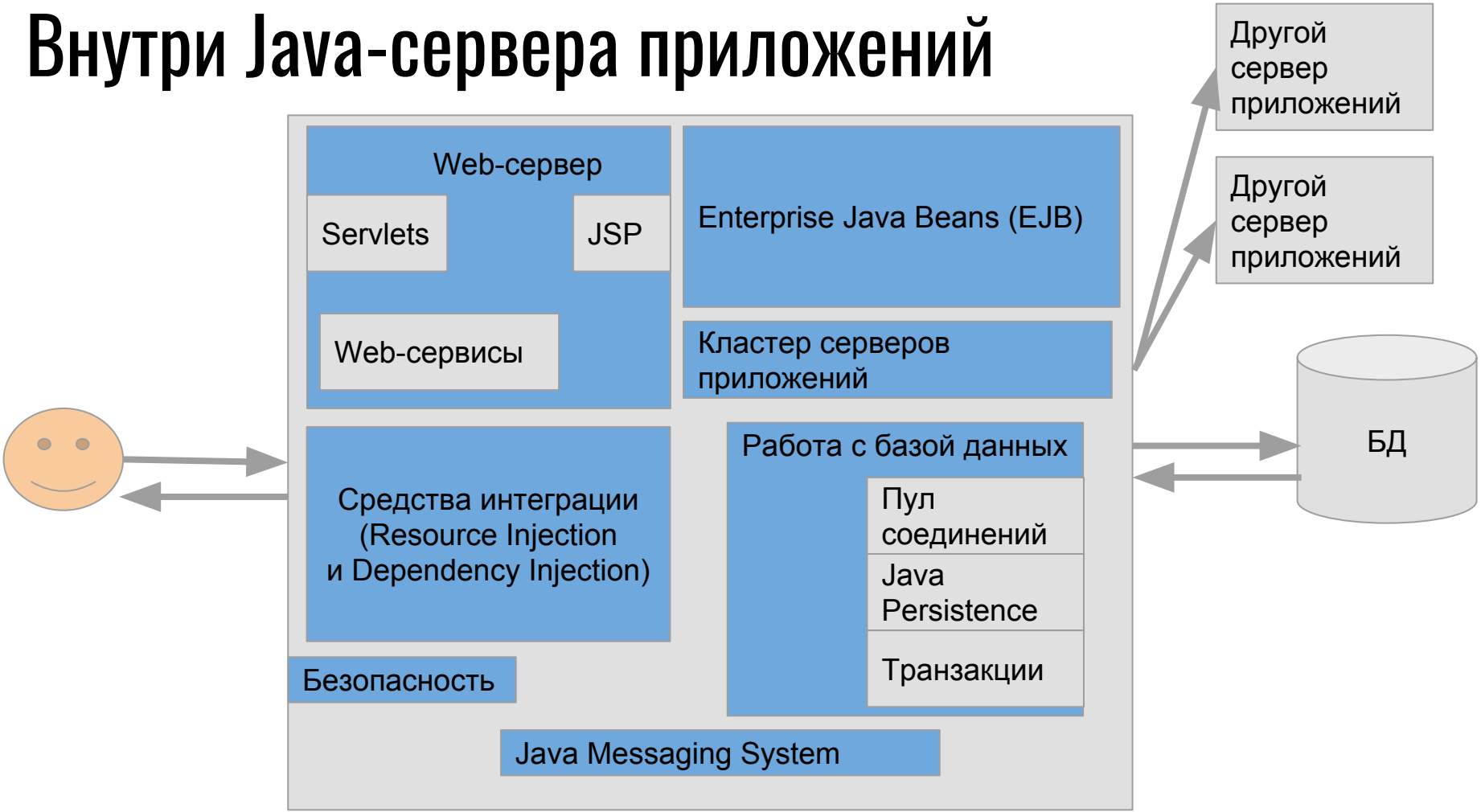


Рынок Java-серверов приложений (на 2017)*



https://plumbr.io/blog/java/most-popular-java-application-servers-2017-edition

Внутри Java-сервера приложений



Архитектура приложения на Java Servlets

Servlet

- специфичный для Java механизм обработки HTTP и FTP запросов
- используется для построения Web-порталов
- используется вместе с Web сервером и/или
—— контейнером сервлетов

Реализация Servlet

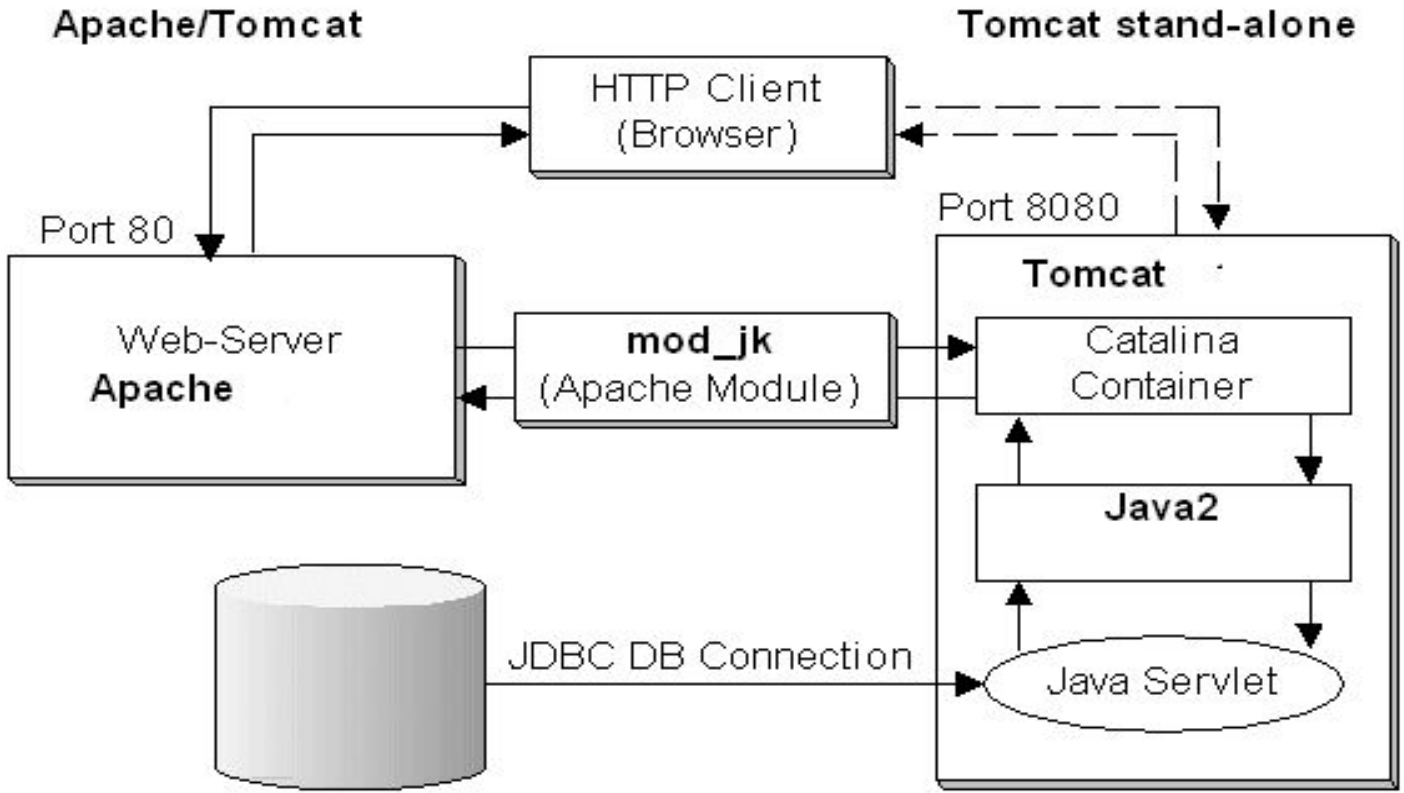
Пакеты
javax.servlet,
javax.servlet.http,
javax.servlet.jsp

Контейнер сервлетов

специальный Web-сервер, который принимает данные, передает на исполнение сервлету, и полученный ответ возвращает клиенту

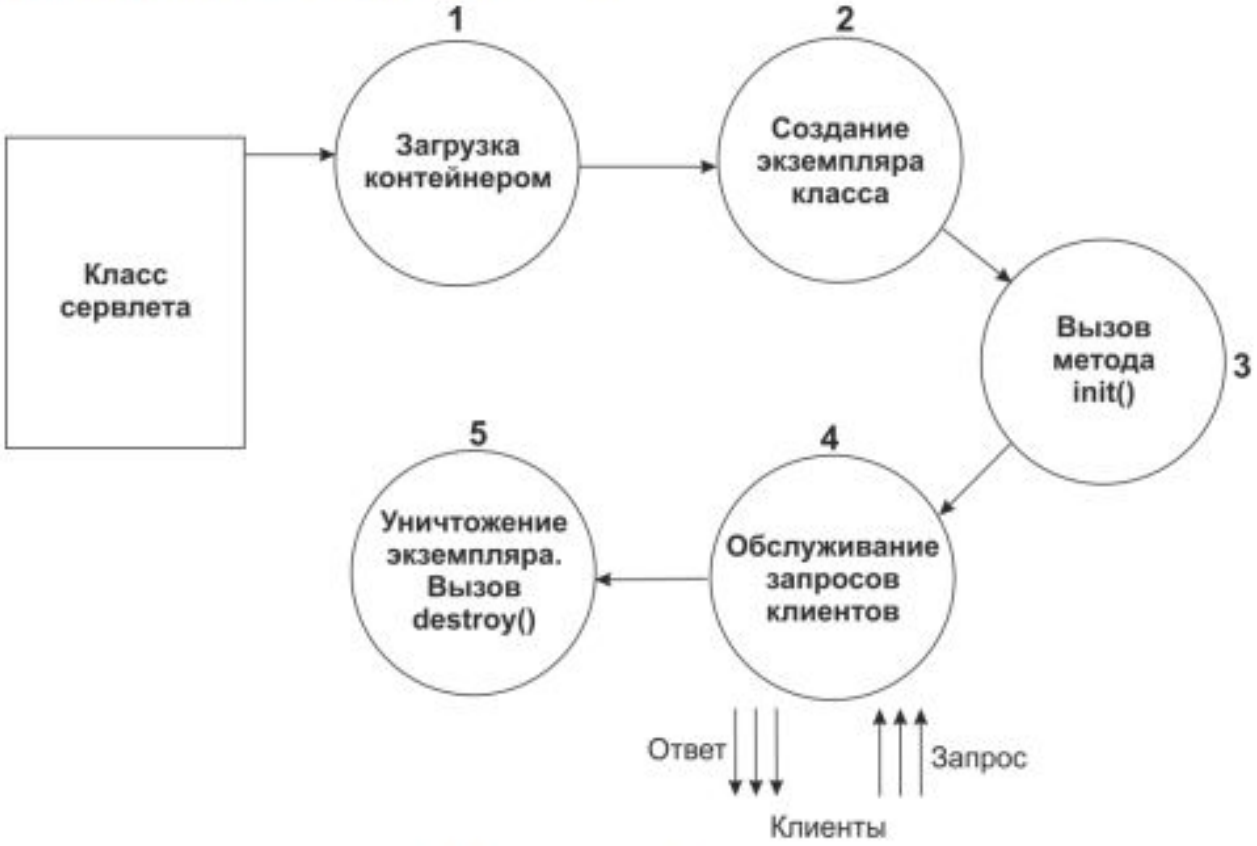
Например, Apache Tomcat

Контейнер сервлетов: в связке с Web-сервером и отдельно



Контейнер сервлетов: несколько сервлетов

Жизненный цикл сервлета



Сценарий жизни обычного сервлета

1) Пользователь вводит URL в браузере. Конфигурационный файл Web-сервера web.xml (= дескриптор доставки сервлета) из папки /WEB-INF/ проекта указывает, что этот URL предназначен для сервлета, управляемого контейнером сервлетов на сервере

http://server-address:port/sample1

2) Если экземпляр сервлета еще не был создан (существует только один экземпляр сервлета для приложения), контейнер загружает класс и создает экземпляр объекта

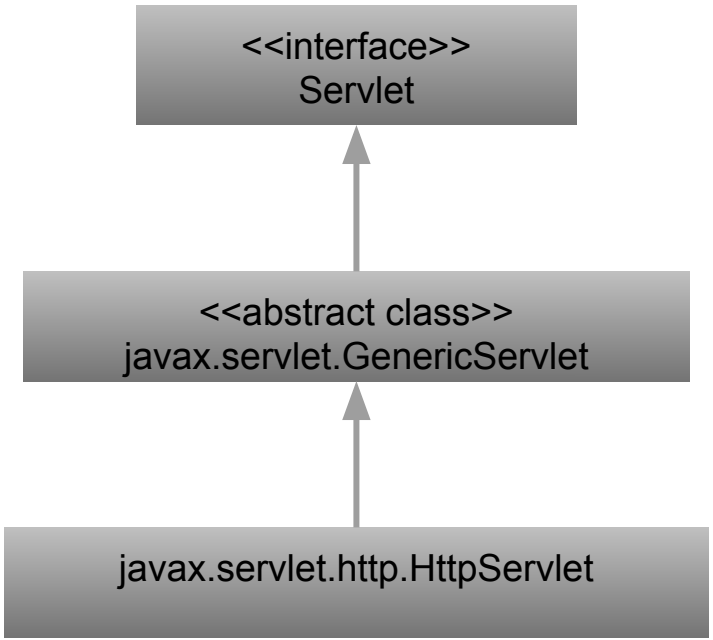
3) Контейнер вызывает метод init() сервлета

4) Контейнер вызывает метод service() сервлета и передает HttpServletRequest и HttpServletResponse

5) Сервлет выполняет свою работу (обращение к базе данных, вычисления и т.д.)

6) контейнер вызывает метод destroy() сервлета

Сервлет в системе классов javax.servlet.*



doGet, if the servlet supports HTTP GET requests

doPost, for HTTP POST requests

doPut, for HTTP PUT requests

doDelete, for HTTP DELETE requests

init and destroy, to manage resources that are held for the life of the servlet

getServletInfo, which the servlet uses to provide information about itself

Любой подкласс HttpServlet **должен переопределить** минимум один из этих методов

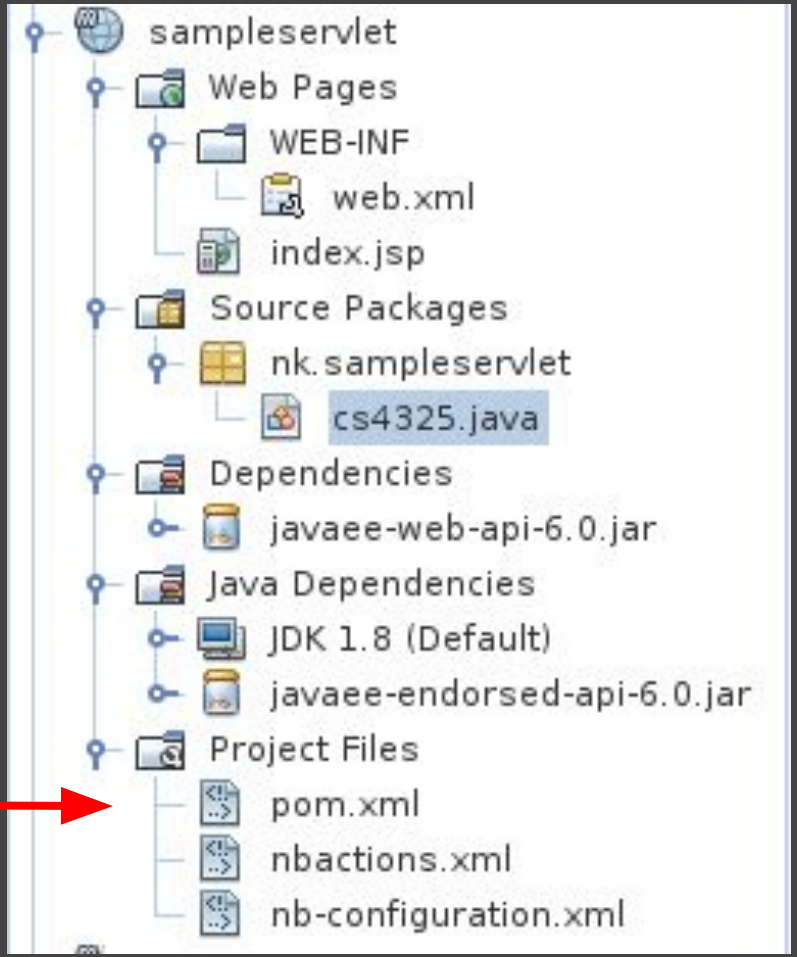
Сервлет: пример

файл SampleServlet.java

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class SampleServlet extends HttpServlet
        {
public MyServlet() { super(); }
public void init() throws ServletException { }
protected void doGet( HttpServletRequest request, HttpServletResponse response)
                    throws ServletException, IOException { ... }
protected void doPost( HttpServletRequest request, HttpServletResponse response)
                    throws ServletException, IOException { ... }
public void destroy() { super.destroy();}
        }
```

Структура проекта с Servlets

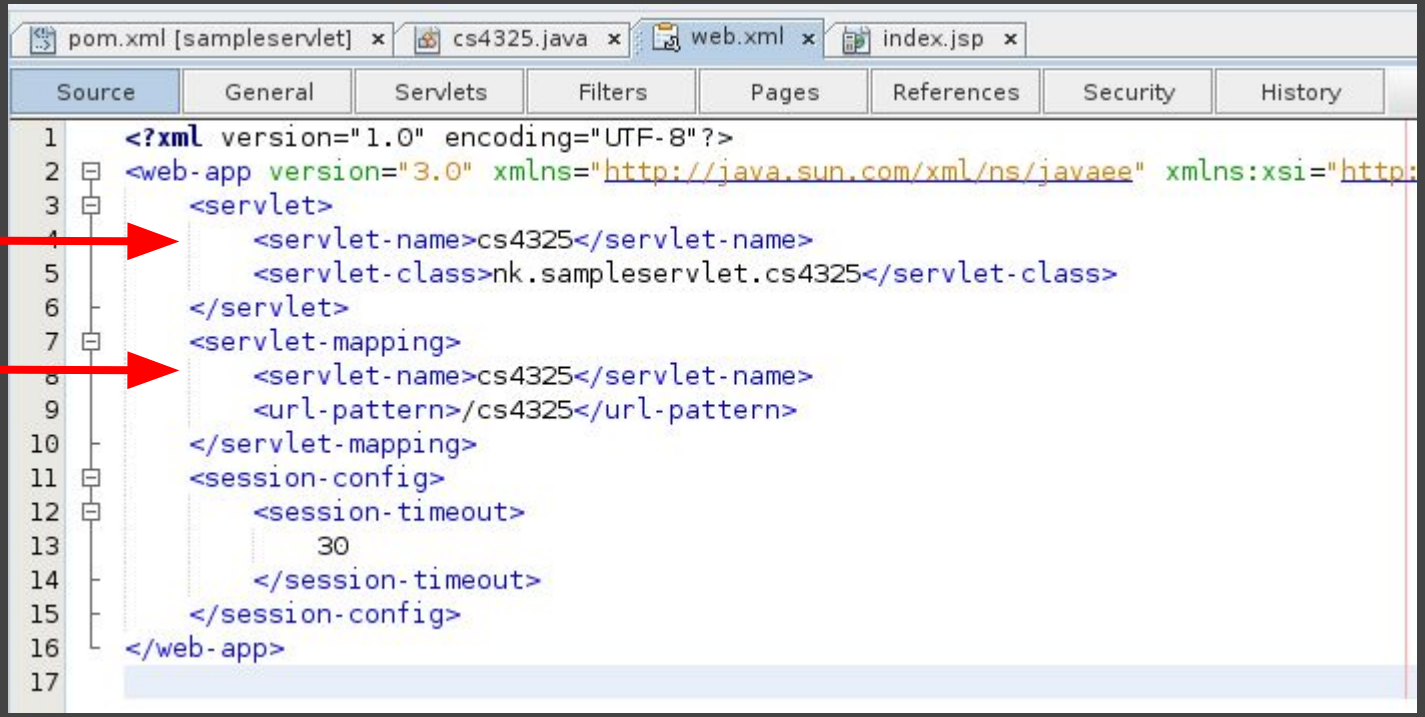


параметры прокси-сервера

проверьте себя!

Что означает каждый “лист” дерева этого проекта?

Файл web.xml


pom.xml [sampleservlet]
cs4325.java
web.xml
index.jsp
Source
General
Servlets
Filters
Pages
References
Security
History
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http:
<servlet>
<servlet-name>cs4325</servlet-name>
<servlet-class>nk.sampleservlet.cs4325</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>cs4325</servlet-name>
<url-pattern>/cs4325</url-pattern>
</servlet-mapping>
<session-config>
<session-timeout>
30
</session-timeout>
</session-config>
</web-app>

Java Server Pages (JSP)

JSP

- технология для создания Web-страниц со статическими и динамическими компонентами

Страница JSP =

{HTML, SVG, WML, или XML} статичные элементы

+

JSP элементы, которые конструируют динамическое содержимое

JSP

JSP страница компилируется в сервлет со статическим содержимым, и подается в поток вывода, связанный с методом service()

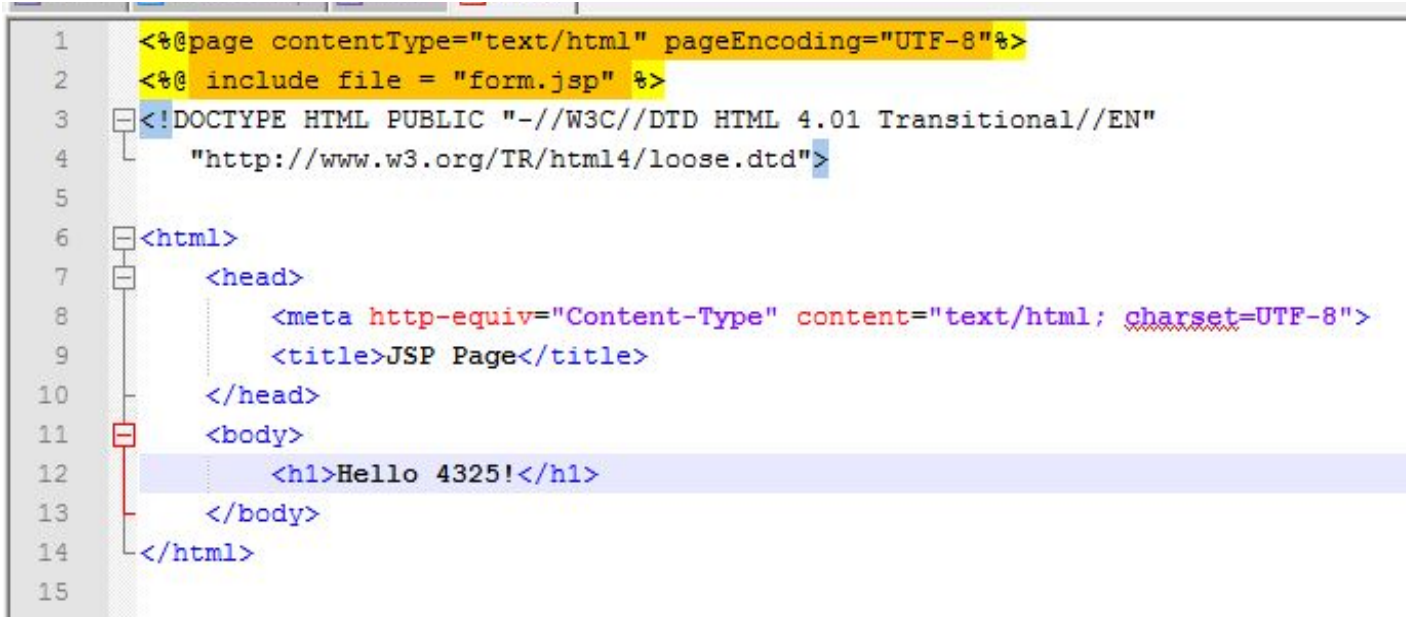
JSP выглядит как HTML, где собственно JSP код помещается в теги <% ... %>

JSP страницы имеют расширение .jsp

Пример 1:
index.jsp

```
1   <%@page contentType="text/html" pageEncoding="UTF-8"%>
2   <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
3      "http://www.w3.org/TR/html4/loose.dtd">
4
5   <html>
6       <head>
7           <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
8           <title>JSP Page</title>
9       </head>
10      <body>
11          <h1>Hello 4325!</h1>
12      </body>
13  </html>
```

Пример 2: index.jsp использует form.jsp



```jsp
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ include file = "form.jsp" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Hello 4325!</h1>
    </body>
</html>

```

Пример 3: form.jsp описывает HTML-форму и обработчик

request - один из неявных объектов на JSP-странице

Пока здесь нет упоминания о сервлетах!

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP form</title>
    </head>
    <body>
        <h1>4325, let us test a form</h1>
        <ul>
         <li><p><b>First Name:</b>
            <%= request.getParameter("first_name")%>
         </p></li>
         <li><p><b>Last  Name:</b>
            <%= request.getParameter("last_name")%>
         </p></li>
       </ul>

<form action = "form.jsp" method = "GET">
            First Name: <input type = "text" name = "first_name">
            <br />
            Last Name: <input type = "text" name = "last_name" />
            <input type = "submit" value = "Submit" />
       </form>

    </body>
</html>
```

JSP: неявные объекты

Неявные объекты JSP (implicit objects) - Java объекты, которые сделаны доступными из JSP для разработчиков

JSP: неявные объекты

request - HttpServletRequest объект, ассоциированный с запросом

response - HttpServletResponse объект, ассоциированный с ответом

out - PrintWriter объект, который используется для отсылки результата клиенту

session - HttpSession объект, ассоциированный с запросом

application - ServletContext объект, ассоциированный с контекстом приложения

config - ServletConfig объект, ассоциированный со страницей.

pageContext - инкапсулирует зависящие от сервера свойства, например, высокопроизводительные JspWriters

page - синоним для this

Exception предоставляет доступ к данным исключительной ситуации

Совместное использование Servlets и JSP

Пример: передача управления с сервлета на JSP-страницу

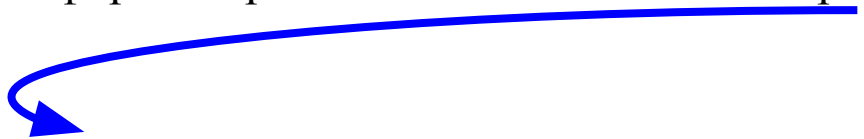
файл form4325.java

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class form4325 extends HttpServlet {
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
                String pageName = "index.jsp"; // default page
                request.getRequestDispatcher(pageName).include(request, response); }
...
@Override
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException { processRequest(request, response); }
...}
```

Пример: передача управления с JSP-страницы на сервлет

Логика:

- создали JSP страницу для UI (HTML-формы + JSP-код)
- создали класс сервлета, в котором происходит обработка параметров с форм (метод передачи - Get/Post/...)
- На JSP странице указали, что форма обрабатывается с помощью сервлета

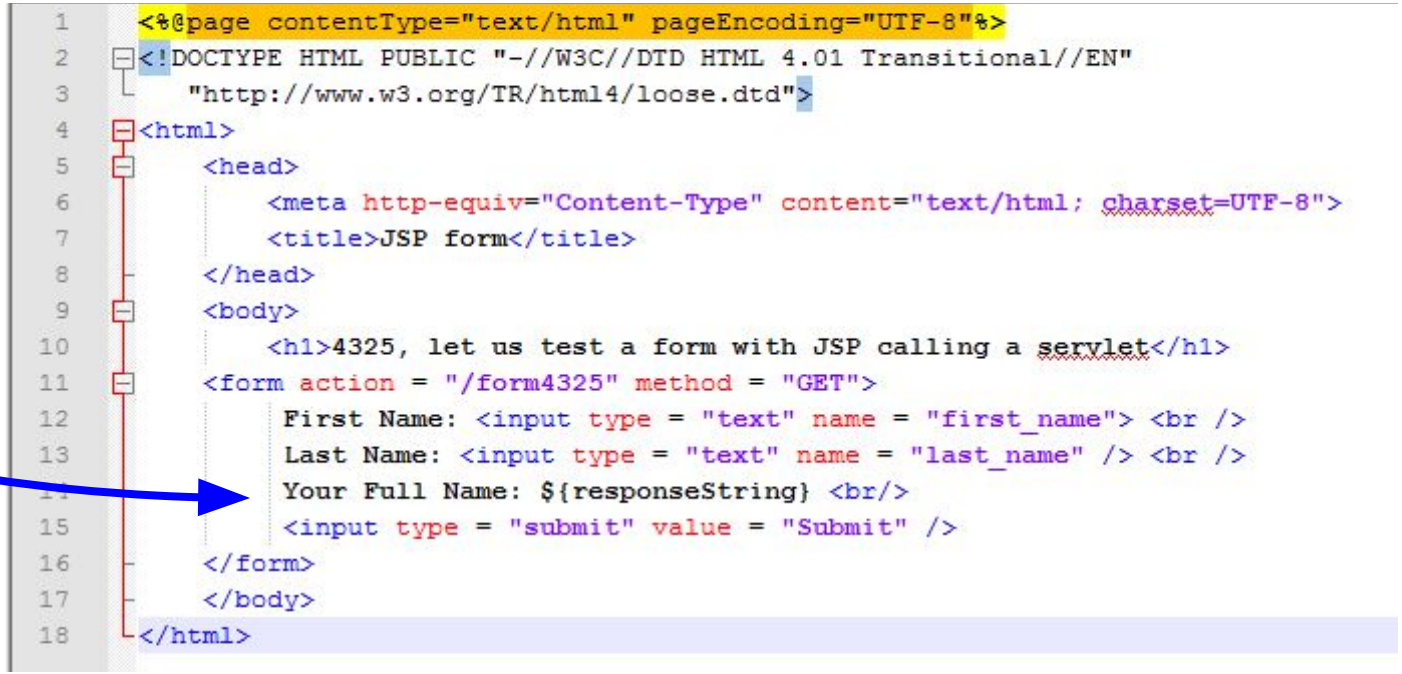


```
<form action="/form4325" method="POST">
        First Name: <input type="text" name="first_name" /> <br />
        Last Name: <input type="text" name="last_name" /> <br />
        <input type='submit' />
</form>
```

Пример: передача управления с JSP-страницы на сервлет

файл form2.jsp

тут ожидается
ответ от сервлета



```jsp
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP form</title>
    </head>
    <body>
        <h1>4325, let us test a form with JSP calling a servlet</h1>
    <form action = "/form4325" method = "GET">
         First Name: <input type = "text" name = "first_name"> <br />
         Last Name: <input type = "text" name = "last_name" /> <br />
         Your Full Name: ${responseString} <br/>
         <input type = "submit" value = "Submit" />
    </form>
    </body>
</html>
```

Пример: передача управления с JSP-страницы на сервлет

```
//imports section
public class form4325 extends HttpServlet {
protected void processRequest(HttpServletRequest request,
                                        HttpServletResponse response)
                                          throws ServletException, IOException {
        String firstname = request.getParameter("first_name");
        String lastname = request.getParameter("last_name");
        String output = firstname + " " + lastname ;
        request.setAttribute("responseString", output);
        // или другие, более сложные преобразования
        request.getRequestDispatcher("form2.jsp").forward(request, response);
    }
...}
```