**Java EE – Java Enterprise Edition**

Это набор спецификаций и соответствующей документации языка Java для написания клиент-серверных приложений, средних и больших проектов.

Структура

webapp -> WEB-INF -> **web.xml** – файл с описанием веб проекта, тут прописывается настройки нашего web проекта.

Обьявление файла – который будет по default открываться

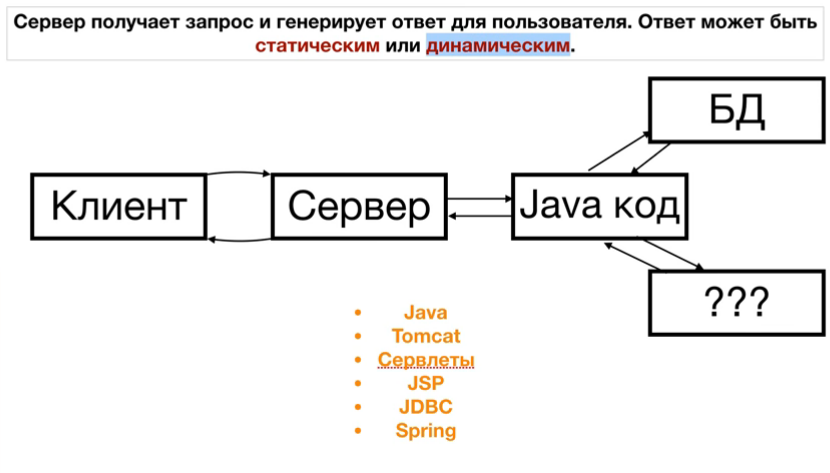
<welcome-file-list>  
 <welcome-file>index.html</welcome-file>  
</welcome-file-list>

**Request и Response**

При **Request** основной параметр который передаём серверу - это метод (даём понять серверу что хотим сделать: GET, POST, DELETE).

При **Response** основные параметры которые приходят с сервера Content-type, Status, Date и сами запрашиваемые данные.

Типы status (200-всё ок, 3xx – редирект, сервер послал запрос нашему браузеру что бы он перешёл на другую страницу, 4xx – ошибка клиента, 5xx-ошибка сервера). Типы содержимого Content-type (text/html, text/css, text/xml, application/json).

Основная структура клиент-серверных приложений:  


**Apache Tomcat**

Это веб сервер приложений, программа которая крутится без остановки, обрабатывает запросы, перераспределяет запросы на обьекты сервлетов, которые загружены и созданы внутри контейнера сервлетов (включён в tomcat).

Есть веб-сервер – принимает String и всё (чистый apache)

Есть сервер веб-приложений – это собственно и есть наш apache tomcat (обрабатывает String и создаёт обьекты HttpServletRequest и HttpServletResponse).

**Servlet**

Сервлет – программа написанная на java, работает по принципу запрос-ответ.

При запуске приложения:

1) контейнер сервлетов создаёт копии всех сервлетов программы

2) вызывает **конструктор** каждого сервлета

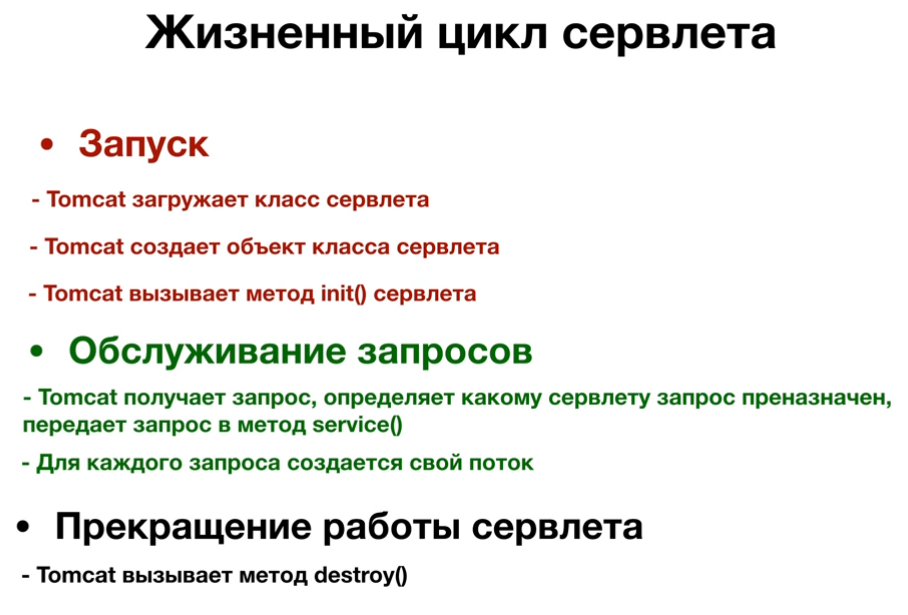
3) вызывается метод **init()** на кадждом сервлете и только 1 раз, при создании обьекта сервлета, этот метод можно переопределить

4) при этом создаются только **по 1 экземпляру** каждого класса сервлета

5) все запросы создают отдельный поток, идут только через 1 конкретный созданный обьект сервлета (не рекомендуется в сервлете использовать глобальные переменные)

6) при каждом запросе вызывается метод service() который определяет к какому методу сервлета пренадлежит запрос (GET,POST and etc).

4) если долгое время к обьекту сервлета не было запросов, контейнер сервлетов выгружает обьект из памяти, перед выгрузкой вызывается метод **destroy()**



Есть 2 способа задать URL для сервлета, через xml и через аннотации.

Через xml:

<servlet>  
 <servlet-name>FirstServlet</servlet-name>  
 <servlet-class>FirstServlet</servlet-class>  
</servlet>  
<servlet-mapping>  
 <servlet-name>FirstServlet</servlet-name>  
 <url-pattern>/hello-world</url-pattern>  
</servlet-mapping>

Через аннотации (над классом сервлета):

@WebServlet("/login")

Задать URL для страницы JSP:

<servlet>  
 <servlet-name>FirstJsp</servlet-name>  
 <jsp-file>/FirstJsp.jsp</jsp-file>  
</servlet>  
<servlet-mapping>  
 <servlet-name>FirstJsp</servlet-name>  
 <url-pattern>/hello-world-jsp</url-pattern>  
</servlet-mapping>

В сервлете HTML код не пишем, грязь, так же считается не лучшим вариантом совмещать это в JSP странице, но лучше чем в сервлете (выход использовать Spring):

public class FirstServlet extends javax.servlet.http.HttpServlet {  
 protected void doGet(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response){  
 PrintWriter pw = response.getWriter();  
 pw.println("<html>");  
 pw.println("<h1> Hello world! </h1>");  
 pw.println("</html>");  
 }  
}

Создают JSP страницы (тут используется html код с вкраплениями Java кода), под капотом всё равно эта JSP преобразуется в сервлет – но зато без лишнего шума и более читабельно, всё что пишем в <% код %> вставляется под капотом в тело метода doGet() или doPost(), тоесть при написания кода считай что находишься в методе

<%@ **page** contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
 <title>First JSP</title>  
</head>  
<body>  
 <h1>Testing JSP</h1>  
 <% //Первый вариант вставки java кода, тут можно ничего не возвращать %>  
 <%= System.*out*.print("Тут обязаны вернуть String"); %>  
</body>  
</html>

Так как классы мы не можем импортировать (в методе) в JSP используем полный путь для подключения классов (либо используем директивы), так же все вкрапления java кода связаны между собой и названия переменных видны в различных блоках <%%>:

<%  
 java.util.Date now = new java.util.Date();  
 String someString = "Текущая дата : " + now;  
 %>  
 <p><%= someString %></p>

Что бы получить в JSP параметры переданные клиентом (request и response иденые):

<% String name = request.getParameter("name"); %>  
<%= "Hello from JSP, " + name%>

Есть возможность получать GET параметры от клиента, в сервлете:

request.getParameter("name");

**Директивы в JSP**

Этоструктура вида <%@ директива имяАтрибута="значение" %> используется всегда вначале страницы, есть 3 типа директив, одна из них page (определяет свойства страницы JSP)

</head>  
 <h1>Testing JSP</h1>  
 <%@ **page** import="java.util.Date, logic.TestClass" %>

<p>  
 <%= new Date() %>  
 </p>  
</html>

**Forward and Redirect**

Логика в методе до выполнения forward или redirect выполняется.

**Forward** только на внутренние ресурсы сервера, при этом в браузере страница не поменяется, браузер даже знать не будет что перешёл на другую внутреннюю cтраницу сервера:

RequestDispatcher dispatcher = request.getRequestDispatcher("/testJsp.jsp");  
dispatcher.forward(request,response);

**Redirect** переведёт клиента на другой ресурс, при этом URL у клиента поменяется (так как клиент получит новую ссылку – куда необходимо перейти и перейдёт)

response.sendRedirect("https://www.google.com");

**Сессия**

В Java хранение сессии пользователя организуем при помощи cookies (время хранения около 30 по default). Cookies это пара ключ-значение. Что бы знать какой пользователь к нам вернулся, так как HTTP не содержит состояний запросов и ничего не знает о пользователе.

Cookies это информация на браузере клиента которую он посылает каждый раз при обращении к серверу. При первом обращении на сервере создаются cookies и передаются клиенту, браузер клиента сохраняет их и при каждом след обращении посылает cookies на сервер как дополнение. Что бы получить/установить cookies используем следующее:

HttpSession session = request.getSession(); //получаем обьект сессии  
 Integer count = (Integer) session.getAttribute("count"); //получаем куку от клиента  
  
 if(count == null){ //если нету куки  
 session.setAttribute("count", 1);  
 count = 1;  
 }else{  
 session.setAttribute("count", count + 1);  
 }  
  
 PrintWriter writer = response.getWriter();  
 writer.println("<html>");  
 writer.println("<p> Your count is " + count + " </p>");  
 writer.println("</html>");

Cookies дли всех сервлетов, jsp страниц единый, делится между всеми (можем получить к одной и той-же сессии из разных сервлетов).

У любой cookies есть: название и значение, время действия (жизни), domain/path – к какому сайту относится куки (при обращении к этому сайту, браузер будет высылать ему куки в которых совпадают эти же поля).

**Заметки по Java EE:**

1) Если создать папку в webapp -> WEB-INF -> lib и занести тут jar (он автоматически подгрузится tomcat).