# Java.EE.01

# SERVLETS

**Author:** **Ihar Blinou**
**Oracle Certified Java Instructor**
**ihar_blinou@epam.com**

# Содержание

1. Основы протокола HTTP
2. Платформа J2EE
3. Структура web-приложения
4. Сервлет. Request. Response
5. MVC
6. JSP (start)
7. Жизненный цикл сервлета и JSP
8. Взаимодействие сервлета и JSP
9. Сессии
10. Cookies
11. Совместное использование ресурсов
12. События
13. Фильтры

# ОСНОВЫ ПРОТОКОЛА HTTP

# Основы протокола HTTP

**HTTP - Hyper Text Transfer Protocol**

- Определяет передачу данных во время жизненного цикла сокета
- Работает в среде запрос-ответ (request-response)
- Путь общения между браузерами и web-серверами

Работа по протоколу HTTP происходит следующим образом: программа-клиент устанавливает TCP-соединение с сервером (стандартный номер порта-80) и выдает ему HTTP-запрос. Сервер обрабатывает этот запрос и выдает HTTP-ответ клиенту.

## Структура HTTP-запроса

- HTTP-запрос состоит из заголовка запроса и тела запроса, разделенных пустой строкой. Тело запроса может отсутствовать.

- Заголовок запроса состоит из главной (первой) строки запроса и последующих строк, уточняющих запрос в главной строке. Последующие строки также могут отсутствовать.

# Основы протокола HTTP

Запрос в главной строке состоит из трех частей, разделенных пробелами:

**Метод** (иначе говоря, команда HTTP):

- **GET** - запрос документа. Наиболее часто употребляемый метод.
- **HEAD** - запрос заголовка документа. Отличается от GET тем, что выдается только заголовок запроса с информацией о документе. Сам документ не выдается.
- **POST** - этот метод применяется для передачи данных скриптам. Сами данные следуют в последующих строках запроса в виде параметров.
- **PUT** - разместить документ на сервере. Запрос с этим методом имеет тело, в котором передается сам документ.

# Основы протокола HTTP

**Ресурс** –

это путь к определенному файлу на сервере, который клиент хочет получить (или разместить - для метода PUT). Если ресурс - просто какой-либо файл для считывания, сервер должен по этому запросу выдать его в теле ответа. Если же это путь к какому-либо скрипту, то сервер запускает скрипт и возвращает результат его выполнения. Кстати, благодаря такой унификации ресурсов для клиента практически безразлично, что он представляет собой на сервере.

# Основы протокола HTTP

**Версия протокола** –

версия протокола HTTP, с которой работает клиентская программа.

Таким образом, простейший HTTP-запрос может выглядеть следующим образом:

**GET / HTTP/1.0**

- здесь запрашивается корневой файл из корневой директории web-сервера.

# Основы протокола HTTP

Строки после главной строки запроса имеют следующий формат:

**параметр:значение**

Таким образом задаются параметры запроса. Это является необязательным, все строки после главной строки запроса могут отсутствовать; в этом случае сервер принимает их значение по умолчанию или по результатам предыдущего запроса (при работе в режиме Keep-Alive).

# Основы протокола HTTP

Некоторые наиболее употребительные параметры HTTP-запроса:

- **Connection** (соединение) – может принимать значения ***Keep-Alive*** и *close*. Keep-Alive ("оставить в живых") означает, что после выдачи данного документа соединение с сервером не разрывается, и можно выдавать еще запросы. Большинство браузеров работают именно в режиме Keep-Alive, так как он позволяет за одно соединение с сервером "скачать" html-страницу и рисунки к ней. Будучи однажды установленным, режим Keep-Alive сохраняется до первой ошибки или до явного указания в очередном запросе Connection: close. close ("закрыть") - соединение закрывается после ответа на данный запрос.

# Основы протокола HTTP

- **User-Agent** - значением является "кодовое обозначение" браузера, например:

  Mozilla/4.0 (compatible; MSIE 5.0; Windows 95; DigExt)

- **Accept** - список поддерживаемых браузером типов содержимого в порядке их предпочтения данным браузером, например для моего IE5:

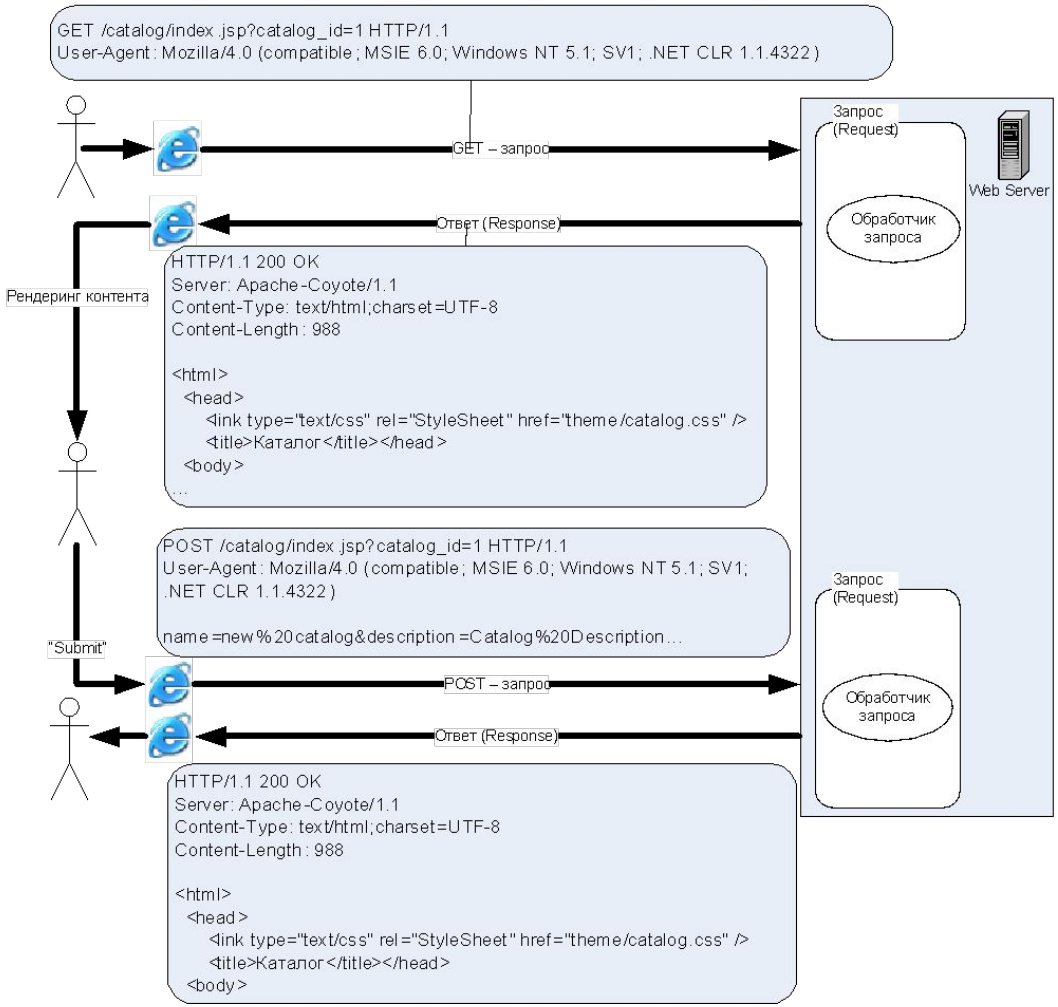
  Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/msword, application/vnd.ms-powerpoint, */*

Это нужно для случая, когда сервер может выдавать один и тот же документ в разных форматах.

# Основы протокола HTTP

- **Referer** - URL, с которого перешли на этот ресурс.
- **Host** - имя хоста, с которого запрашивается ресурс. Полезно, если на сервере имеется несколько виртуальных серверов под одним IP-адресом. В этом случае имя виртуального сервера определяется по этому полю.
- **Accept-Language** - поддерживаемый язык. Имеет значение для сервера, который может выдавать один и тот же документ в разных языковых версиях.

# Основы протокола HTTP



# ПЛАТФОРМА J2EE

# Платформа J2EE

**Java 2 Platform, Enterprise Edition (J2EE)** определяет стандарт для разработки многоуровневых корпоративных приложений.

**Java 2 Enterprise Edition** - это комплекс взаимодействующих Java-технологий, базирующихся на спецификациях, разработанных фирмой Sun Microsystems (http://java.sun.com/j2ee/), представляющих стандарт разработки серверных приложений уровня предприятия.

# Платформа J2EE

**J2EE** - это не конкретный продукт, а **набор спецификаций**, устанавливающих правила, которых следует придерживаться поставщикам конкретной реализации платформы J2EE, а также разработчикам корпоративных приложений.

Цифра "2" в названии спецификации связана с тем, что все технологии, охватываемые спецификациями J2EE, базируются на инструментальном комплекте поддержки разработок в среде Java - JDK (Java Development Kit) версии 1.2 и старше.

# Платформа J2EE

Технологии J2EE ориентированы на разработку серверной стороны приложения и облегчают, в первую очередь, процесс эффективной реализации среднего уровня (Middle tier), содержащего бизнес-логику.

Стандарт Java EE расширяет платформу Java Standard Edition (Java SE) добавляя поддержку веб-сервисов, модели бизнес компонентов, API управления, коммуникационных протоколов, распределенных и веб-приложений. Веб-приложения являются лишь частью того, что описывает Java EE.

# Платформа J2EE

Стандартный Java EE сервер приложений должен поддерживать такие технологии как EJB (сервер и контейнер), JNDI, JMS (Java Message Service), JTA (Java Transaction API), архитектуру J2EE Connector.

Java EE задает контейнеры для серверных приложений, сервлетов, EJB компонентов.

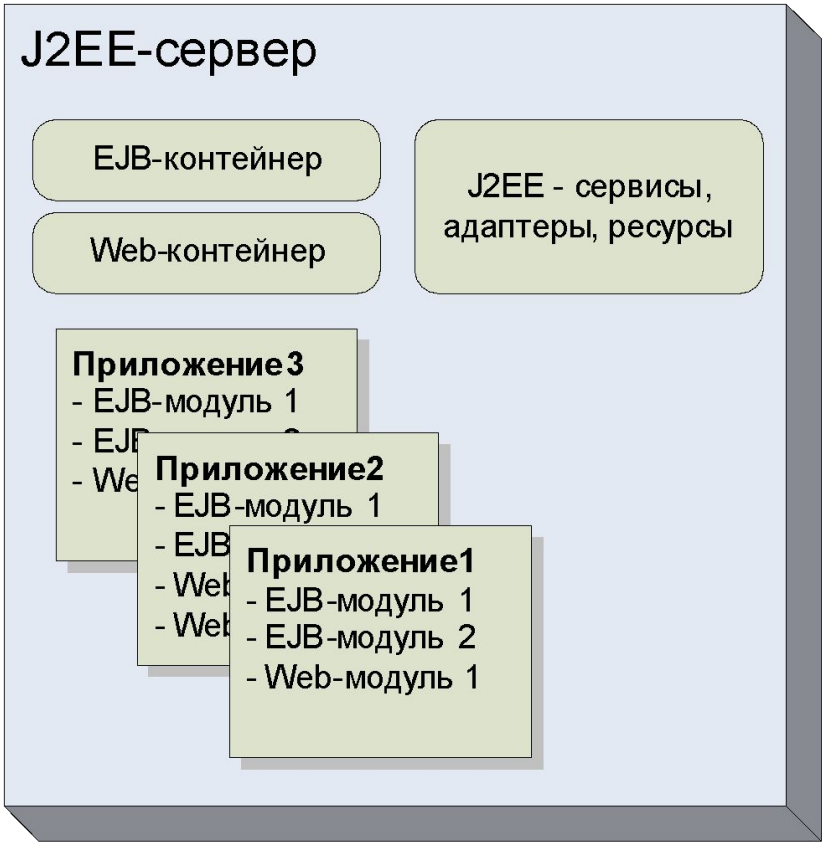
Такие контейнеры предоставляют функциональность, позволяющую устанавливать, сохранять и выполнять поддерживаемые компоненты.

# Платформа J2EE

Java EE также предоставляет стандартную архитектуру для взаимодействия Java EE приложений и серверов приложений с гетерогенными корпоративными информационными системами, такими как ERP-системы, мейнфреймы, системы баз данных и т.д.

Архитектура Java EE Connector предоставляет средства взаимодействия с корпоративными системами используя стандартный интерфейс, известный как адаптер ресурсов (Resource adapter).

# Платформа J2EE




## Сервис:

Java Naming Directory (JNDI)

– универсальный сервис хранения объектов в иерархической структуре имен (аналогично файловой системе)

## Ресурс:

DataSource

- объект, позволяющий приложению получить доступ к соединению к БД

# Платформа J2EE

Платформа J2EE использует модель многоуровневого распределенного приложения.

Логически приложение разделено на компоненты в соответствии с их функциональностью.

Различные компоненты, составляющие J2EE-приложение, установлены на различных компьютерах (слоях) в зависимости от их уровня в многоуровневой среде J2EE, которой данный компонент принадлежит.

- Компоненты клиентского уровня работают на клиентской машине.
- Компоненты Web-уровня работают на J2EE-сервере.
- Компоненты бизнес-уровня работают на J2EE-сервере.
- Программное обеспечение уровня корпоративной информационной системы (EIS) работает на EIS-сервере.

# Платформа J2EE

Приложение **Java EE** может состоять из нескольких компонентов, таких как, например, **EJB, веб-модули, адаптеры ресурсов**, клиентские модули J2EE.

Каждый из компонентов может иметь свой дескриптор развертывания – XML файл, описывающий компонент.

Для разворачивания Java EE компонентов применяется файл в формате «Java архив» (JAR), который расширяется несколькими дополнительными форматами в зависимости от того или иного типа компонента. Формат JAR основан на формате ZIP.

# Платформа J2EE

**JAR файл** может содержать Java классы, XML файлы, вспомогательные ресурсы, статические HTML файлы и другие.

Стандартное Java веб-приложение разворачивается в WAR (Web Application Archive) файле, который является JAR файлом с расширением WAR.

Стандартное Java EE приложение разворачивается в EAR (Enterprise Application Archive) файле, который является JAR файлом с расширением EAR.

# Платформа J2EE

**WAR файл** - это специализированный JAR файл, содержащий такие компоненты веб-приложения как сервлеты, JSP файлы, HTML файлы, дескрипторы развертывания, библиотеки классов и тому подобное.

WAR файл может быть развернут на веб-сервере, таком как Tomcat.

# Платформа J2EE

**EAR файл** – это специализированный JAR файл, содержащий компоненты Java EE приложения, такие как веб-приложения (WAR), EJB, адаптеры ресурсов и так далее.

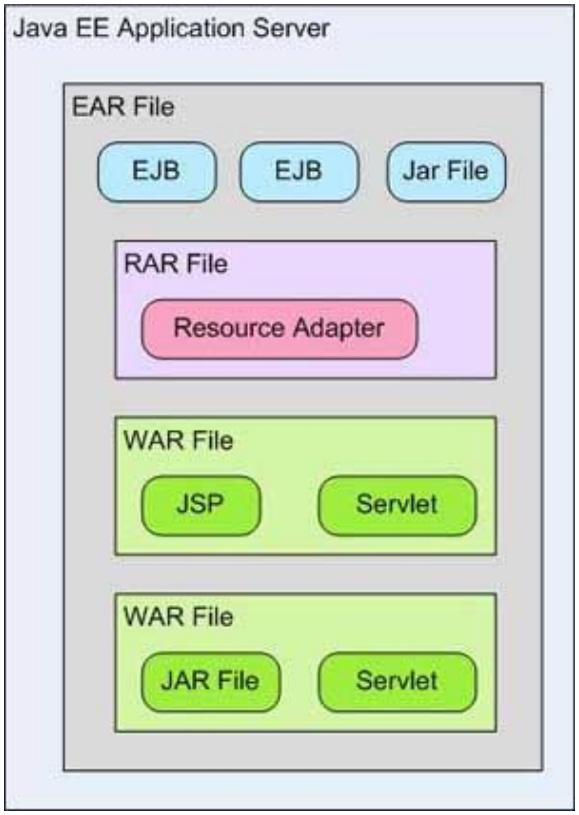
Файл EAR может быть развернут на сервере приложений Java EE, таком как JBoss, WebLogic или WebSphere.

Сервер приложений Java EE загружает EAR файлы в процессе выполнения и устанавливает обнаруженные в нем компоненты на основе инструкций, описанных в дескрипторе развертывания каждого компонента.

# СТРУКТУРА WEB-ПРИЛОЖЕНИЯ

# Структура web-приложения

Компоненты типичного Java EE приложения



# Структура web-приложения

Традиционно, **корпоративное Java EE приложение** определяется как набор следующих компонентов и технологий:

- EAR файлы
- Java сервлеты
- JavaServer Pages или JavaServer Faces
- Enterprise JavaBeans (EJB)
- Сервисы аутентификации и авторизации (JAAS)
- Архитектура J2EE Connector
- JavaBeans Activation Framework (JAF)
- JavaMail
- Java Message Service (JMS)
- Java Persistence API (JPA)
- Java Transaction API (JTA)
- The Java Management Extensions (JMX) API
- Java API for XML Processing (JAXP)
- The Java API for XML-based RPC (JAX-RPC)
- The Java Architecture for XML Binding (JAXB)

# Структура web-приложения

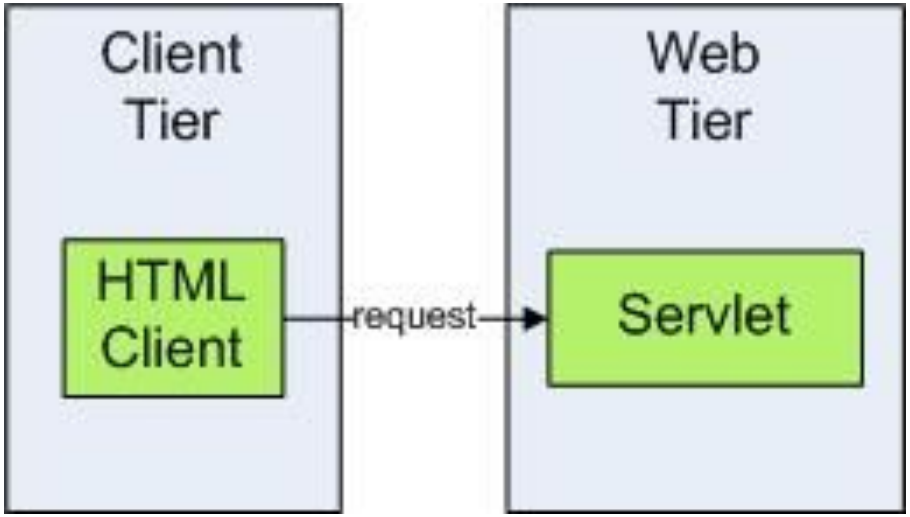
В свою очередь, **веб-приложение Java** объединяет подмножество компонентов и технологий корпоративного приложения, таких как:

- WAR файлы
- Java сервлеты
- JavaServer Faces или JavaServer Pages
- Java Database Connectivity (JDBC) framework

Популярные Фреймворки, такие как Apache Struts, Spring, Hibernate и другие стирают традиционную грань между корпоративными приложениями и веб-приложениями. Каждый фреймворк предлагает свою точку зрения на то, как решить ту или иную задачу. Каждый фреймворк пытается более эффективно решить те или иные сложности, возможно используя набор дополнительных технологий.

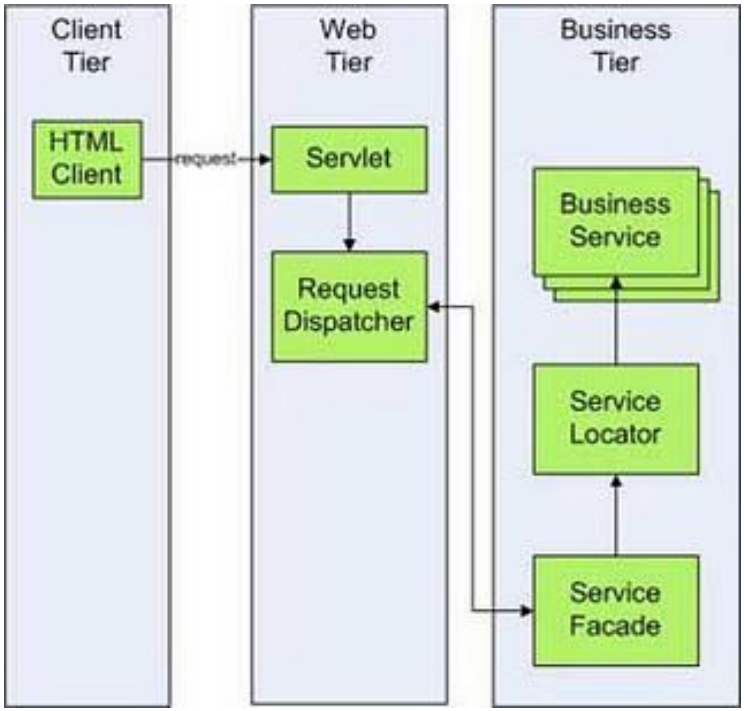
# Структура web-приложения

В типичном Java EE веб-приложении HTML клиент отправляет запрос серверу, где этот запрос обрабатывается веб-контейнером сервера приложений. Веб-контейнер вызывает сервлет, который сконфигурирован для обработки определенного контекста запроса.



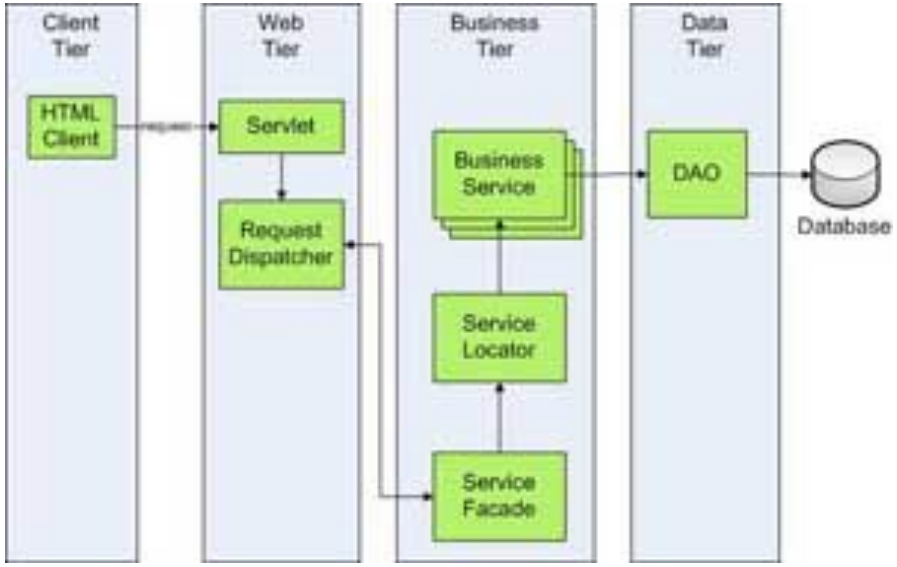
# Структура web-приложения

Как только сервлет получает начальный запрос, инициируется выполнение определенной бизнес логики для завершения запроса. При этом вызывается один или несколько бизнес сервисов или компонентов



# Структура web-приложения

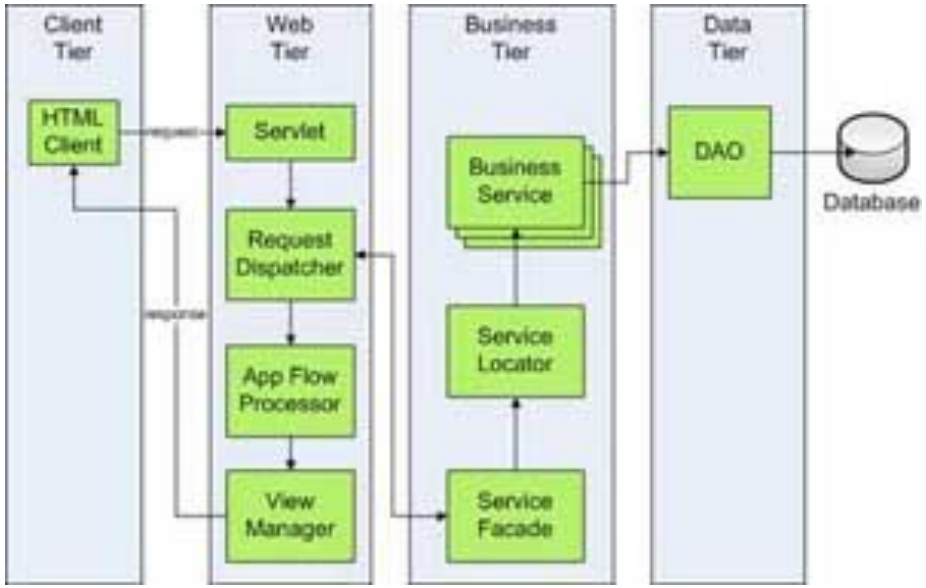
Некоторым бизнес сервисам или компонентам требуется доступ к хранилищам данных или информационным системам. Как правило, для этих целей вводится дополнительный слой абстракции между бизнес-сервисами и хранилищем данных, что позволяет сделать максимально безболезненным возможное изменение источника данных в будущем. При этом объекты доступа к данным (DAO) представляются отдельными компонентами



# Структура web-приложения

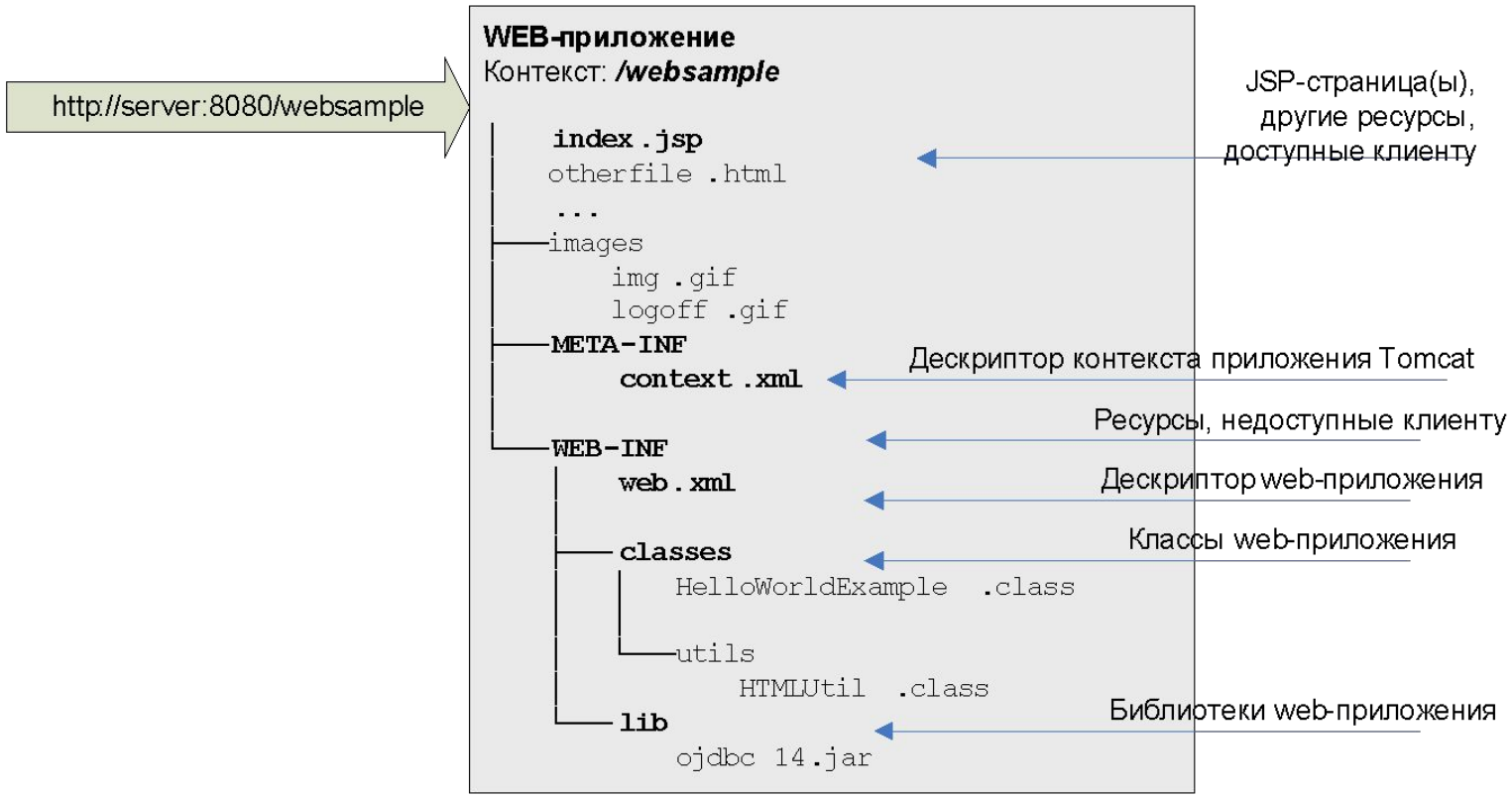
Когда обработка запроса на сервере завершена, HTML клиенту возвращается ответ в определенном формате.

Этот сценарий может быть реализован с использованием ограниченного набора Java EE технологий, таких как, например, сервлеты, JDBC, JSP и тому подобные. Tomcat как и другие веб-серверы, предоставляющие среду выполнения сервлетов и JSP, вполне могут быть использованы как серверы приложений для данного сценария.



# Структура web-приложения

## Структура J2EE Web-приложения



# Структура web-приложения

Web-приложение должно включать:

- Основную (базовую)директорию [имя директории является именем приложения]
- WEB-INF директорию
- *web.xml* файл конфигурации [Deployment Descriptor (дескриптор развертывания)]

Web-приложение может включать:

- Servlets [located in WEB-INF\classes dir]
- JSP files
- HTML files
- Documents, Images
- Jars and other classes [usually in a *lib* directory]

# Структура web-приложения

Базовая структура веб-приложения должна включать **корневую директорию, WEB-INF директорию** и **дескриптор развертывания web.xml**.

Название корневого каталога будет частью URL, указывающего на один из содержащихся ресурсов, и используемый в качестве имени приложения.

Например, вызов *index.html* файла, расположенного в корне каталога '*mysite*', может быть сделан как:

http://localhost:8080/**mysite** или
http://localhost:8080/**mysite**/index.html

# Структура web-приложения

**Web.xml конфигурационный файл** используется для:

- Объявление классов servlet  и JSPs
- Отображения servlets и JSPs в URL шаблоны
- Определения welcom-страниц
- Установления безопасности содержимого, ролей и методов аутентификации

# Структура web-приложения

Любой класс, который загружен и выполнен в веб-контейнерах, должен быть расположен в WEB-INF\CLASSES.

Это могут быть:

- Servlets
- Java Beans (used in JSP)
- Tag libraries classes (used in JSP)
- Helper classes

Все классы, расположенные в директории classes, могут быть собраны в пакеты.

# Структура web-приложения

Другие файлы как JSPs и статическое содержание могут быть расположены где угодно в соответствии с корневой директорией, но местоположение будет отражено в URL, используемом, чтобы вызвать их.

Например, если index.html страница расположена в mysite\pages\ каталоге, URL должен быть похож на:

http://localhost:8080/mysite/**pages/**index.html

# Структура web-приложения

Jars и другие java классы, используемые в приложении, могут быть помещены куда угодно в соответствии с корнем директории или даже вне приложения (обычно в lib).

# СЕРВЛЕТ. REQUEST. RESPONSE

# Сервлет. Request. Response

Сервлеты – это компоненты приложений Java Enterprise Edition, выполняющиеся на стороне сервера, способные обрабатывать клиентские запросы и динамически генерировать ответы на них.

Наибольшее распространение получили сервлеты, обрабатывающие клиентские запросы по протоколу HTTP.

Сервлеты можно внедрять в различные сервера, так как API сервлета, который вы используете для его написания, ничего не «знает» ни о среде сервера, ни о его протоколе.

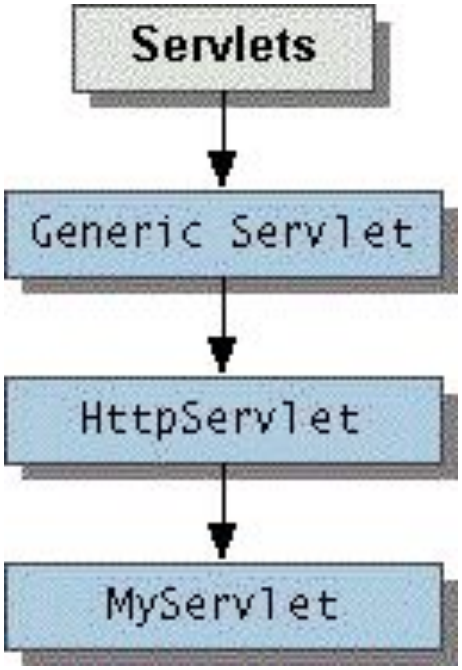
Множество Web-серверов поддерживает технологию Java Servlet.

# Сервлет. Request. Response

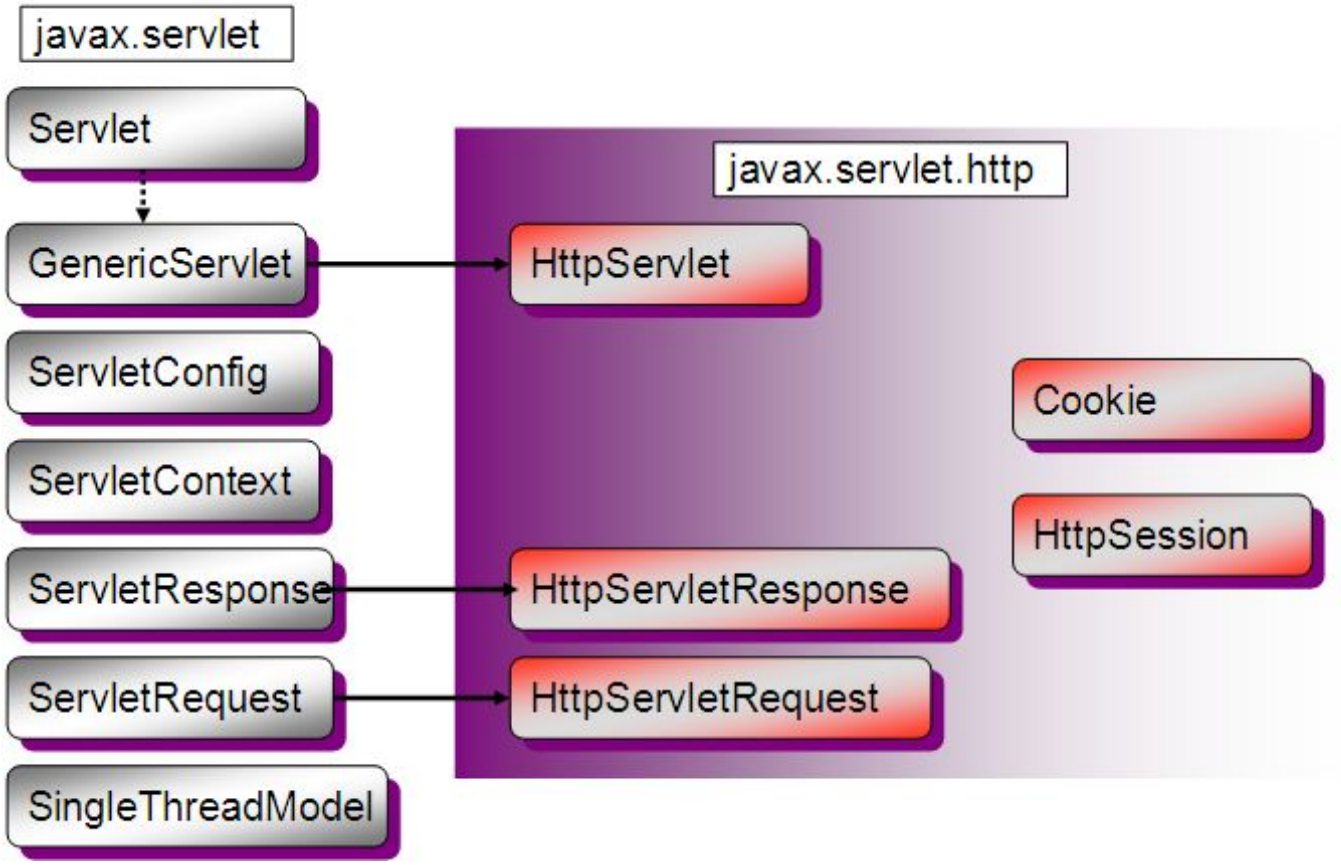
Пакет **javax.servlet** обеспечивает интерфейсы и классы для написания сервлетов.

Центральной абстракцией API сервлета является интерфейс **Servlet**. Все сервлеты реализуют данный интерфейс напрямую, но более распространено расширение класса, реализующего его, как **HttpServlet**.

Интерфейс **Servlet** объявляет, но не реализует методы, которые управляют сервлетом и его взаимодействием с клиентами.



# Сервлет. Request. Response

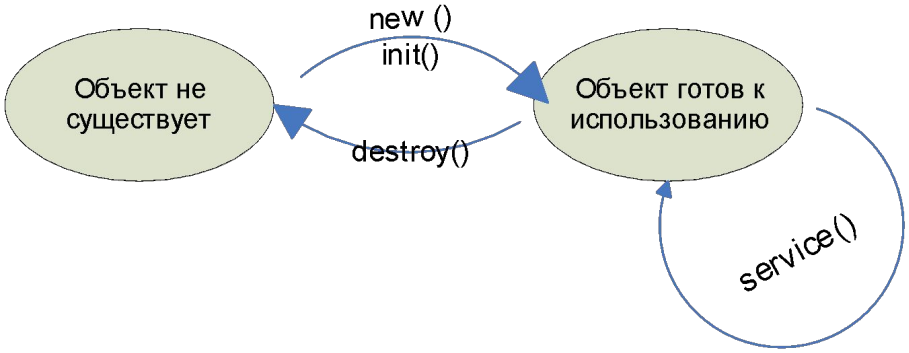


# Сервлет. Request. Response

Принимая запрос от клиента, сервлет получает два объекта:

1. **ServletRequest**, который инкапсулирует связь клиента с сервером.
2. **ServletResponse**, который инкапсулирует обратную связь сервлета с клиентом.

**ServletRequest** и **ServletResponse** – это интерфейсы, определенные пакетом **javax.servlet**.



# Сервлет. Request. Response

Интерфейс **ServletRequest** дает сервлету доступ к:

1. Информации, такой, как имена параметров, переданных клиентом, протоколы (схемы), используемые клиентом и имена удаленного хоста, создавшего запрос и сервера который их получает.
2. Входному потоку **ServletInputStream**. Сервлеты используют входной поток для получения данных от клиентов, которые используют протоколы приложений, такие как HTTP POST и PUT методы.

Интерфейсы, которые расширяют интерфейс **ServletRequest**, позволяют сервлету получать больше данных, характерных для протокола. К примеру, интерфейс **HttpServletRequest** содержит методы для доступа к главной информации по протоколу HTTP.

# Сервлет. Request. Response

Интерфейс **ServletResponse** дает сервлету методы для ответа на запросы клиента.

Он:

1. Позволяет сервлету устанавливать длину содержания и тип MIME ответа (метод **setContentType(String type)**).
2. Обеспечивает исходящий поток **ServletOutputStream** и **Writer**, через которые сервлет может отправлять ответные данные.

Интерфейсы, которые расширяют интерфейс **ServletResponse**, дают сервлету больше возможнойстей для работы с протоколами. Например, интерфейс **HttpServletResponse** содержит методы, которые позволяют сервлету манипулировать информацией заголовка HTTP.

# Сервлет. Request. Response

**HttpServlet** - запросы и ответы

Методы класса **HttpServlet**, которые управляют клиентскими запросами принимают два аргумента:

1. Объект **HttpServletRequest**, который инкапсулирует данные от клиента
2. Объект **HttpServletResponse**, который инкапсулирует ответ к клиенту

## Объект HttpServletRequest

Объекты **HttpServletRequest** предоставляют доступ к данным HTTP заголовка и позволяют получить аргументы, которые клиент направил вместе с запросом.

Непосредственно в интерфейсе **HttpServletRequest** объявлен ряд методов, позволяющих манипулировать содержимым запросов:

- **String getParameter(String name)** - возвращает величину именованных параметров.
- **String[] getParameterValues(String name)** - возвращает массив величин именованного параметра. Используется, если параметр может иметь более чем одну величину.

# Сервлет. Request. Response

- **Enumeration getParameterNames()** - предоставляет имена параметров.
- **String getQueryString()** - возвращает строковую (String) величину необработанных данных клиента для HTTP запросов GET.
- **BufferedReader getReader()** - возвращает объект BufferedReader (текстовая информация) для считатывания необработанных данных (для HTTP запросов POST, PUT, и DELETE).
- **ServletInputStream getInputStream()** - возвращает объект ServletInputStream (двоичная информация) для считывания необработанных данных (для HTTP запросов для POST, PUT, и DELETE).

# Сервлет. Request. Response

Объект **HttpServletResponse**

Объект **HttpServletResponse** обеспечивает два способа возвращения данных пользователю:

- Метод **getWriter** возвращает Writer
- Метод **getOutputStream** возвращает ServletOutputStream

Используйте метод **getWriter** для возвращения пользователю *текстовых данных* и метод **getOutputStream** для *бинарных*. Закрытие Writer или ServletOutputStream после отправления запроса позволит серверу определить, что ответ готов.

# Сервлет. Request. Response

Интерфейс **HttpServletResponse**

- **void sendError(int sc, String msg)** – сообщение о возникших ошибках, где sc – код ошибки, msg – текстовое сообщение;
- **void setDateHeader(String name, long date)** – добавление даты в заголовок ответа;
- **void setHeader(String name, String value)** – добавление параметров в заголовок ответа. Если параметр с таким именем уже существует, то он будет заменен.

## Данные HTTP заголовка

Прежде чем получить доступ к объектам **Writer** или **OutputStream** необходимо установить данные HTTP заголовка.

Класс **HttpServletResponse** предоставляет методы для доступа к данным заголовка:

- **setContentType(String type)** - устанавливает тип содержимого (Content-type).

```
response.setContentType("text/html");
```

# Сервлет. Request. Response. Example 01

```
package _java._ee._01.firstservlet;

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

public class FirstServlet extends HttpServlet {

	private static final long serialVersionUID = 1L;

	public FirstServlet() {
		super();
	}

	protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		PrintWriter out = response.getWriter();
		String title = "Simple Servlet Output";
		// сначала установите тип содержания и другие поля заголовков ответа
```

# Сервлет. Request. Response. Example 01

```
            response.setContentType("text/html");
            out.println("<HTML><HEAD><TITLE>");
            out.println(title);
            out.println("</TITLE></HEAD><BODY>");
            out.println("<H1>" + title + "</H1>");
            out.print("<P>This is ");
            out.print(this.getClass().getName());
            out.print(", using the GET method");
            out.println("</BODY></HTML>");
            out.close();
      }
      protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
            doGet(request, response);
      }
}
```

# Сервлет. Request. Response

Сервлет расширяет класс **HttpServlet** и переопределяет метод **doGet**.

Чтобы ответить клиенту, метод **doGet** в примере использует **Writer** из объекта **HttpServletResponse** для возвращения клиенту текстовых данных. Прежде чем получить доступ к writer, пример устанавливает заголовок **content-type**. В конце метода **doGet**, после того как был отправлен запрос, **Writer** закрывается.

Для работы с конкретным HTTP-методом передачи данных достаточно расширить свой класс от класса **HttpServlet** и реализовать один из этих методов. Метод **service()** переопределять не надо, в классе HttpServlet он только определяет, каким HTTP-методом передан запрос клиента, и обращается к соответствующему методу **doXxx()**.

# Сервлет. Request. Response

Для того, чтобы сервлет работал на сервере, необходимо подготовить файл описания сервлета (XML-файл).

**Элементы web.xml**

**<servlet>** - блок, описывающий сервлеты
- **<display-name>** - название сервлета
- **<description>** - текстовое описание сервлета
- **<servlet-name>** - имя сервлета
- **<servlet-class>** - класс сервлета
- **<init-param>** - блок, описывающий параметры инициализации сервлета
  - **<param-name>** - название параметра
  - **<param-value>** - значение параметра

# Сервлет. Request. Response

**<servlet-mapping>** - блок, описывающий соответсвие url и запускаемого сервлета
  **<servlet-name>** - имя сервлета
  **<url-pattern>** - описывает url-шаблон

**<session-config>** - блок, описывающий параметры сессии
  **<session-timeout>** - максимальное время жизни сессии

**<login-config>** - блок, описывающий параметры, как пользователь будет логиниться к серверу
  **<auth-method>** - метод авторизации (BASIC, FORM, DIGEST, CLIENT-CERT)

# Сервлет. Request. Response

**<welcome-file-list>** - блок, описывающий имена файлов, которые будут пытаться открыться при запросе только по имени директории (без названия файла). Сервер будет искать первый существующий файл из списка и загрузит именно его
**<welcome-file>** - имя файла

**<error-page>** - блок, описывающий соответствие ошибки и загружаемой при этом страницы
**<error-code>** - код произошедшей ошибки
**<exception-type>** - тип произошедшей ошибки
**<location>** - загружаемый файл

**<taglib>** - блок, описывающий соответствие JSP Tag library descriptor с URI-шаблоном
**<taglib-uri>** - название uri-шаблона
**<taglib-location>** - расположение шаблона

# Сервлет. Request. Response. Example 01

## web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID"
version="2.5">
  <display-name>Java_EE_01_1</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
  <servlet>
    <description></description>
    <display-name>FirstServlet</display-name>
    <servlet-name>FirstServlet</servlet-name>
    <servlet-class>_java._ee._01.firstservlet.FirstServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>FirstServlet</servlet-name>
    <url-pattern>/FirstServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

## Запуск контейнера сервлетов и размещение проекта

Действия по запуску сервлета с помощью контейнера сервлетов Tomcat, который установлен в каталоге **/Apache Software Foundation/Tomcat…**

В этом же каталоге размещаются следующие подкаталоги:

- **/bin** – содержит файлы запуска контейнера сервлетов **tomcatX.exe**, **tomcatXw.exe** и некоторые необходимые для этого библиотеки;
- **/common** – содержит библиотеки служебных классов, в частности Servlet API;
- **/conf** – содержит конфигурационные файлы, в частности конфигурационный файл контейнера сервлетов **server.xml**;
- **/logs** – помещаются log-файлы;
- **/lib** – размещены необходимые библиотеки;
- **/webapps** – в этот каталог помещаются приложение (в отдельный каталог)

# Сервлет. Request. Response

Web-приложение поставляется в виде архива **.war**, содержащего все его файлы.

На самом деле это zip-архив, расширение **.war** нужно для того, чтобы Web-контейнер узнавал архивы развертываемых на нем Web-приложений.

Содержащаяся в этом архиве структура директорий Web-приложения должна включать директорию WEB-INF, вложенную непосредственно в корневую директорию приложения.

# Сервлет. Request. Response

Директория **WEB-INF** содержит две поддиректории —

- **/classes** для .class-файлов сервлетов, классов и интерфейсов EJB-компонентов и других Java-классов, и
- **/lib** для .jar и .zip файлов, содержащих используемые библиотеки.

Файл **web.xml** также должен находится непосредственно в директории **WEB_INF**.

War-файл необходимо разместить в **папке /webapps контейнера Tomcat**

# Сервлет. Request. Response. Example 01

# Сервлет. Request. Response

Если вызывать сервлет из **index.jsp**, то тег **FORM** должен выглядеть следующим образом:

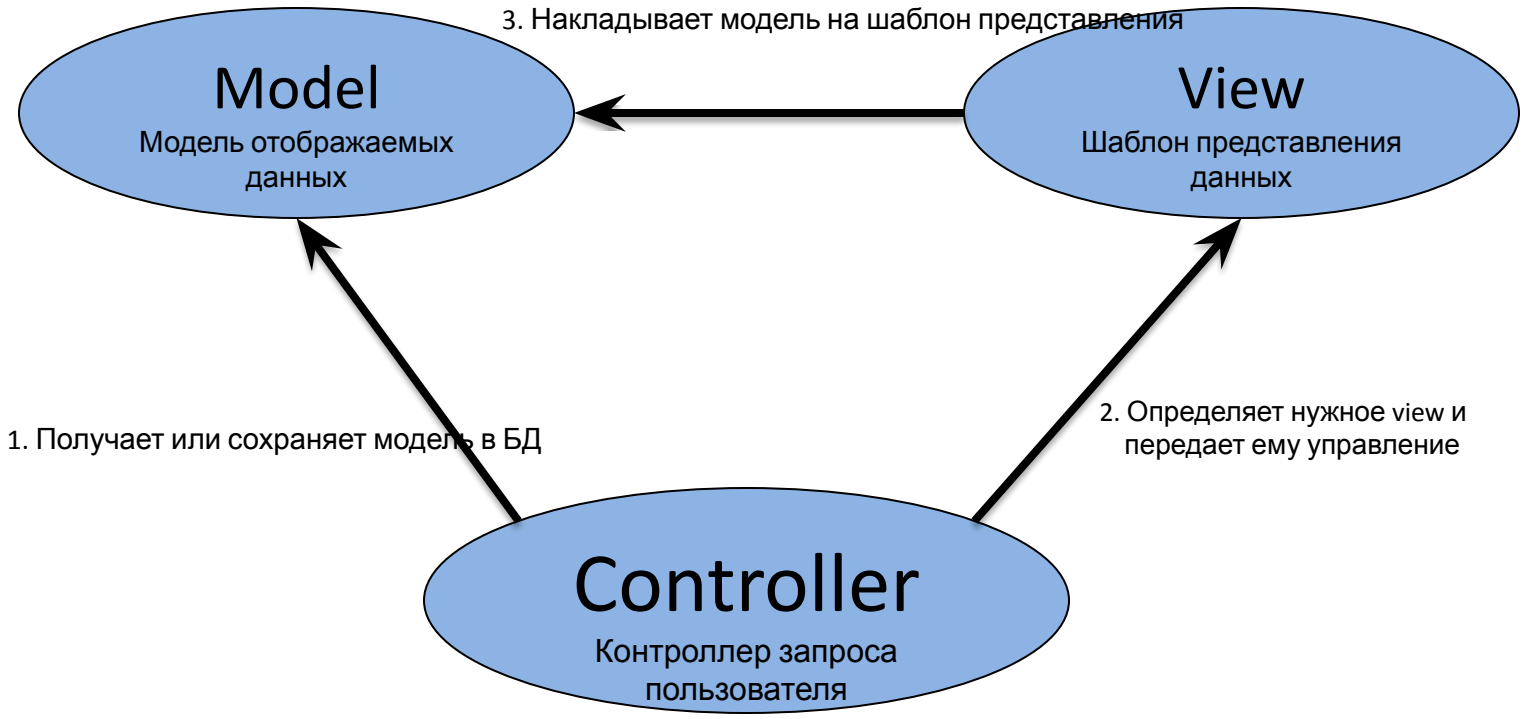
```
<FORM action="FirstServlet">
  <INPUT type="submit" value="Execute"/>
</FORM>
```

Файл **index.jsp** помещается в папку **/webapps/FirstWebProject** и в браузере набирается строка:

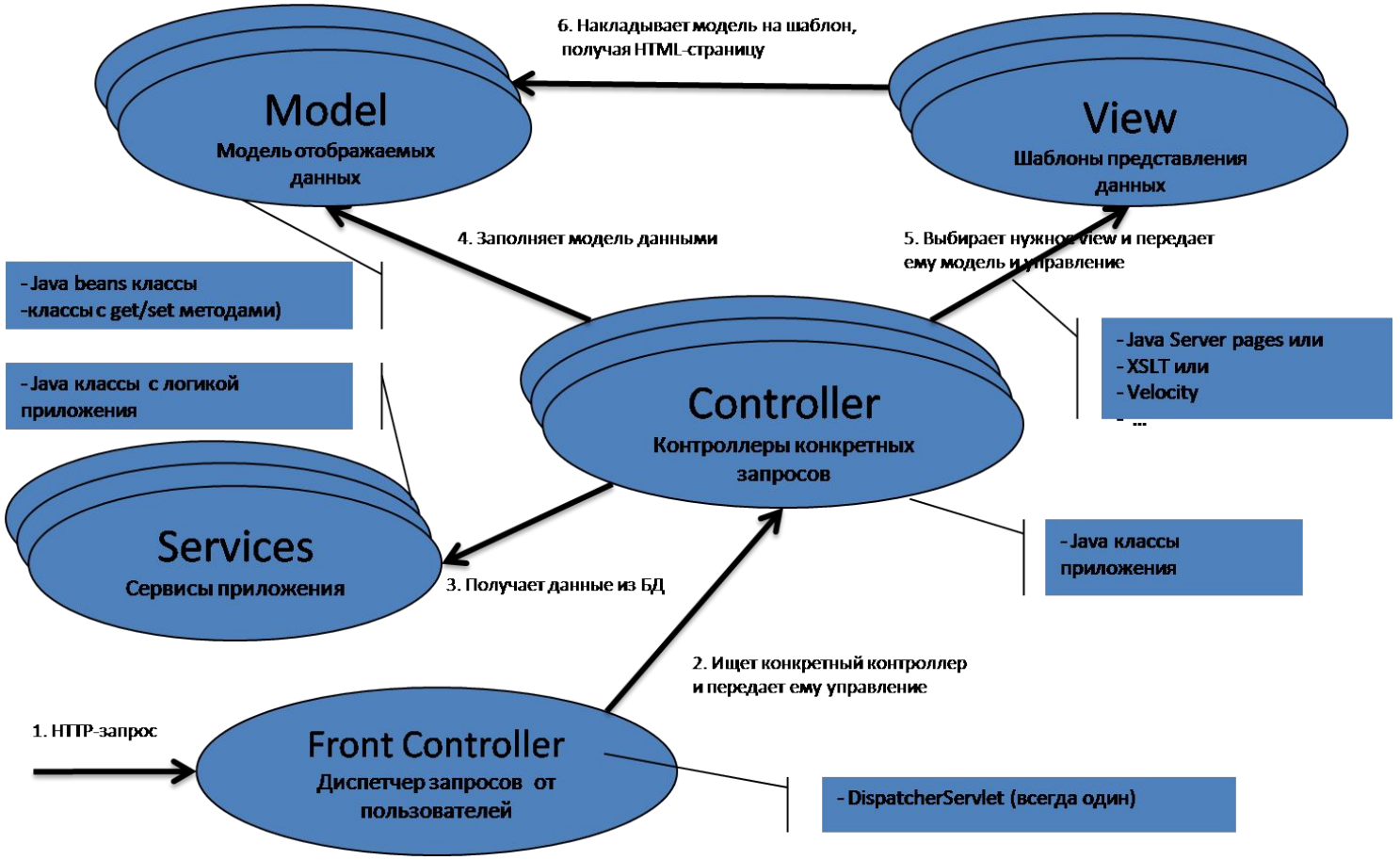
**http://localhost:8080/Java_EE_01_1/index.jsp**

# MVC

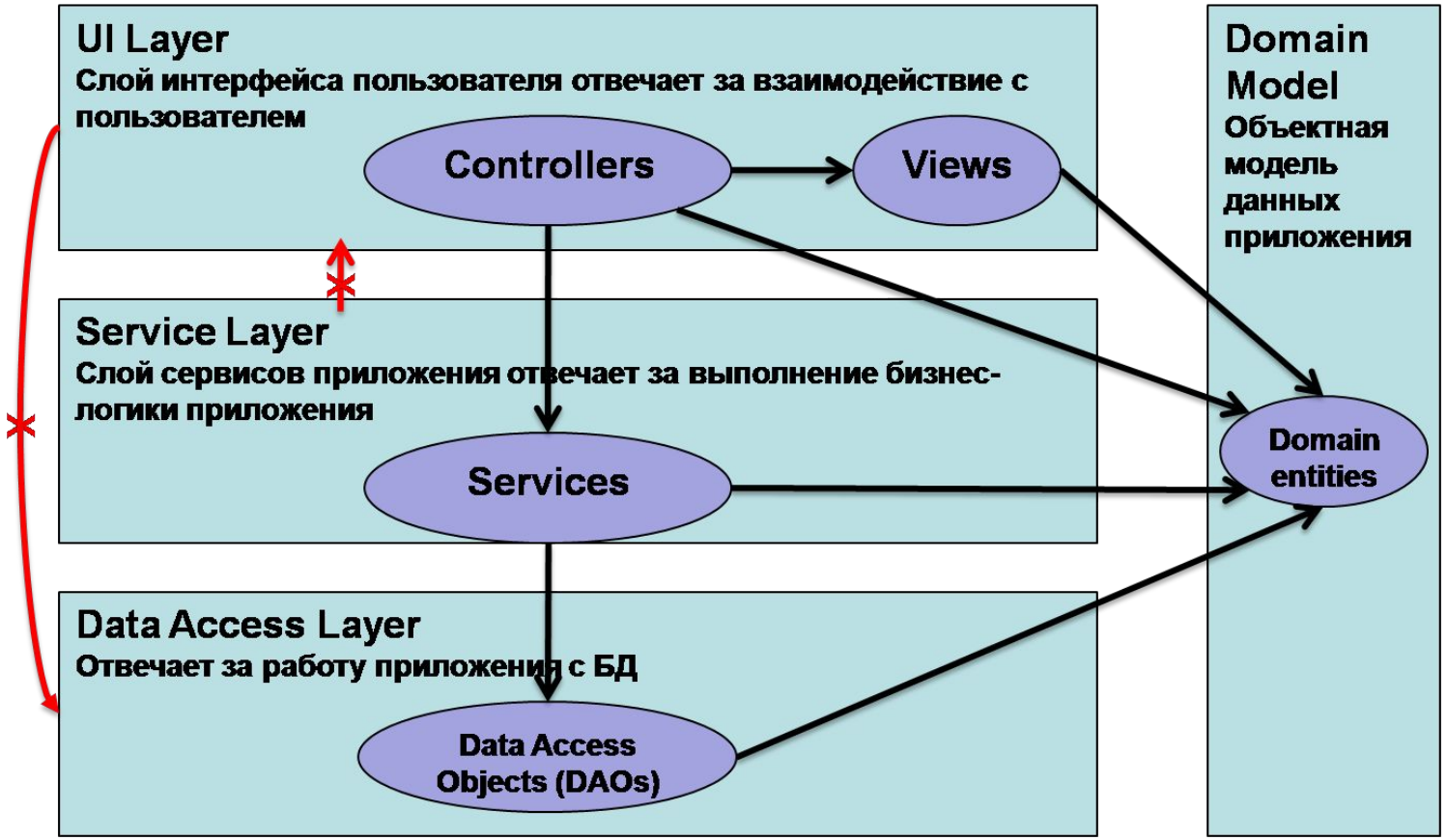
# MVC



# MVC



# MVC



# JSP (START)

# JSP (start)

**Java Server Pages (JSP)** обеспечивает разделение динамической и статической частей страницы, результатом чего является возможность изменения дизайна страницы, не затрагивая динамическое содержание.

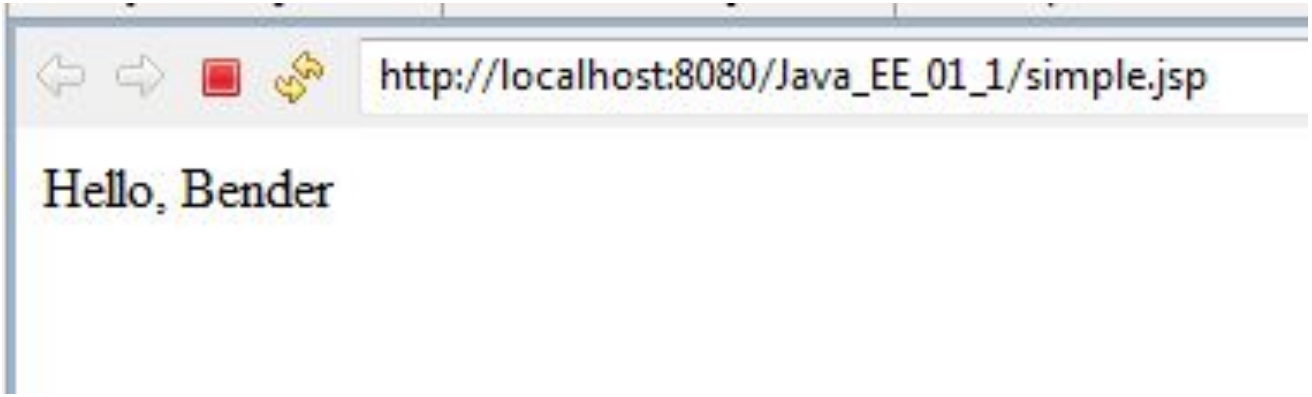
Это свойство используется при разработке и поддержке страниц, так как дизайнерам нет необходимости знать, как работать с динамическими данными.

# JSP (start)

## simple.jsp

```
<%@ page language="java" contentType="text/html; charset=utf-8"
    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>Simple</title>
</head>
<body>
<body>
    <jsp:text>Hello, Bender</jsp:text>
</body>
</html>
```

# JSP (start)



# JSP (start)

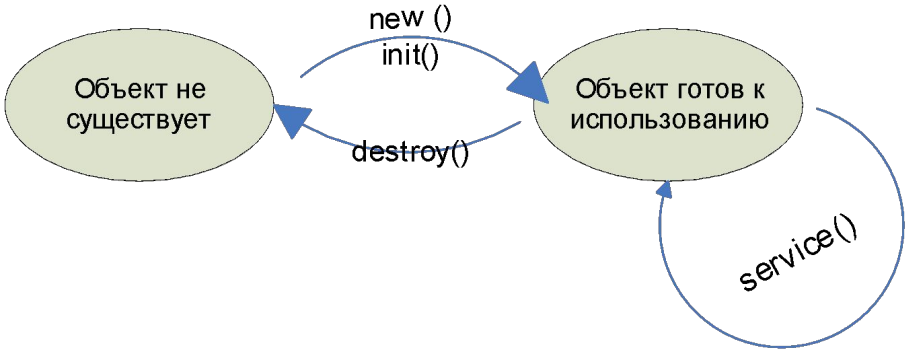
Код сервлета, полученный после компиляции **simple.jsp**

```
public final class simple_jsp {
public void _jspService(HttpServletRequest request, HttpServletResponse response)
   throws java.io.IOException, ServletException {
      try {
            _jspxFactory = JspFactory.getDefaultFactory();
            response.setContentType("text/html");
            pageContext =
          _jspxFactory.getPageContext(this,request,response,null,true,8192,true);
            _jspx_page_context = pageContext;
            application = pageContext.getServletContext();
            config = pageContext.getServletConfig();
            session = pageContext.getSession();
            out = pageContext.getOut();
            _jspx_out = out;
            out.write("<html><head>\r\n<title>Simple</title>\r\n</head>\r\n");
           out.write("<body>\r\nHello, Bender\r\n </body></html>");
      } catch (Throwable t) {
      }
   } finally {
      if (_jspxFactory != null) _jspxFactory.releasePageContext(_jspx_page_context);
   }
}
```

# ЖИЗНЕННЫЙ ЦИКЛ СЕРВЛЕТА И JSP

# Жизненный цикл сервлета и JSP

В обязанности веб контейнера входит управление жизненным циклом сервлета. Веб контейнер **создает объект сервлета** и тогда вызывает **метод init()**. По окончании выполнения этого метода, сервлет находится в состоянии готовности принимать и обрабатывать клиентские запросы. Контейнер вызывает метод **service()** в сервлете для управления каждым запросом. Перед уничтожением объекта, контейнер вызовет метод **destroy()**. После этого сервлет становится потенциальным кандидатом для сборщика мусора.



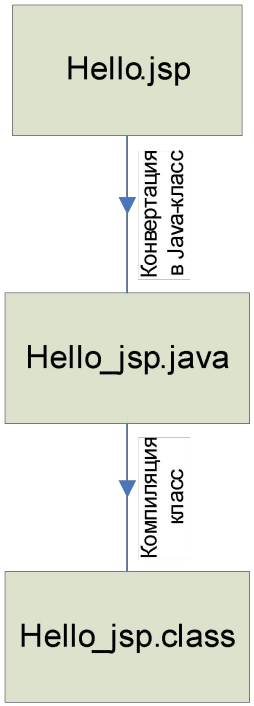
# Жизненный цикл сервлета и JSP

Процессы, выполняемые с файлом JSP при первом вызове:

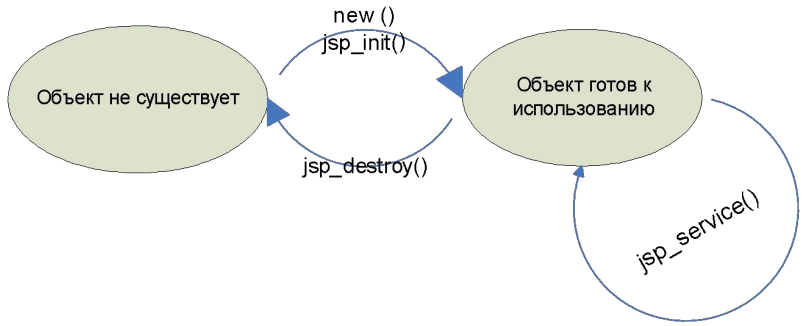
1. Браузер делает запрос к странице JSP.
2. JSP-engine анализирует содержание файла JSP.
3. JSP-engine создает временный сервлет с кодом, основанным на исходном тексте файла JSP, при этом контейнер транслирует операторы Java в метод **_jspService()**.
4. Полученный код компилируется в файл ***.class**.
5. Вызываются методы **init()** и **_jspService()**, и сервлет логически исполняется.
6. Сервлет на основе JSP установлен. Комбинация статического HTML и графики вместе с динамическими элементами, определенными в оригинале JSP, пересылаются браузеру через выходной поток объекта ответа **ServletResponse**.

# Жизненный цикл сервлета и JSP

1. Жизненный цикл класса страницы




2. Жизненный цикл объекта страницы



# ВЗАИМОДЕЙСТВИЕ СЕРВЛЕТА И JSP

# Взаимодействие сервлета и JSP

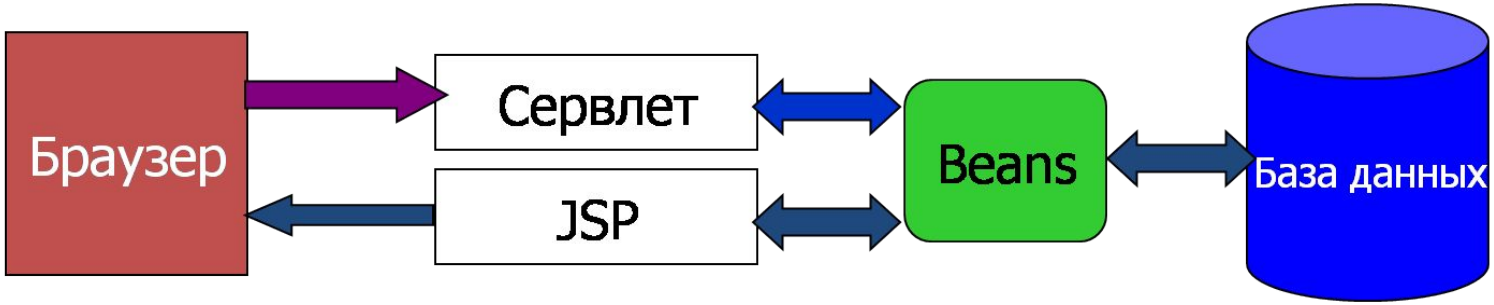
Страницы JSP и сервлеты никогда не используются в информационных системах друг без друга.

Причиной являются принципиально различные роли, которые играют данные компоненты в приложении.

Страница JSP ответственна за формирование пользовательского интерфейса и отображение информации, переданной с сервера.

Сервлет выполняет роль контроллера запросов и ответов, то есть принимает запросы от всех связанных с ним JSP-страниц, вызывает соответствующую бизнес-логику для их (запросов) обработки и в зависимости от результата выполнения решает, какую JSP поставить этому результату в соответствие.

# Взаимодействие сервлета и JSP



# Взаимодействие сервлета и JSP. Example 02

Параметры клиента передаются в сервлет как параметры запроса.

**index.jsp**

```
<%@ page language="java" contentType="text/html; charset=utf-8"
    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>MyJSP</title>
</head>
<body>
     <form action="MyServlet" method="post">
     <input type="hidden" name="command" value="forward" />
     Введите что-нибудь:<br/>
     <input type="text" name="anything" value="" /><br/>
     <input type="submit" value="Отправить" /><br/>
     </form>
</body>
</html>
```

# Взаимодействие сервлета и JSP. Example 02

```
package _java._ee._01.servlet;
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 MyServlet extends HttpServlet {
      private static final long serialVersionUID = 1L;
      protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
            response.setContentType("text/html");
            if (request.getParameter("command").equals("forward"))
            {
                  request.getRequestDispatcher("jsp/main.jsp").forward(request,
response);
            }
      }
}
```

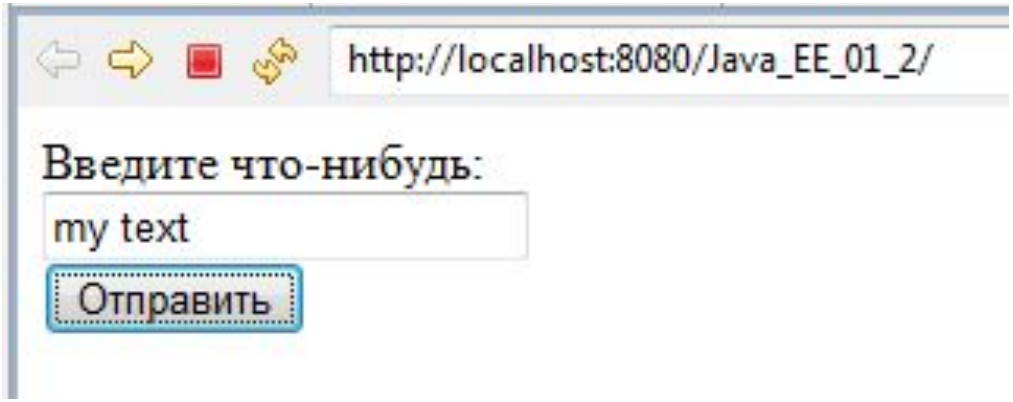
```
...
  <servlet>
    <description></description>
    <display-name>MyServlet</display-name>
    <servlet-name>MyServlet</servlet-name>
    <servlet-class>_java._ee._01.servlet.MyServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>MyServlet</servlet-name>
    <url-pattern>/MyServlet</url-pattern>
  </servlet-mapping>
...
```

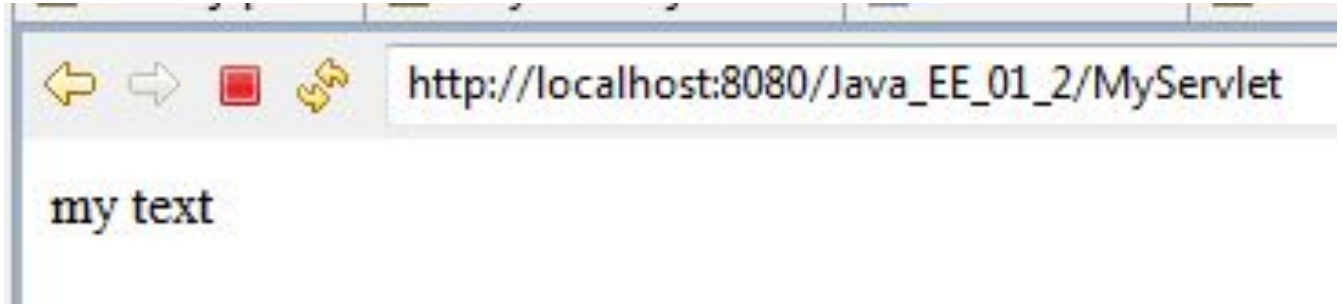
# Взаимодействие сервлета и JSP. Example 02

**jsp/main.jsp**

```
<%@ page language="java" contentType="text/html; charset=utf-8"
    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; utf-8">
<title>Insert title here</title>
</head>
<body>
    <jsp:expression>request.getParameter("anything")</jsp:expression>
</body>
</html>
```

# Взаимодействие сервлета и JSP. Example 02



# Взаимодействие сервлета и JSP

Передачу информации между JSP и сервлетом можно осуществлять, в частности, с помощью добавления атрибутов к объектам **HttpServletRequest**, **HttpSession**, **HttpServletContext.** Вызов **main.jsp** из сервлета в данном случае производится методом **forward**() интерфейса **RequestDispatcher**.

**RequestDispatcher getRequestDispatcher()** класса **ServletContext** - возвращает объект **RequestDispatcher**. Этот метод в качестве аргумента берет **URL** запрашиваемого ресурса.

# Взаимодействие сервлета и JSP

**URL** ресурса должен быть доступным на сервере, на котором запущен сервлет в момент обращения. Если ресурс недоступен, или у сервера не реализован объект RequestDispatcher для ресурса данного типа, этот метод вернет значение null. Сервлет должен быть готов к таким ситуациям.

# Взаимодействие сервлета и JSP

Обладая объектом **RequestDispatcher**, Вы можете дать возможность ассоциированному с ним ресурсу отвечать на запрос клиента, т.е. делать перенаправление.

```
public class BookStoreServlet extends HttpServlet {
   public void service (HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    RequestDispatcher dispatcher;
    dispatcher = request.getRequestDispatcher("jsp/main.jsp");
    dispatcher.forward(request, response);
       ...
    }
}
```

Перенаправление очень полезно, например, когда сервлет производит запрос, и ответ носит общий характер, так что он может быть передан другому ресурсу. Сервлет может, например, заведовать информацией кредитных карт, когда пользователь размещает заказ, и потом отправлять запрос клиента к заказу, который возвращает страницу "Спасибо за заказ".

# Взаимодействие сервлета и JSP

- **void forward (ServletRequest request, ServletResponse response)** должен быть использован тогда, когда необходимо отдать другому ресурсу возможность отвечать пользователю.

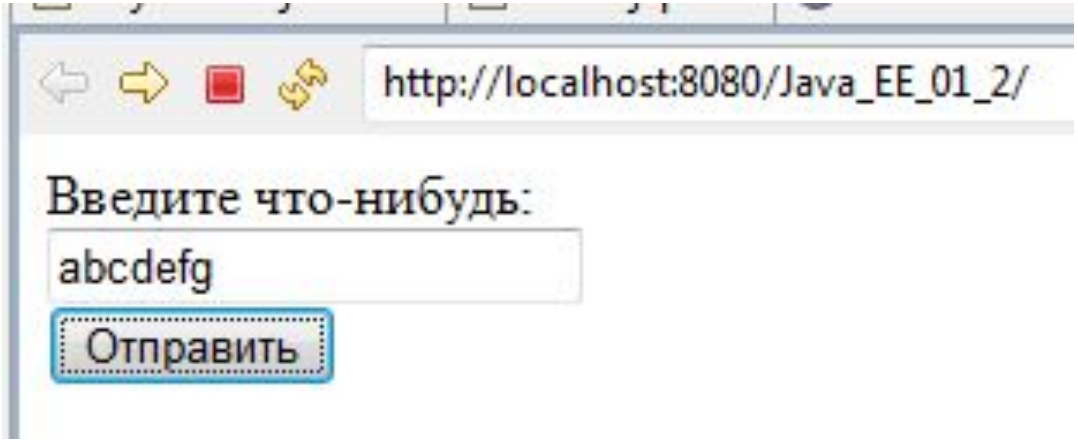
Если Вы уже начали отвечать пользователю, используя объекты **PrintWriter** или **ServletOutputStream**, Вам необходимо использовать метод **void include (ServletRequest request, ServletResponse response).**

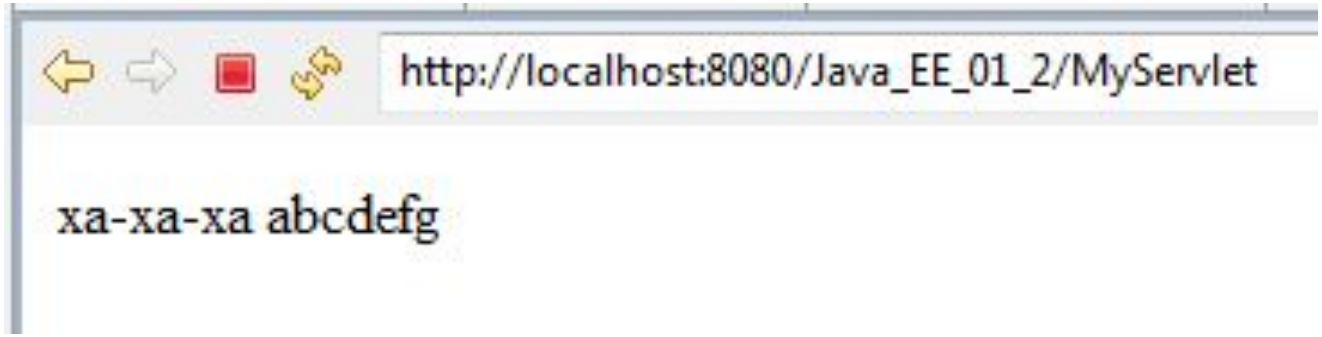
# Взаимодействие сервлета и JSP. Example 03

Метод **void include(ServletRequest request, ServletResponse response)** интерфейса **RequestDispatcher** позволяет вызываемому сервлету отвечать клиенту и использовать в качестве части ответа ресурс, ассоциированный с объектом **RequestDispatcher**.

```
public class MyServlet extends HttpServlet {
     private static final long serialVersionUID = 1L;
     protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
          response.setContentType("text/html");
          if (request.getParameter("command").equals("forward"))
          {
               response.getWriter().println("xa-xa-xa");
               RequestDispatcher dispatcher;
               dispatcher = request.getRequestDispatcher("jsp/main.jsp");
               if (dispatcher != null) {
                    dispatcher.include(request, response);
               }          }     }}
```

# Взаимодействие сервлета и JSP. Example 03



## Вызов сервлетов из сервлетов

Чтобы Ваш сервлет вызвал другой сервлет, Вы можете:

- либо дать сервлету сделать HTTP запрос к другому сервлету.
- либо сервлет может вызвать общедоступные методы другого сервлета напрямую, если оба сервлета запущены на одном и том же сервере.

Далее обсуждается второй из вышеуказанных пунктов. Чтобы вызвать общедоступный метод другого сервлета напрямую, Вам надо:

# Взаимодействие сервлета и JSP

Надо:

- Знать имя сервлета, метод которого Вы хотите вызвать.
- Получить доступ к объекту сервлета Servlet.
- Вызвать общедоступный метод.

**Servlet getServlet(String name)** класса ServletContext - возвращает доступ к объекту Servlet.

Как только Вы получили объект сервлета, Вы можете вызывать любой общедоступный метод этого сервлета.

## Потоковый вывод

Сервлеты HTTP, как правило, поддерживают обработку нескольких клиентов одновременно. Если методы в вашем сервлете, работающие на клиента, используют общие ресурсы, то Вы должны, согласовать управление, создав сервлет, который обслуживает только одного клиента в определенный момент времени.

Чтобы сервлет обслуживал только одного клиента в определенный момент времени, Вам надо реализовать интерфейс **SingleThreadModel** в добавление к наследованию класса **HttpServlet** .

# СЕССИИ

## Отслеживание сессии

Для поддержки статуса между сериями запросов от одного и того же пользователя используется механизм отслеживания сессии.

Сессии используются разными сервлетами для доступа к одному клиенту. Это удобно для приложений построенных на нескольких сервлетах.

Чтобы использовать отслеживание сессии:

- Создайте для пользователя сессию (объект HttpSession).
- Сохраняйте или читайте данные из объекта HttpSession.
- Уничтожьте сессию (необязательно).

## Получение сессии

**HttpSession getSession(bool)** объекта **HttpServletRequest** возвращает сессию пользователя. Когда метод вызывается со значением **true**, реализация при необходимости создает сессию. Значение **false** возвратит **null**, если обнаружена сессия.

Чтобы правильно организовать сессию, Вам надо вызвать метод **getSession** прежде чем будет запущен выходной поток ответа.

```
public class SessionServlet extends HttpServlet {
   public void doGet (HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
      HttpSession session = request.getSession( true);
      out = response.getWriter();
      ...
    }}
```

## Сохранение и получение данных сессии

Интерфейс **HttpSession** предоставляет методы, которые сохраняют и возвращают данные:

- Стандартные свойства сессии, такие как идентификатор сессии.
- Данные приложения, которые сохраняются в виде пары с именным ключом, когда имя это строка (**String**) и величина - объект **Java**.

# Сессии

Чтобы сохранить значения переменной в текущем сеансе, используется метод **setAttribute()** класса **HttpSession**, прочесть – **getAttribute()**, удалить – **removeAttribute()**. Список имен всех переменных, сохраненных в текущем сеансе, можно получить, используя метод **Enumeration getAttributeNames()**, работающий так же, как и соответствующий метод интерфейса **HttpServletRequest**.

Метод **String getId()** возвращает уникальный идентификатор, который получает каждый сеанс при создании. Метод **isNew()** возвращает **false** для уже существующего сеанса и **true** – для только что созданного.

## Завершение сессии

Сессия пользователя может быть завершена вручную или, в зависимости от того, где запущен сервлет, автоматически. (Например, Java Web Server автоматически завершает сессию, когда в течение определенного времени не происходит запросов, по умолчанию 30 минут.) Завершить сессию означает удаление объекта HttpSession и его величин из системы.

Чтобы вручную завершить сессию, используйте метод сессии **invalidate**.

Этот метод выдает **InvalidStateException** если вызывается для сессии, которая была завершена. Перед тем как завершить сессию, контейнер освободит все объекты, связанные с ней. Метод valueUnbound() будет вызван для всех сессионных атрибутов, которые реализуют интерфейс HttpSessionBindingListener.

**Превышение времени ожидания сессии**

Дескриптор размещения можно использовать для установки промежутка времени для сессии. Если клиент не активизировался в течение этого промежутка времени, сессия автоматически завершается.

Элемент **<session-timeout>** определяет интервал времени ожидания сессии по умолчанию (в минутах) для всех сессий, созданных в веб-приложении. Отрицательное или нулевое значение приводит к тому, что сессия никогда не завершается.

# Сессии

Следующая настройка в дескрипторе размещения приводит к установке времени ожидания сессии на 10 минут:

```
<session-config>
  <session-timeout>10</session-timeout>
</session-config>
```

Вы также можете программно установить интервал времени ожидания сессии. Следующий метод, предоставляемый интерфейсом **HttpSession**, может использоваться для установки времени ожидания для текущей сессии (в секундах):

**public void setMaxInactiveInterval(int seconds)**

Если этому методу придано отрицательное значение, время ожидания сессии никогда не истечет.

## Перезапись URL

Сессии становятся доступными при помощи обмена уникальными метками, которые называются идентификаторами сессии (session id), между клиентом, осуществляющим запрос, и сервером. Если в браузере клиента разрешены cookies, идентификатор сессии будет внесен в файл cookie, отправляемый с HTTP-запросом/ответом.

Для браузеров, не поддерживающих cookies, для того, чтобы сделать возможной обработку сессий, используется способ перезаписи URL. Если используется перезапись URL, идентификатор сессии должен быть прибавлен к URL, включая гиперссылки, которым необходим доступ к сессии, а также ответы сервера.

# Сессии

Методы **HttpServletResponse**, которые поддерживают перезапись URL:

- public String encodeURL(String url)
- public String encodeRedirectURL(String url)

Метод **encodeURL()** кодирует заданный URL, добавляя в него идентификатор сессии, или, если кодирование не требуется, возвращает исходный URL.

Метод **encodeRedirectURL()** кодирует заданный URL для использования в методе **sendRedirect()** **HttpServletResponse**. Этот метод тоже возвращает неизмененный URL, если кодирование не требуется.

Перезапись URL должна использоваться так, чтобы обеспечивалась поддержка клиентов, которые не поддерживают или не принимают cookies, чтобы не допустить потери сессионной информации.

# Сессии. Example 04

```
package _java._ee._01.servlet;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
public class SessionDemo extends HttpServlet {
     protected void doGet(HttpServletRequest request,
               HttpServletResponse response) throws ServletException, IOException
{    performTask(request, response);  }
     protected void doPost(HttpServletRequest request,
               HttpServletResponse response) throws ServletException, IOException
{    performTask(request, response);  }
     private void performTask(HttpServletRequest req, HttpServletResponse resp)
               throws ServletException {
          SessionLogic.printToBrowser(resp, req);
     }
}
```

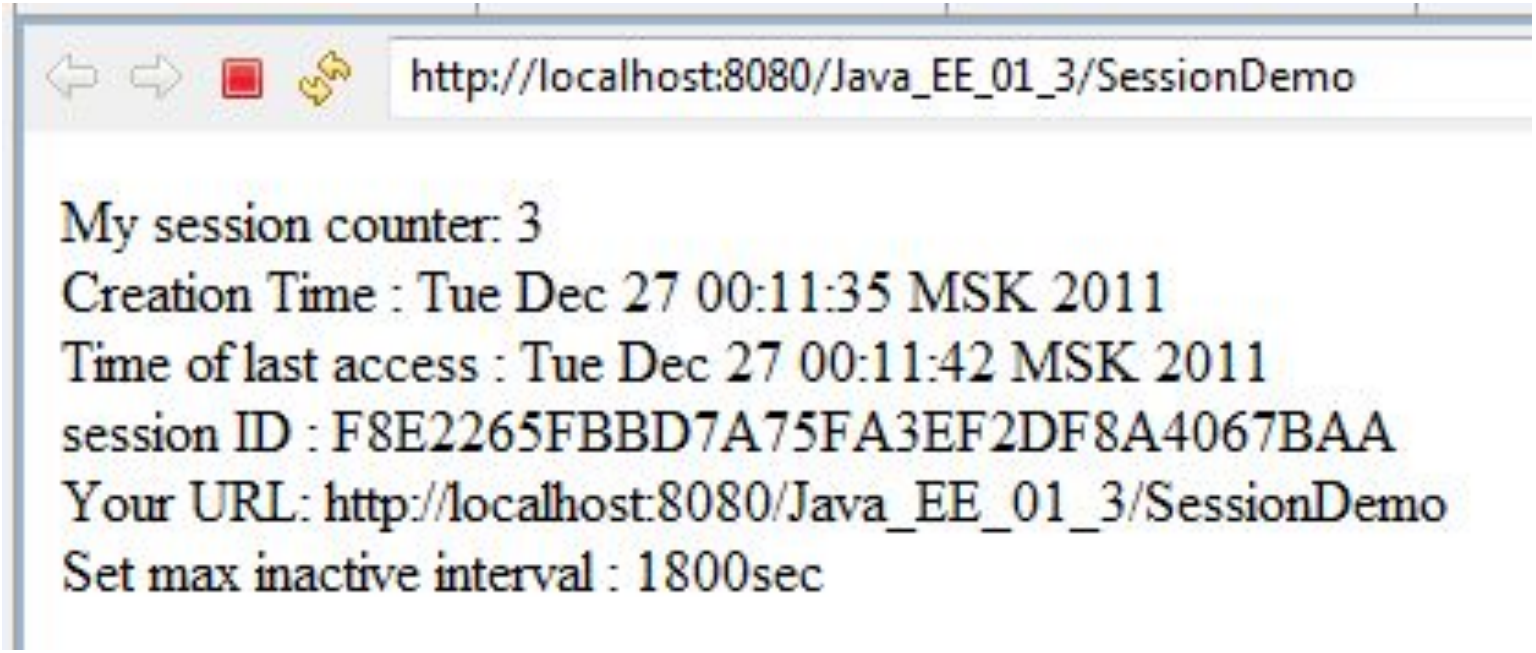
# Сессии. Example 04

```
class SessionLogic {
     public static void printToBrowser(HttpServletResponse resp,
                HttpServletRequest req) {
          try {
                HttpSession session = req.getSession(true);
                PrintWriter out = resp.getWriter();
                StringBuffer url = req.getRequestURL();
                session.setAttribute("URL", url);
                out.write("My session counter: ");
                out.write(String.valueOf(prepareSessionCounter(session)));
                out.write("<br> Creation Time : "
                          + new Date(session.getCreationTime()));
                out.write("<br> Time of last access : "
                          + new Date(session.getLastAccessedTime()));
                out.write("<br> session ID : " + session.getId());
                out.write("<br> Your URL: " + url);
                int timeLive = 60 * 30;
                session.setMaxInactiveInterval(timeLive);
                out.write("<br>Set max inactive interval : " + timeLive + "sec");
                out.flush();out.close();        }
```

# Сессии. Example 04

```
	catch (IOException e) {
			e.printStackTrace();
			throw new RuntimeException("Failed : " + e);
		}
	}
	private static int prepareSessionCounter(HttpSession session) {
		Integer counter = (Integer) session.getAttribute("counter");
		if (counter == null) {
			session.setAttribute("counter", 1);
			return 1;
		} else {
			counter++;
			session.setAttribute("counter", counter);
			return counter;
		}
	}
}
```

# Сессии. Example 04



# COOKIES

## Использование Cookie

- Закладки (cookies) используются для хранения части информации на машине клиента.
- Закладки передаются клиенту в коде ответа в HTTP.
- Клиенты автоматически возвращают закладки, добавляя код в запросы в HTTP заголовках.
- Каждый заголовок HTTP запроса и ответа именован и имеет единственное значение.
- Множественные закладки могут иметь одно и тоже имя.

# Cookies

- Сервер может обеспечить одну или более закладок для клиента. Предполагается, что программа клиента, как web браузер, должна поддерживать 20 закладок на хост, как минимум четыре килобайта каждая.

- Закладки, которые клиент сохранил для сервера, возвращаются клиентом только этому серверу. Сервер может включать множество сервлетов. Потому как закладки возвращаются серверу, сервлеты работающие на этом сервере совместно используют эти закладки.

## Использование Cookie

Чтобы отправить закладку:

1. Создайте объект Cookie
2. Установите любые атрибуты
3. Отправьте закладку

Чтобы извлечь информацию из закладки:

1. Запросите все закладки из пользовательского запроса
2. Найдите закладку или закладки с именем, которое Вас интересует, используя стандартные программные операции
3. Получите значения закладок, которые были найдены

## Создание **Cookie**

Конструктор класса **javax.servlet.http.Cookie** создает закладу с начальным именем и значением. Вы можете изменить значение закладки позже, вызвав метод **setValue**.

Если сервлет возвращает ответ пользователю, используя Writer, создавайте закладку, прежде чем обратитесь к **Writer**.

```
public void doGet (HttpServletRequest request, HttpServletResponse
 response)  throws ServletException, IOException {
   String bookId = request.getParameter("Buy");
   if (bookId != null) {
      Cookie getBook = new Cookie("Buy", bookId);
   }
   response.setContentType("text/html");
   PrintWriter out = response.getWriter();
   out.println("<html>" + "<head><title> Book Catalog
    </title></head>" + ...);
   ...
}
```

## Установка атрибутов закладки

```
public void doGet (HttpServletRequest request,
   HttpServletResponse
 response)
 throws ServletException, IOException {
   String bookId = request.getParameter( "Remove");
   if (bookId != null) {
      thisCookie.setMaxAge(0);
   }
   response.setContentType( "text/html");
   PrintWriter out = response.getWriter();
   out.println("<html> <head>" + "<title>Your Shopping
    Cart</title>" + ...);
   ...
}
```

## Отправка закладки

Закладки отправляются как заголовки ответа клиенту; они добавляются с помощью метода **addCookie** класса **HttpServletResponse**.

Закладки отправляются как заголовки ответа клиенту; они добавляются с помощью метода addCookie класса HttpServletResponse. если Вы используете Writer для отправки закладки пользователю, Вы должны использовать метод addCookie, прежде чем вызвать метод getWriter класса HttpServletResponse.

# Cookies

```
public void doGet (HttpServletRequest request,
HttpServletResponse
 response)
 throws ServletException, IOException {
   if (values != null) {
      bookId = values[0];
      Cookie getBook = new Cookie("Buy", bookId);
      getBook.setComment( "User has indicated a desire "  +
       "to buy this book from the bookstore." );
      response.addCookie(getBook);
   }
   ...
}
```

## Запрашивание закладок

Клиенты возвращают закладки как поля, добавленные в HTTP заголовок запроса.

**Cookie[] getCookies()** из класса **HttpServletRequest** – возвращает все закладки ассоциированные к данному хосту.

```
public void doGet (HttpServletRequest request, HttpServletResponse
 response)
 throws ServletException, IOException {
   String bookId = request.getParameter("Remove");
   if (bookId != null) {
      Cookie[] cookies = request.getCookies();
      thisCookie.setMaxAge(0);
   }
   response.setContentType("text/html");
   PrintWriter out = response.getWriter();
   out.println("<html> <head>" + "<title>Your Shopping Cart</title>" + ...);
   ...
}
```

## Получение значения закладки

- **String getValue()** - возвращает значение закладки.

```
public void doGet (HttpServletRequest request, HttpServletResponse
 response)
 throws ServletException, IOException {
   String bookId = request.getParameter("Remove");
   if (bookId != null) {
      Cookie[] cookies = request.getCookies();
      for(i=0; i  thisCookie.getValue().equals(bookId); i++) {
          thisCookie.setMaxAge(0);
      }
   }
   response.setContentType("text/html");
   PrintWriter out = response.getWriter();
   out.println("<html> <head>" + "<title>Your Shopping Cart</title>" + ...);
   ...
}
```

# Cookies. Example 05

```
package _java._ee._01.servlet;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class CookieDemo extends HttpServlet {
      protected void doGet(HttpServletRequest request,
      HttpServletResponse response) throws ServletException, IOException {
            perfomTask(request, response);
      }
      protected void doPost(HttpServletRequest request,
      HttpServletResponse response) throws ServletException, IOException {
            perfomTask(request, response);
      }
      protected void perfomTask(HttpServletRequest request,
      HttpServletResponse response) throws ServletException, IOException {
            response.setContentType("text/html");
            CookieLogic.seCookie(request, response);
            CookieLogic.doLogic(request, response);
      }}
```

# Cookies. Example 05

```
class CookieLogic {
private static int index = 0;
public static void doLogic(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
      PrintWriter out = response.getWriter();
      Cookie[] cookies = request.getCookies();
      if (cookies != null) {
        out.print("Number cookies :" + cookies.length + "<br/>");
        for (int i = 0; i < cookies.length; i++) {
           Cookie c = cookies[i];
           out.print("Secure :" + c.getSecure() + "<br/>");
           out.print(c.getName() + " = " + c.getValue() + "<br/>");
        }
      }
      out.write("My Cookie counter: ");
      out.write(String.valueOf(prepareCookieCounter(request, response)));
      out.print("<form action=\"\" type=\"get\"> <input type=\"submit\" />
</form>");
      out.close();
}
```

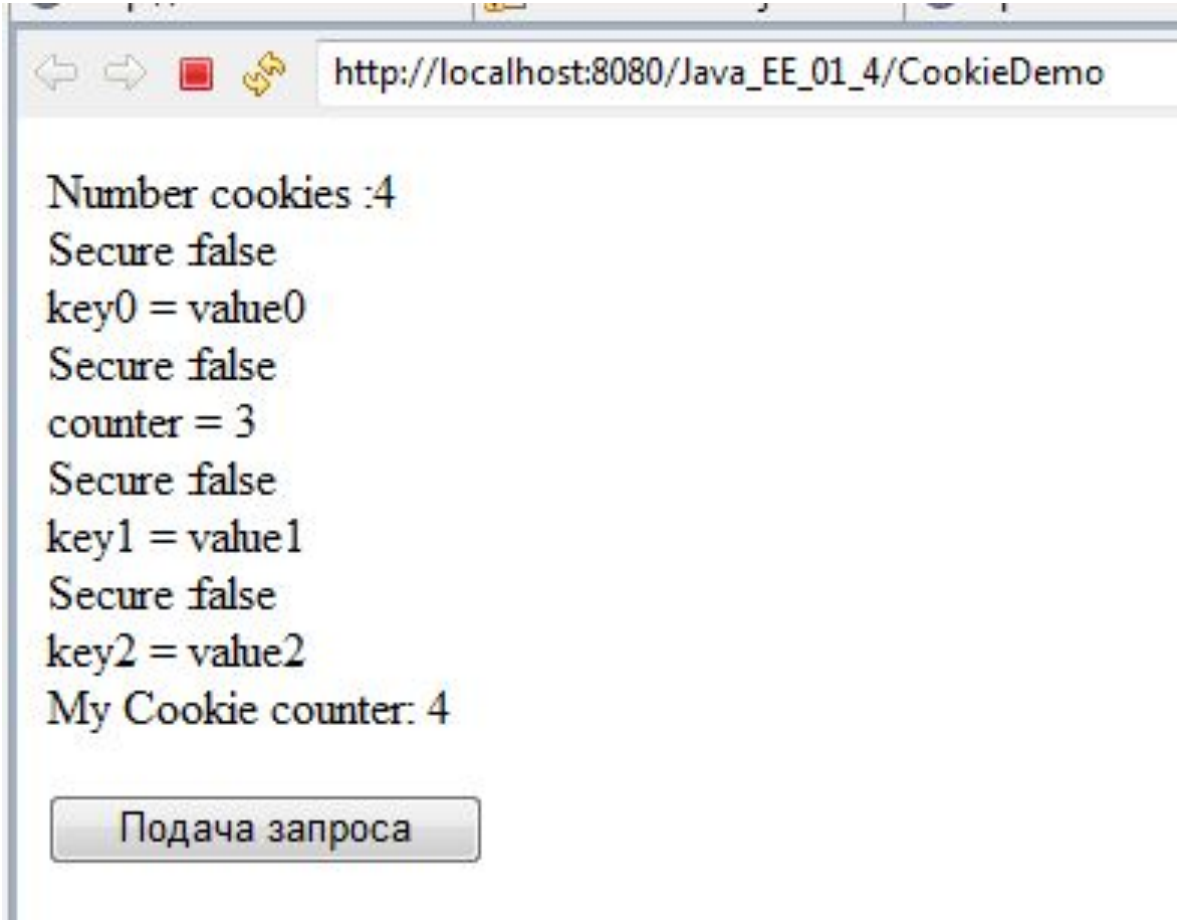
# Cookies. Example 05

```
public static void seCookie(HttpServletRequest request,
          HttpServletResponse response) {
     String key = "key" + index;
     String value = "value" + index;
     Cookie c = new Cookie(key, value);
     c.setMaxAge(3600);
     response.addCookie(c);
     index++;
}
```

# Cookies. Example 05

```
private static int prepareCookieCounter(HttpServletRequest request,
      HttpServletResponse response) {
      Cookie[] cookies = request.getCookies();
      Cookie counterCookie;
      if (cookies != null) {
            for (int i = 0; i < cookies.length; i++) {
            if ("counter".equals(cookies[i].getName())) {
                  String counterStr = cookies[i].getValue();
                  int counterValue;
                  try {
                  counterValue = Integer.parseInt(counterStr);
                  } catch (NumberFormatException e) {
                        counterValue = 0;
                  }
                  counterValue++;
                  counterCookie = new Cookie("counter", String.valueOf(counterValue));
                  counterCookie.setMaxAge(3600);
                  response.addCookie(counterCookie);
                  return counterValue;
            }}
      }
      counterCookie = new Cookie("counter", "1");
      counterCookie.setMaxAge(3600);
      response.addCookie(counterCookie);
      return 1;
}}
```

# Cookies. Example 05



# СОВМЕСТНОЕ ИСПОЛЬЗОВАНИЕ РЕСУРСОВ

# Совместное использование ресурсов

Сервлеты выполняемые на одном сервере иногда совместно используют ресурсы.

Это справедливо для сервлетов, которые являются компонентами одного приложения.

Сервлеты исполняемые на одном сервере могут совместно использовать ресурсы с помощью методов интерфейса **ServletContext** для манипулирования атрибутами: **setAttribute**, **getAttribute**, и **removeAttribute**.

Все сервлеты в контексте совместно используют атрибуты, находящиеся в интерфейсе **ServletContext**.

Чтобы избежать столкновений имен атрибутов, имена их используют те же правила что и имена пакетов.

# Совместное использование ресурсов

Пример названия атрибута

**examples.bookstore.database.BookDBFrontEnd.**

Имена, начинающиеся с префиксов **java.***, **javax.***, и **sun.*** зарезервированы.

# Совместное использование ресурсов

**Интерфейс ServletContext**

Интерфейс **ServletContext** используется для взаимодействия с контейнером сервлетов.

Сервлеты исполняемые на одном сервере могут совместно использовать ресурсы с помощью методов интерфейса **ServletContext** для манипулирования атрибутами:

- **void setAttribute(String name, Object object)** – добавляет атрибут и его значение в контекст; обычно это производится во время инициализации.

# Совместное использование ресурсов

Когда у Вас несколько сервлетов используют атрибут, каждый должен проинициализировать этот атрибут. А раз так, каждый сервлет должен проверить значение атрибута, и устанавливать его только в том случае если предыдущий сервлет не сделал этого.

# Совместное использование ресурсов

*ServletContext*

- **Object getAttribute(String name)** – возвращает совместный ресурс.
- **Enumeration getAttributeNames()** – получает список имен атрибутов;
- **void removeAttribute(String name)** – удаляет совместный ресурс.
- **ServletContext getContext(String uripath)** – позволяет получить доступ к контексту других ресурсов данного контейнера сервлетов;
- **String getServletContextName()** – возвращает имя сервлета, которому принадлежит данный объект интерфейса ServletContext.
- **String getCharacterEncoding()** – определение символьной кодировки запроса.

**Интерфейс ServletConfig**

Представляет собой конфигурацию сервлета, используется в основном на этапе инициализации. Все параметры для инициализации устанавливаются в web.xml

Некоторые методы:

- **String getServletName()** – определение имени сервлета;
- **Enumeration getInitParameterNames()** – определение имен параметров инициализации сервлета из дескрипторного файда web.xml;
- **String getInitParameter(String name)** – определение значения конкретного параметра по его имени.

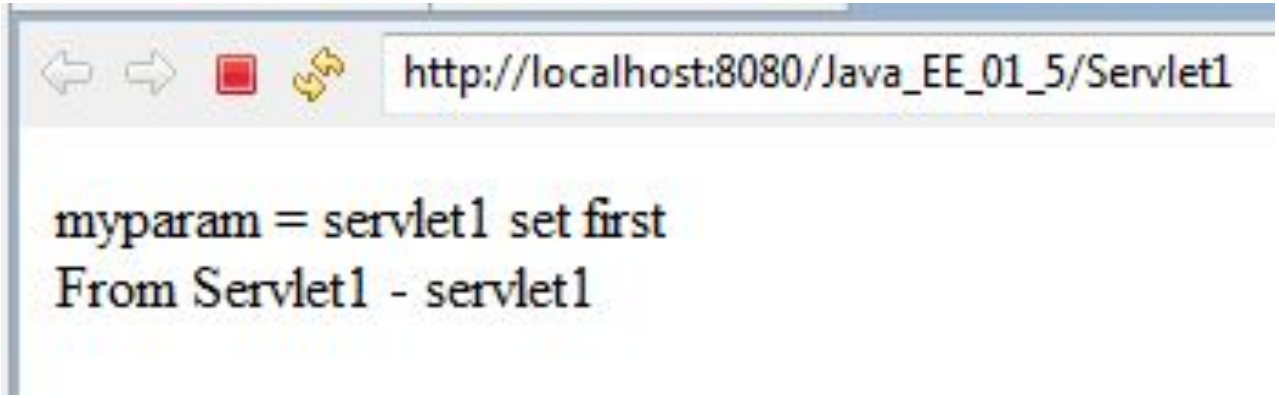
# Совместное использование ресурсов. Example 06

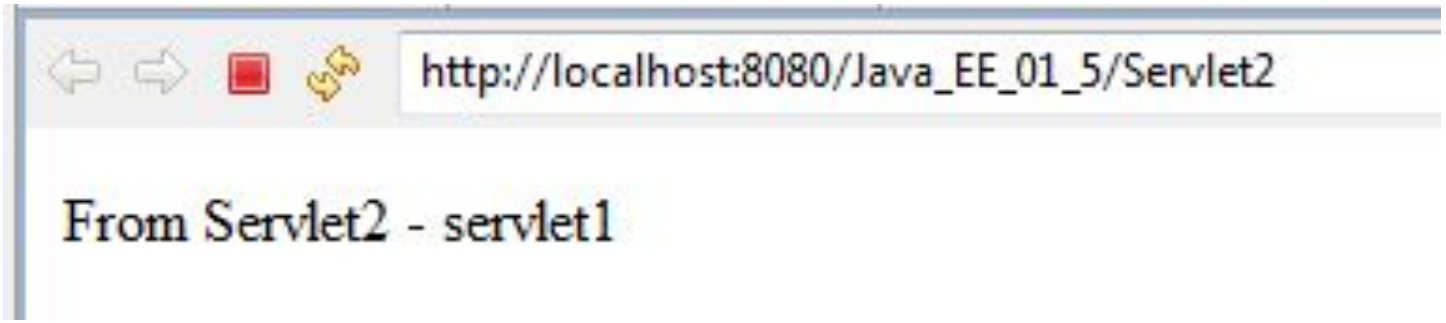
```
package _java._ee._01.servlet;
+import java.io.IOException;..
public class Servlet1 extends HttpServlet {
  private ServletConfig config;

  public void init (ServletConfig config) throws ServletException
  {
        this.config = config;
  }

  protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
     response.setContentType("text/html");
     String param = (String) config.getServletContext().getAttribute("myparam");
     if (param == null){
          config.getServletContext().setAttribute("myparam", "servlet1");
          response.getWriter().println("myparam = servlet1 set first<br/>");
     }
     response.getWriter().println("From Servlet1 - " +
config.getServletContext().getAttribute("myparam"));
  }
}
```

# Совместное использование ресурсов. Example 06

```
package _java._ee._01.servlet;
+import java.io.IOException;..
public class Servlet2 extends HttpServlet {
    private ServletConfig config;

    public void init (ServletConfig config) throws ServletException
    {
         this.config = config;
    }

  protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
      response.setContentType("text/html");
      String param = (String) config.getServletContext().getAttribute("myparam");
      if (param == null){
        config.getServletContext().setAttribute("myparam", "servlet2");
        response.getWriter().println("myparam = servlet2 set first<br/>");
      }
      response.getWriter().println("From Servlet2 - " +
config.getServletContext().getAttribute("myparam"));
  }
}
```

# Совместное использование ресурсов. Example 06



# СОБЫТИЯ

# События

Существует несколько интерфейсов, которые позволяют следить за событиями, связанными с сеансом, контекстом и запросом сервлета, генерируемыми во время жизненного цикла Web-приложения:

- **javax.servlet.ServletContextListener** – обрабатывает события создания/удаления контекста сервлета;
- **javax.servlet.http.HttpSessionListener** – обрабатывает события создания/удаления HTTP-сессии;
- **javax.servlet.ServletContextAttributeListener** – обрабатывает события создания/удаления/модификации атрибутов контекста сервлета;
- **javax.servlet.http.HttpSessionAttributeListener** – обрабатывает события создания/удаления/модификации атрибутов HTTP-сессии;
- **javax.servlet.http.HttpSessionBindingListener** – обрабатывает события привязывания/разъединения объекта с атрибутом HTTP-сессии;
- **javax.servlet.http.HttpSessionActivationListener** – обрабатывает события связанные с активацией/дезактивацией HTTP-сессии;
- **javax.servlet.ServletRequestListener** – обрабатывает события создания/удаления запроса;
- **javax.servlet.ServletRequestAttributeListener** – обрабатывает события создания/удаления/модификации атрибутов запроса сервлета.

# События

| Интерфейсы и их методы |
| --- |
| `ServletContextListener`<br>contextInitialized(ServletContextEvent e)<br>contextDestroyed(ServletContextEvent e) |
| `HttpSessionListener`<br>sessionCreated(HttpSessionEvent e)<br>sessionDestroyed(HttpSessionEvent e) |
| `ServletContextAttributeListener`<br>attributeAdded(ServletContextAttributeEvent e)<br>attributeRemoved(ServletContextAttributeEvent e)<br>attributeReplaced(ServletContextAttributeEvent e) |
| `HttpSessionAttributeListener`<br>attributeAdded(HttpSessionBindingEvent e)<br>attributeRemoved(HttpSessionBindingEvent e)<br>attributeReplaced(HttpSessionBindingEvent e) |

# События

| Интерфейсы и их методы |
| --- |
| `HttpSessionBindingListener`<br>valueBound(HttpSessionBindingEvent e)<br>valueUnbound(HttpSessionBindingEvent e) |
| `HttpSessionActivationListener`<br>sessionWillPassivate(HttpSessionEvent e)<br>sessionDidActivate(HttpSessionEvent e) |
| `ServletRequestListener`<br>requestDestroyed(ServletRequestEvent e)<br>requestInitialized(ServletRequestEvent e) |
| `ServletRequestAttributeListener`<br>attributeAdded(ServletRequestAttributeEvent e)<br>attributeRemoved(ServletRequestAttributeEvent e)<br>attributeReplaced(ServletRequestAttributeEvent e) |

# События. Example 07

```
package _java._ee._01.servlet;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
public class ListenerExample extends HttpServlet {
protected void doGet(HttpServletRequest request,
      HttpServletResponse response) throws ServletException, IOException {
  try {
      response.setContentType("text/html");
      HttpSession session = request.getSession(true);
      PrintWriter out = response.getWriter();
      StringBuffer url = request.getRequestURL();
      session.setAttribute("URL", url);
      out.write(String.valueOf(prepareSessionCounter(session)));
      out.write("<br> Creation Time : “
                + new Date(session.getCreationTime()));
      out.write("<br> Time of last access : “
                + new Date(session.getLastAccessedTime()));
```

# События. Example 07

```
      out.write("<br> session ID : " + session.getId());
      out.write("<br> Your URL: " + url);
      int timeLive = 60 * 30;
      session.setMaxInactiveInterval(timeLive);
      out.write("<br>Set max inactive interval : " + timeLive + "sec");
      out.flush();
      out.close();
  } catch (IOException e) {
      e.printStackTrace();
      throw new RuntimeException("Failed : " + e);
  }
}
private static int prepareSessionCounter(HttpSession session) {
      Integer counter = (Integer) session.getAttribute("counter");
      if (counter == null) {
            session.setAttribute("counter", 1);
            return 1;
      } else {
            counter++;
            session.setAttribute("counter", counter);
            return counter;
      }
}
}
```

# События. Example 07

```
package _java._ee._01._listener;
import javax.servlet.http.HttpSessionAttributeListener;
import javax.servlet.http.HttpSessionBindingEvent;

public class MyAttributeListener implements HttpSessionAttributeListener {
  private String counterAttr = "counter";

  public void attributeAdded(HttpSessionBindingEvent ev) {
      String currentAttributeName = ev.getName();
      String urlAttr = "URL";
      if (currentAttributeName.equals(counterAttr)) {
            Integer currentValueInt = (Integer) ev.getValue();
            System.out.println("Counter added in Session=“
                        + currentValueInt);
      } else if (currentAttributeName.equals(urlAttr)) {
            StringBuffer currentValueStr = (StringBuffer) ev.getValue();
            System.out.println("URL added in Session =“
                        + currentValueStr);
      } else
            System.out.println("new attribute added");
  }
```

# События. Example 07

```
public void attributeRemoved(HttpSessionBindingEvent ev) {
  }


public void attributeReplaced(HttpSessionBindingEvent ev) {
      String currentAttributeName = ev.getName();
      if (currentAttributeName.equals(counterAttr)) {
            Integer currentValueInt = (Integer) ev.getValue();
            System.out.println("counter changed in Session = “
                        + currentValueInt);
      }
  }
}
```

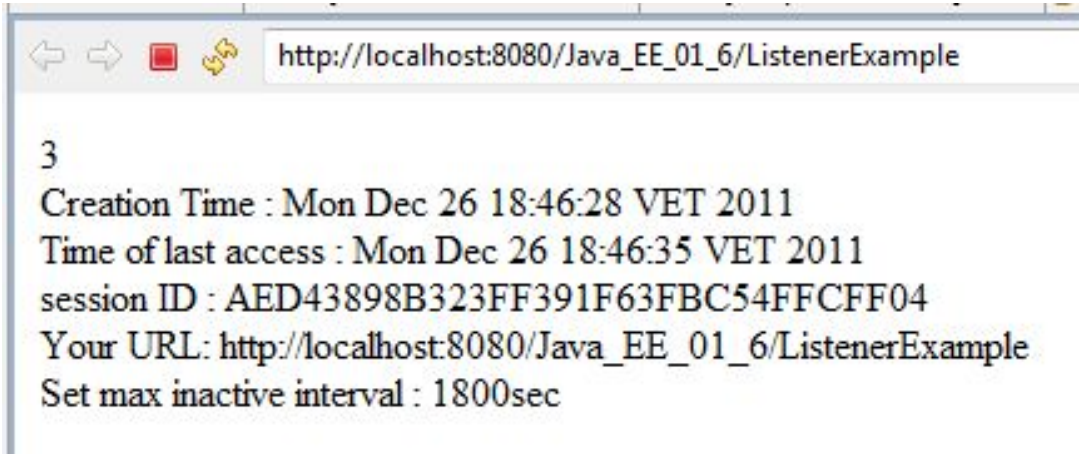
# События. Example 07

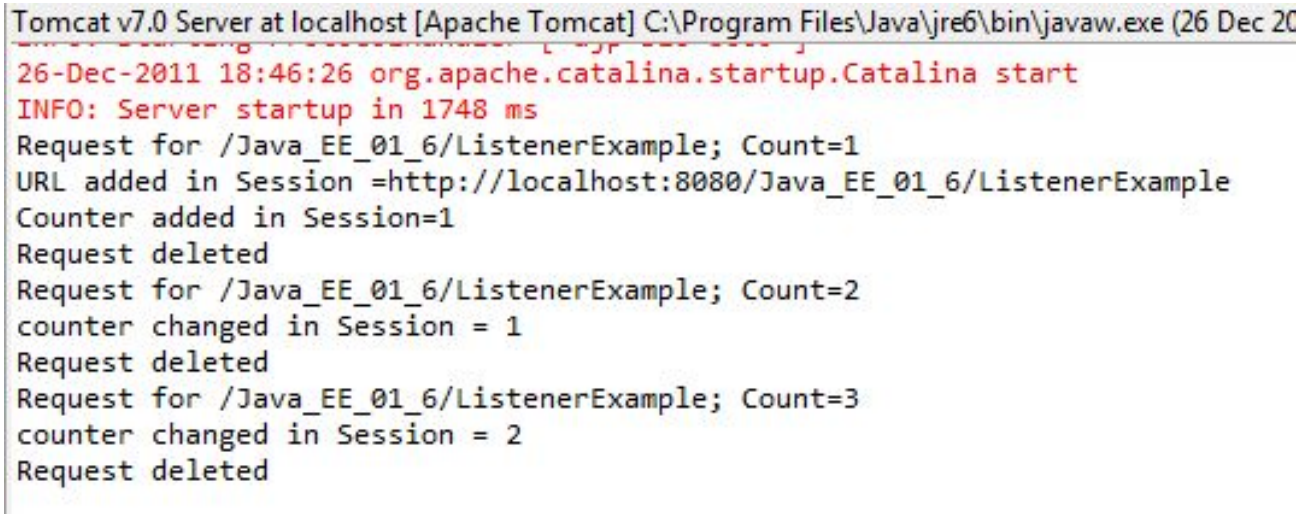
```
package _java._ee._01._listener;

import javax.servlet.ServletRequest;
import javax.servlet.ServletRequestEvent;
import javax.servlet.ServletRequestListener;
import javax.servlet.http.HttpServletRequest;

public class MyRequestListener implements ServletRequestListener {
  private static int reqCount;

  public void requestInitialized(ServletRequestEvent e) {
      ServletRequest req = e.getServletRequest();
      String name = "";
      name = ((HttpServletRequest) req).getRequestURI();
      System.out.println("Request for " + name + "; Count=" + ++reqCount);
  }

  public void requestDestroyed(ServletRequestEvent e) {
      System.out.println("Request deleted");
  }
}
```

# События. Example 07

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
... version="2.5">
  <display-name>Java_EE_01_6</display-name>
  <welcome-file-list>
...
  </welcome-file-list>
  <listener>
    <listener-class>_java._ee._01._listener.MyAttributeListener</listener-class>
  </listener>
  <listener>
    <listener-class>_java._ee._01._listener.MyRequestListener</listener-class>
  </listener>
  <servlet>
    <description></description>
    <display-name>ListenerExample</display-name>
    <servlet-name>ListenerExample</servlet-name>
    <servlet-class>_java._ee._01.servlet.ListenerExample</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ListenerExample</servlet-name>
    <url-pattern>/ListenerExample</url-pattern>
  </servlet-mapping>
</web-app>
```

# События. Example 07



# ФИЛЬТРЫ

# Фильтры

Реализация интерфейса **Filter** позволяет создать объект, который может трансформировать заголовок и содержимое запроса клиента или ответа сервера.

Фильтры не создают запрос или ответ, а только модифицируют его. Фильтр выполняет предварительную обработку запроса, прежде чем тот попадает в сервлет, с последующей (если необходимо) обработкой ответа, исходящего из сервлета.

Фильтр может взаимодействовать с разными типами ресурсов, в частности и с сервлетами, и с JSP-страницами.

# Фильтры

Основные действия, которые может выполнить фильтр:

- перехват инициализации сервлета и определение содержания запроса, прежде чем сервлет будет инициализирован;
- блокировка дальнейшего прохождения пары request-response;
- изменение заголовка и данных запроса и ответа;
- взаимодействие с внешними ресурсами;
- построение цепочек фильтров;
- фильтрация более одного сервлета.

# Фильтры

При программировании фильтров следует обратить внимание на интерфейсы **Filter**, **FilterChain** и **FilterConfig** из пакета **javax.servlet**. Сам фильтр определяется реализацией интерфейса **Filter**. Основным методом этого интерфейса является метод

**void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)**,

которому передаются объекты запроса, ответа и цепочки фильтров. Он вызывается каждый раз, когда запрос/ответ проходит через список фильтров **FilterChain**. В данный метод помещается реализация задач, обозначенных выше.

# Фильтры

Кроме того, необходимо реализовать метод **void init(FilterConfig config)**, который принимает параметры инициализации и настраивает конфигурационный объект фильтра **FilterConfig**. Метод **destroy()** вызывается при завершении работы фильтра, в тело которого помещаются команды освобождения используемых ресурсов.

Жизненный цикл фильтра начинается с однократного вызова метода **init()**, затем контейнер вызывает метод **doFilter()** столько раз, сколько запросов будет сделано непосредственно к данному фильтру. При отключении фильтра вызывается метод **destroy()**.

# Фильтры

С помощью метода **doFilter()** каждый фильтр получает текущий запрос и ответ, а также список фильтров **FilterChain**, предназначенных для обработки. Если в **FilterChain** не осталось необработанных фильтров, то продолжается передача запроса/ответа. Затем фильтр вызывает **chain.doFilter()** для передачи управления следующему фильтру.

# Фильтры. Example 08

```
package _java._ee._01.servlet;

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

public class FilterDemo extends HttpServlet {
  protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
       response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();

        out.println( "Encoding response: " + response.getCharacterEncoding() );
        out.println("<br/>");
        out.println("Encoding request: " + request.getCharacterEncoding());

        out.println(request.getParameter("mytext"));
        out.flush();
        out.close();
  }
}
```

# Фильтры. Example 08

```
package _java._ee._01._filter;
import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
public class CharsetFilter implements Filter {
  private String encoding;
  public void init(FilterConfig config) throws ServletException {
      encoding = config.getInitParameter("requestEncoding");
      if (encoding == null)
            encoding = "utf-8";
  }
  public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain next) throws IOException, ServletException {
      request.setCharacterEncoding(encoding);
      response.setCharacterEncoding(encoding);
      System.out.println("I am here");
      next.doFilter(request, response);
  }
  public void destroy() {
  }
}
```

# Фильтры. Example 08

```
package _java._ee._01._filter;
import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
public class AnotherFilter implements Filter{
    private String message;
    public void init(FilterConfig config) throws ServletException
    {
        message = "Another Filter";
    }
    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain next)
    throws IOException, ServletException
    {
            response.getWriter().println(message);
      System.out.println(message);
      next.doFilter(request, response);
    }
    public void destroy(){}
}
```

# Фильтры. Example 08

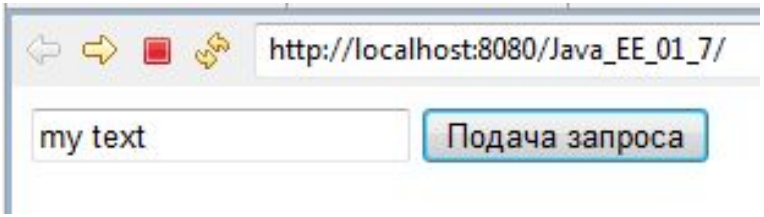
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ....>
      <display-name>Java_EE_01_7</display-name>
...
      </welcome-file-list>
      <servlet>
            <servlet-name>FilterDemo</servlet-name>
            <servlet-class>_java._ee._01.servlet.FilterDemo</servlet-class>
      </servlet>
      <servlet-mapping>
            <servlet-name>FilterDemo</servlet-name>
            <url-pattern>/FilterDemo</url-pattern>
      </servlet-mapping>
      <filter>
            <filter-name>setCharFilter</filter-name>
            <filter-class>_java._ee._01._filter.CharsetFilter</filter-class>
      </filter>
      <filter-mapping>
            <filter-name>setCharFilter</filter-name>
            <url-pattern>/FilterDemo</url-pattern>
      </filter-mapping>
      <filter>
            <filter-name>setAnotherFilter</filter-name>
            <filter-class>_java._ee._01._filter.AnotherFilter</filter-class>
      </filter>
      <filter-mapping>
            <filter-name>setAnotherFilter</filter-name>
            <url-pattern>/FilterDemo</url-pattern>
      </filter-mapping>
</web-app>
```

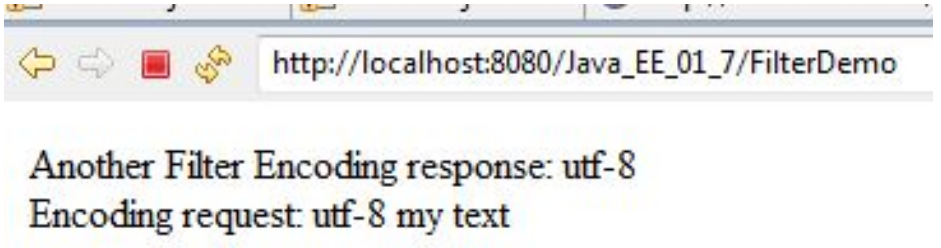
# Фильтры. Example 08

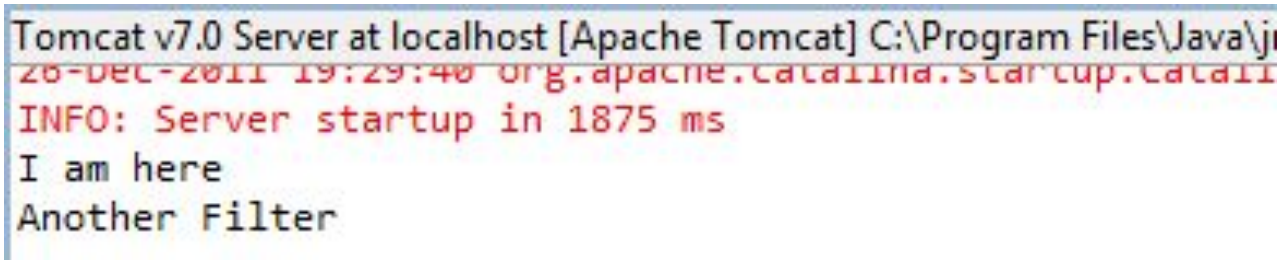
## index.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    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>Insert title here</title>
</head>
<body>
     <form action="FilterDemo" method="post">
          <input type="text" name="mytext" value=""/>
          <input type="submit" name="send"/>
     </form>
</body>
</html>
```

# Фильтры. Example 08



# СПАСИБО ЗА ВНИМАНИЕ!

# ВОПРОСЫ?

**Java.EE.01**

Servlets

**Author:** **Ihar Blinou, PhD**
**Oracle Certified Java Instructor**
**Ihar_blinou@epam.com**