Enterprise Application Development

An Introduction to Java Enterprise Edition

Shahid Beheshti University

Sadegh Aliakbary
Outline

- Enterprise Application Development
- Web Programming
- Java Enterprise Edition
  - Architectures
  - Patterns
  - Standards
  - Technologies
Static Web Pages

www.abc.com/index.html

HTML Files

index.html
Dynamic Web Pages

Browser

Web Server

Form

Submit Form
Web Application

- **Definition:** A web application is an application delivered to users from a web server over a network such as the Internet
- Only needs a web browser to use the application (Thin Client)
  - Software application that is coded in a browser-supported language
- Common web applications, e.g., webmail, Google Docs, Portals, …
Web Applications Layers

- Logical Partitioning → Layering
- Common layering in web applications
  - Presentation Layer
  - Business logic Layer
  - Data (management/source) Layer
- These layers are purely abstractions
- These layers may not correspond to physical distribution (tiers)
Presentation Layer

- Handling the interactions between the user and the software
  - GUI
  - HTML based browser

- The user interface of the application
  - Can be made up client side & server side codes

- It communicates with other layers by outputting results to the browser/client software and all other layers

- What is this layer in Facebook?
Business Logic Layer

- The work that the application needs to do for the domain
- It controls an application’s functionality by performing detailed processing
  - Validation of the data from the presentation
  - Processing/Computing/Calculations
  - Dispatching data source logic
  - …

- What does this layer do in Facebook?
Data Layer

- Communicating with other systems that carry out tasks (typically data retrieval) on behalf of the application
- Database server
- Files
- Transaction monitor

- What is this layer in Facebook?
Multilayer Architecture

Presentation Layer

Business logic Layer

Data Layer
Data Layer Trends

- New Patterns and technologies in data layer:
  - Object Databases
  - ORM
  - NoSQL
  - CQRS
    - Command Query Responsibility Segregation
  - Data-warehousing
Client-Server Architecture

- Client-Server: The traditional architecture for distributed computing (including web)
- Client: Active (master), Sends requests, Awaits response
- Server: passive (slave), waits for requests, serves requests and sends a response
- Thin client (Pros and Cons?)
  - function is mainly presentational
    - e.g. standard browser functionality
  - All significant processing done by server
- Fat client (Pros and Cons?)
  - Significant processing on client
    - e.g. Java applet, Flash
  - less server load
Multitier Architecture

- Physical separation of these layers is another story
  - **Tiers**: the physical separation of layers
- Three-tier Architecture:
- N-tier Architecture:
Three-Tier (Web Server)

- Browser handles presentation logic
- Browser talks to Web server via HTTP protocol
- Business logic and data model are handled by “dynamic contents generation” technologies (CGI, Servlet/JSP, ASP)
N-Tier Architecture

- In N-tier deployments, presentation layer, business logic layer, and data layer are separated into respective physical tiers
  - 3 tier: client + server + data base
- Presentation layer is implemented by parts in both client & server sides
  - E.g., dynamic web page using AJAX + PHP
  - 4 tier: Browser + Web server + Application Server + Database server
- Complicated Business logic layer itself can be distributed multitier application → N-tier
Typical Web Application N-tier Architecture

HTML Client (browser)

Web Server

Application Server

Database Server

Sadegh Aliakbary

JavaEE
N-Tier Architecture Characteristics

- Migration costs are low
  - Business logic application migration
  - Database switching
  - Web server switch
  - OS upgrade
  - Each tier can be upgraded independently
- Communication performance suffers
- Maintenance costs are high
Application servers

- Many common requirements in applications
  - Transaction, Logging and audit, Security, and much more
- These are not implemented by neither OS nor Application developer
  - They are called middleware
- Application servers provide middleware services
  - Application components live inside application servers
Application Servers

- Existing technologies can be classified into three broad categories
- Java based platform (Java Enterprise Edition)
- .NET Framework
- Other web application development frameworks
  - PHP frameworks: Zend, …
  - Ruby on Rail
  - …
Java Enterprise Edition
The Enterprise Today

- Availability 7×24
- Performance
- Extensibility
- Security
- Scalability
- Integration

Enterprise
- a project, typically one that is difficult or requires effort.
- a business or company.
The Java™ Platform

- Java Technology Enabled Devices
- Java Technology Enabled Desktop
- Workgroup Server
- High-End Server

Micro Edition

Standard Edition

Enterprise Edition
The Java™ Platform

Java 2 Platform Micro Edition (J2ME™)

Java 2 Enterprise Edition (J2EE)
Java 2 Standard Edition (J2SE)

Optional Packages
Optional Packages

Personal Basis Profile
Personal Profile
Foundation Profile
MIDP
CDC
CLDC

JVM
KVM
CardVM

Java Card APIs

Sadegh Aliakbary
Java EE

- Java Platforms
  - Java Card: Smart card version
  - Java ME (Micro Edition): Embedded systems, e.g. Mobile handheld
  - Java SE (Standard Edition): Desktop application development
  - Java EE (Enterprise Edition): Enterprise distributed application software
- Java EE add standards and libraries to SE for fault-tolerant, distributed, multi-tier based components
  - Until java 5, it has been called J2EE
JavaEE

- JavaEE platform is a simple, unified standard for distributed applications through a component-based application model
- Provides a component-based approach to the design, development, assembly, and deployment of enterprise applications
- It’s based on 3-tier Application Architecture
J2EE Components

- J2EE Client
  - Web Client (DHTML, HTML, XML,...)
  - Applet
  - Application Client

- J2EE Server
  - Web Component
  - Business Component

- Enterprise Information System (EIS)
  - DBMS,...
Web Client

- Web pages containing various types of markup language (e.g. HTML, XML), which are generated by web components running in the web tier
- Web Browser
- is called thin client
Application Client

- It typically has a graphical user interface (GUI) created from Swing or AWT APIs, but a command-line interface is certainly possible.
- Application clients directly access enterprise beans running in the business tier.
- is called thick client
J2EE Server

Client Tier

- Web Browser
- Web Pages, Applets, and Optional JavaBeans Components
- Application Client and Optional JavaBeans Components

Web Tier

Business Tier

J2EE Server
Java EE

- Java EE provides technologies (libraries) for enterprise level applications
- Java EE technologies for web applications
  - Servlet
  - JavaServer Pages
  - JavaServer Faces
  - Java Enterprise Beans
- Many other required libraries
  - Remote method invocation, Security, Database connectors, XML, …
Java EE Standards and Technologies

- Java API for RESTful Web Services (JAX-RS)
- Web Services
- Java API for XML-Based Web Services (JAX-WS)
- Java Architecture for XML Binding (JAXB)
- Java API for XML-based RPC (JAX-RPC)
- Java APIs for XML Messaging (JAXM)
- Java Servlet
- JavaServer Faces (JSF)
- JavaServer Pages (JSP)
- JavaServer Pages Standard Tag Library (JSTL)
- Enterprise JavaBeans (EJB)
- Java Persistence API (JPA)
- Java EE Connector Architecture
- Java Message Service API (JMS)
- Java Transaction API (JTA)
- JavaMail API
- Java Authentication Service Provider Interface for Containers (JASPIC)
- Java Authorization Service Provider Contract for Containers (JACC)
Containers

- Containers provide the runtime support for Java EE applications components.
- Containers provide a view of the underlying Java EE API to the application components.
- Java EE application components never interact directly with each other.
  - They use the protocol and methods of the container for interacting.
  - Remote Procedure Invocation (RMI).
Java EE Presentation Tier Components

- **Client side**
  - Client can use HTML, Java Applet, Java Application, …

- **Server side**
  - **Servlets** are special classes to realize the request-response model (get, post of HTTP)
    - External server side code
  - **JSP** is a developer-friendly wrapper over the servlet classes
    - Embed server side code
  - **Faces** & Facelets similar to JSP but uses custom tags which can be converted to anything
Java EE Presentation Tier Components

- JavaServer Faces
- JavaServer Pages Standard Tag Library
- JavaServer Pages
- Java Servlet
Servlet

- A Java application run on the web server in response to HTTP GET or POST requests
- Servlet is used to generate dynamic content to return to browser: HTML, XML, ...
- Servlet is a Java program that runs as separated thread inside servlet container
  - Servlet container is part of web server
    - It interacts with web client using the request/response paradigm
The Servlet Model

Browser

HTTP
Get/Post

Java
Enabled
Web
Server

Servlet
Servlet
Servlet
container

Servlet

Resources
JavaBeans
Database

HTML
XML
Servlet (cont’d)

- Servlet container runs servlets and send back their output to web client
  - HTML page is produced by print statements
    - `out.println("<html>"); ...`
- Loaded into memory once and then called many times
  - Performance enhancement
- Provides APIs for session management, access to GET/POST data, ...
Servlet Implementation

- Servlet container provides API for session & request management through implicit objects
  - **Session** object: Session management
  - **Request** object: Access to request fields: headers, cookies, …
  - **Response** object: The response object is used to build the HTTP response
- When a request for the servlet is received, the servlet engine spawns a new thread and calls appropriate service method
  - **doGet**: Process HTTP GET requests
  - **doPost**: Process HTTP POST requests
  - **doDelete**, **doPut**, …
- **destroy()** is called by to destroy the servlet
  - On web application shutdown or to release some resources
  - By default, servlets are kept alive as long as possible
public class HelloServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response) 
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        out.println("<HTML><n" +
            "<HEAD><TITLE>Hello</TITLE></HEAD><n" +
            "<BODY><n" +
            "<H1>Hello World</H1><n" +
            "</BODY></HTML>");
    }
}
web.xml => servlet

```xml
<servlet>
  <servlet-name>Manager</servlet-name>
  <servlet-class>org.apache.catalina.manager.ManagerServlet</servlet-class>
</servlet>
```
web.xml => servlet-mapping

```xml
<servlet-mapping>
  <servlet-name>Manager</servlet-name>
  <url-pattern>/text/*</url-pattern>
</servlet-mapping>
```
Servlets vs. CGI Scripts

- **Advantages:**
  - Running a servlet doesn’t require creating a separate process each time
  - A servlet stays in memory, so it doesn’t have to be reloaded each time
  - Untrusted servlets can be run in a “sandbox”
    - A secured environment
- **Disadvantage:**
  - Servlets must be in Java
  - CGI scripts can be in any language
JavaServer Pages (JSP)

- JavaServer Pages technology is an extension of servlet
  - It is the embed version of servlet in HTML
    - JSPs are easier to develop than servlets
  - It runs on the web server tier
- Contains some static HTML and some JSP tags in .jsp file, Java code inside the tags creates dynamic content (similar to PHP)
- When JSP is run, it creates a servlet
JSP Example

```html
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Processing "get" requests with data</title>
</head>
<body>
  <% // begin scriptlet
      String name = request.getParameter("firstName");
      if ( name != null ) {
        %>
        <h1>Hello <%= name %>, <br />Welcome to JavaServer Pages! <h1>
      <% // continue scriptlet
      } else {
        %>
        <form action="welcome.jsp" method="get">
          <p>Type your first name and press Submit</p>
          <p><input type="text" name="firstName" />
          <input type="submit" value="Submit" />
        </form>
      <% // continue scriptlet
      } // end else
  %>
</body>
</html>
```
JSP Invocation

- **Request**
  - JSP page (.JSP)
  - JSP translator (Tomcat)
    - Servlet Source Code (Java)
  - Execution phase
    - Text buffer (in memory)
    - Server class (.Class)

- **Translation phase**
  - JSP Container

(a) Translation occurs at this point, if JSP has been changed or is new.
(b) If not, translation is skipped.
JSP Advantages

- Performance
  - Runtime characteristics of servlet
  - Server side complex processing

- Programming
  - Easier to develop
  - Automatic recompilation of modified pages
  - More natural way to dynamic web pages
JSP in Summary

- In comparison to interpreted scripts (e.g., PHP)
  - JSP is compiled
    - More safety & better performance
  - Compiled servlet is in memory
    - Better performance
  - Converted to Servlet (a complete Java program)
    - Full OOP!!
    - More complex logic implementation
JavaServer Faces

- A user interface framework for building web applications

JavaServer Faces Components

- A GUI component framework
  - A set of custom markup tags `<h:form>`, `<h:head>`
- A flexible model for rendering components in different kinds of HTML or different markup
  - A Renderer object generates the markup to render the component & view its data
- A standard RenderKit for generating HTML/4.01
JavaServer Faces Components

- **Backing beans**
  - The logic of application
    - Java classes using Java EE beans
- **Facelet**
  - The view of application
    - XHTML file using component tags
- **Application configuration & description**
  - Mapping between Facelets & Beans
  - File organization, …
EJB

- EJBs are *distributed components* used to implement business logic (no UI)
- Developer concentrates on business logic
  - Availability, scalability, security, interoperability and … handled by the J2EE server
- Client of EJBs can be JSPs, servlets, other EJBs and external applications
- Clients see *interfaces*
EJB Types

- Session Beans
  - *Synchronous Action*: Process oriented service providers
  - Example: Credit Authorization

- Entity Beans
  - *Data*: Represent data
  - Example: Customer, Account

- Message-Driven
  - *Asynchronous Action*: Never called directly, only receive messages
  - JMS
  - Example: Transaction logging
Java EE Summary

[Diagram of Java EE architecture with layers and components such as Client Tier, Middle Tier, and EIS Tier, including JSP, Servlet, EJB, JDBC, DBMS, ERP, Main-frame, and Etc.]

Key:
- Client-side component
- Server-side component
- Backend component
- Backend connectivity via API
- HTTP
- Remote procedure invocation

Sadegh Aliakbary
MVC

What is design pattern?
- Designing complex SW system is really difficult
- Design patterns help us to design in methodological manner

Model-View-Controller (MVC)
- Model
  - Contains data, system state, and logic
- View
  - Presents data and system state
- Controller
  - Handles events/requests affecting model or view
MVC?!
MVC Interactions (cont’d)

- **Model**: Updates controller and view
- **View**: Queries model
- **Controller**: Updates model
- **Event**: Passed to controller

- View queries model
- Model signals changes
- Controller updates view
- Event is passed to the controller
MVC in Web Applications

- **Model** consists of data and system state
- **Database tables**
  - Persistent data
- **Current system state data**
- **Business logic (eCommerce)**
  - Rules governing the transaction
MVC in Web Applications (cont’d)

- **View** gives a presentation of the model
  - Client-side presentation in a browser window
    - (D)HTML
    - CSS style-sheets
    - Server-side templates
- **Administrative information**
  - Server output logs
MVC in Web Applications (cont’d)

- **Controller** handles events
- **User-generated events**
  - Client-side scripting
  - HTTP request processing
  - Redirection
  - Pre-processing
MVC in Java EE: Approach 1

- The pure JSP approach
- Separate JSP pages are used for the controller and the view
- Beans being used for the model part
- This is a good approach when the development is heavy in graphic designers and light in Java programmers
- Relatively complex applications can be constructed with the use of only JSP
  - Is also well suited for prototyping Web applications
MVC in Java EE: Approach 2

- A combination of servlets, JSP, and beans
- A servlet accepts requests and implements business logic
  - The servlet that receives requests can use other servlets to handle various kinds of requests
- Beans store and perform basic data manipulation
- JSP implements the user views of results of requests
- Conclude:
  - Use servlets to implement the controller
  - JSP to implement the view
Application Server vs. Servlet Container

- A servlet-container supports only the servlet API
  - including JSP, JSTL
  - e.g. Apache Tomcat

- An application server supports the whole JavaEE
  - EJB, JMS, JTA, Servlets, etc.
  - E.g. JBoss
Servlet Containers

- Apache Tomcat

- Jetty
  - Eclipse foundation
Application Servers

- Apache Geronimo
  - Tomcat or Jetty as the servlet container
- JBoss
  - An embedded Apache Tomcat
  - JBoss, Red Hat
- WebLogic
  - BEA => Oracle
- GlassGish
  - Sun => Oracle
  - The reference implementation of Java EE
  - A derivative of Apache Tomcat as the servlet container
- Websphere
  - IBM
Oracle, BEA, Sun, ...
A closer look at JSP and Servlet (more practical)
JSP Scripting Elements

- There are four types of scripting elements defined
  - Declaration
  - Expression
  - Scriptlets
  - Comments
Declaration

Declares a variable or method valid in the scripting language used in the JSP page

JSP Syntax

<%! declaration; [ declaration; ]+ ... %>

Examples

<%! int i = 0; %>
<%! int a, b, c; %>
<%! Circle a = new Circle(2.0); %>
Expression

JSP Syntax
<% = expression %>

Description
An expression that is converted to a String

Example
Welcome, <%= userName %> 

Output:
Welcome, James
Expression

<%=new java.util.Date()%>

The resulting servlet code will probably look like this:

out.print(new java.util.Date());
Scriptlets

- Contains a code fragment valid in the page scripting language
- Scriptlets allows you to include a series of java statements
- you must terminate them with semicolon.

JSP Syntax

<% code fragment %>
<%  String name = null;  
    if (request.getParameter("name") == null) { %>
      < %@ include file="error.html" %>
    <%  }
    else {
      userObject.setName(request.getParameter("name"));
    }
  %>
Comments

To denote any lines you want to be completely ignored by the JSP translation process.

Example

<%-- Author: James Gosling --%>
The taglib directive

- Declares that the JSP page uses custom tags
- Names the tag library that defines them
- and specifies their tag prefix.
- Defines a tag library and prefix for the custom tags used in the JSP page.

**JSP Syntax**

```jsp
<%@ taglib {uri="URI" | tagdir="/WEB-INF/tags[/subdir]+"} prefix="tagPrefix" %>
```

**Examples**

```jsp
<%@ taglib uri="http://www.jspcentral.com/tags" prefix="public" %>

<public:loop> ... </public:loop>
```
JSP Implicit objects

- Implicit objects are being created by JSP mechanism automatically.
- They are accessible from the JSP pages:
  - request – represents the HTTP request, which is being serviced by the JSP page; it is an instance of a class, implementing `javax.servlet.http.HttpServletRequest` interface;
    - `getParameter` (only strings)
    - `getAttribute` & `setAttribute` (any objects)
  - response – represents the HTTP response, receiving by the JSP page; it is an instance of a class, implementing `javax.servlet.http.HttpServletResponse` interface;
JSP Implicit objects (cont.)

- **session** – an instance of `javax.servlet.http.HttpSession`, representing an HTTP session;
  - `getAttribute` & `setAttribute` (any objects)
- **application** – represents the **servlet context** for the Web application; it is instance of `javax.servlet.ServletContext` class;
  - `getAttribute` & `setAttribute` (any objects)
- **out** – instance of `javax.servlet.jsp.JspWriter` class, which is being used to write content in the JSP output;
Example

<% 
Integer userviews = (Integer)session.getAttribute("userviews");
if(userviews==null)
    userviews = 0;
session.setAttribute("userviews",++userviews);
%>

<HTML> <BODY>

Number of User Views: 
<%=userviews%>

</BODY></HTML>
Example

```jsp
<% 
Integer usersview = (Integer)application.getAttribute("usersview");
if(usersview==null)
    usersview = 0;
application.setAttribute("usersview",++usersview);
%>

<HTML><BODY>
Number of Users Views:  <%=usersview%>
</BODY></HTML>
```
Example

```jsp
<% request.setAttribute("result", new Double(5.5)); %>
<jsp:forward page="display.jsp" />

<% Double number = (Double)request.getAttribute("result"); String param = request.getParameter("query"); %>

<HTML> <BODY>
Computed Result: <%=number%>
Query Parameter: <%=param%>
</BODY> </HTML>
```

Entering:
http://localhost:8080/app1/3.jsp?query=salam

Results:
Computed Result: 5.5 Query Parameter: salam
Example

```jsp
<% request.setAttribute("result",new Double(5.5)); response.sendRedirect("display.jsp"); %>

```
New Presentation-layer Technologies

- Nowadays, enterprise applications usually use other technologies in presentation layer
- JSF, GWT, Wicket, SpringMVC, Vaadin, …
- But we should know the architecture of JSP/Servlets
- Many technologies are built on servlet
- Servlet concepts are still important and useful
Exercise

- Write a simple JSP file
- Deploy it on Tomcat
- See how it works
- See the translated Servlet
- See the tomcat folders and files
Review some concepts

- Request/Response
  - request.getParameter()
  - request.getAttribute()
  - request.setAttribute()
    - Why?!
    - Forwarding requests

- Session/Application (ServletContext)
  - Share variables
  - getAttribute/setAttribute

- Redirect/Forward
Getting Information From Requests

- **Parameters**
  - used to convey information between clients and servlets
  - String bookId = request.getParameter("Add");

- **Object-valued attributes**
  - used to pass information between the servlet container and a servlet or between collaborating servlets
  - request.setAttribute("id", theObject);
  - Object identifier = request.getAttribute("id");
Information about

- the protocol
- The method (get, put, …)
- Request path
- Headers
- Query String
- …
Constructing Responses

- Indicate the content type
  - `response.setContentType("text/html");`

- Indicate whether to buffer output
  - By default, any content written to the output stream is immediately sent to the client
  - `response.setBufferSize(8192);`

- Retrieve an output stream
  - To send character data, use the `PrintWriter` returned by the response's `getWriter` method
  - To send binary data in a MIME body response, use the `ServletOutputStream` returned by `getOutputStream`

- Using Output stream
  - `output.println("<html>\");`
Invoking Other Web Resources

- To invoke a resource available on the server that is running a web component, you must first obtain a RequestDispatcher using the `getRequestDispatcher("URL")` method.

- To include another resource, invoke the include method of a RequestDispatcher:
  - `include(request, response);`

- To forward to another resource, invoke the forward:
  - `forward(request, response);`
Example: Transferring Control to Another Web Component

```java
public class Dispatcher extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response) {
        ...
        request.setAttribute("avg", new Double(18.5));
        RequestDispatcher dispatcher = request.getRequestDispatcher("/template.jsp");
        if (dispatcher != null)
            dispatcher.forward(request, response);
    }
    public void doPost(HttpServletRequest request, ...
```
The application object in JSP is called the ServletContext object in a servlet

The context in which web components execute is an object that implements the ServletContext interface

We can retrieve the web context with the getServletContext() method

```java
public class MyServlet extends HttpServlet{
    private ServletContext ctx = null;
    @Override
    public void init(ServletConfig config) throws ServletException {
        ctx = config.getServletContext();
    }

    ...
```
Session: Maintaining Client State

- Sessions are represented by an HttpSession object
- You can access a session by calling the getSession() method of a request object

```java
HttpSession session = request.getSession();
session.setAttribute("object", obj);
```
Servlet Container Folder Structure

- bin
  - startup
- conf
  - server.xml
- lib
- logs
- temp
- webapps
- work

War files
A web-app Structure

- Html, css, js, JSPs
- WEB-INF
  - web.xml
  - classes
  - lib
web.xml

- An xml file
- Contains
  - Servlet definitions
  - Servlet-mappings
  - Filter definitions
  - Filter-mappings
  - Error-pages
  - ...

web.xml => servlet

```xml
<servlet>
  <servlet-name>Manager</servlet-name>
  <servlet-class>org.apache.catalina.manager.ManagerServlet</servlet-class>
  <init-param>
    <param-name>debug</param-name>
    <param-value>2</param-value>
  </init-param>
</servlet>
```
web.xml => servlet-mapping

```xml
<servlet-mapping>
    <servlet-name>Manager</servlet-name>
    <url-pattern>/text/*</url-pattern>
</servlet-mapping>
```
web.xml => error pages

<error-page>
  <error-code>401</error-code>
  <location>/WEB-INF/jsp/401.jsp</location>
</error-page>

<error-page>
  <error-code>403</error-code>
  <location>/WEB-INF/jsp/403.jsp</location>
</error-page>

<error-page>
  <error-code>404</error-code>
  <location>/WEB-INF/jsp/404.jsp</location>
</error-page>

- 200 OK
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
Import in JSP

- **Import in Java**
  - Import classes: includes the mentioned class in the program
  - Example:
    - `import java.util.ArrayList;`
    - `import java.sql.Connection;`

- **Import in JSP:**
  - `<%@ page import="CLASS_NAME" %>`
  - Example:
    - `<%@ page import="java.util.List" %>"
Filter

- Acts as preprocessor to request/response for target servlet
- Extracts common scenario among different servlets
- Applications?
  - Authentication
    - SSO
  - Statistics
  - Log
  - ...

`web.xml => filter`

```xml
<filter>
  <filter-name>SetCharacterEncoding</filter-name>
  <filter-class>org.apache.catalina.filters.SetCharacterEncodingFilter</filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>UTF-8</param-value>
  </init-param>
</filter>

<filter-mapping>
  <filter-name>SetCharacterEncoding</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```
Filter Servlets
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class PageHits extends HttpServlet implements Filter {
    private FilterConfig filterConfig = null;

    public void init(FilterConfig filterConfig) throws ServletException {
        this.filterConfig = filterConfig;
    }

    public void destroy()
    {
        this.filterConfig = null;
    }
}
public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain) throws IOException, ServletException {

    if (filterConfig == null)
        return;

    Integer counter = (Integer) filterConfig.getServletContext().getAttribute("Counter");

    if (counter == null)
        counter = new Integer(0);
    counter = new Integer(counter.intValue() + 1);
    filterConfig.getServletContext().log("Number of hits is " + counter);
    filterConfig.getServletContext().setAttribute("Counter", counter);

    chain.doFilter(req, resp)
}
<web-app>

... 

<filter>
  <filter-name>PageHits</filter-name>
  <filter-class>ir.ac.sbu.PageHits</filter-class>
</filter>
<filter-mapping>
  <filter-name>PageHits</filter-name>
  <url-pattern>/payment/*</url-pattern>
</filter-mapping>

</web-app>
Listener Servlet

- Servlet is automatically executed when some external event occurs
- Event Listeners

<table>
<thead>
<tr>
<th>Event Listener</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HttpSessionActivationListener</td>
<td>Session is activated/passivated</td>
</tr>
<tr>
<td>HttpSessionAttributeListener</td>
<td>Session attribute is added/removed</td>
</tr>
<tr>
<td>HttpSessionListener</td>
<td>Session attribute is created/destroyed</td>
</tr>
<tr>
<td>ServletContextAttributeListener</td>
<td>Servlet context attribute is added/removed</td>
</tr>
<tr>
<td>ServletContextListener</td>
<td>Servlet context changes</td>
</tr>
</tbody>
</table>
What Events to Listen?

- **ServletContextListener**
  - contextInitialized
  - contextDestroyed

- **HttpSessionListener**
  - sessionCreated
  - sessionDestroyed

- **HttpSessionAttributeListener**
  - attributeAdded
  - attributeRemoved
  - attributeReplaced

- **ServletRequestAttributeListener**
  - attributeAdded
  - attributeRemoved
  - attributeReplaced

- **ServletContextAttributeListener**
  - attributeAdded
  - attributeRemoved
  - attributeReplaced
public class MyListener implements HttpSessionListener, HttpSessionAttributeListener, HttpSessionActivationListener{

@Override public void sessionDidActivate(HttpSessionEvent p) {} 
@Override public void sessionWillPassivate(HttpSessionEvent p) {} 
@Override public void attributeAdded(HttpSessionBindingEvent p) {} 
@Override public void attributeRemoved(HttpSessionBindingEvent p) {} 
@Override public void attributeReplaced(HttpSessionBindingEvent p) {} 
@Override public void sessionCreated(HttpSessionEvent p) {} 
@Override public void sessionDestroyed(HttpSessionEvent p) {} 
}
Create Listener Servlet

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Listener extends HttpServlet implements ServletContextListener {
    private ServletContext context = null;

    public void contextInitialized(ServletContextEvent event) {
        context = event.getServerContext();
        Integer counter = new Integer(0);
        context.setAttribute("Counter", counter);
        context.log("Created Counter");
    }

    public void contextDestroyed(ServletContextEvent event) {
        event.getServletContext().removeAttribute("Counter");
    }
}

Must implement ServletContextListener
Must override contextInitialized method
Must override contextDestroyed method
<web-app>
  <servlet>
    ...
  </servlet>
  <servlet-mapping>
    ...
  </servlet-mapping>
  ...
  <servlet-mapping>
    ...
  </servlet-mapping>
  <filter>
    ...
  </filter>
  <filter-mapping>
    ...
  </filter-mapping>
  <filter-mapping>
    ...
  </filter-mapping>
  <listener>
    <listener-class>Listener</listener-class>
  </listener>
</web-app>
public void doFilter(ServletRequest request, ServletResponse response,
    FilterChain chain) throws IOException, ServletException
{
    if (filterConfig == null)
        return;
    synchronized (this)
    {
        Integer counter = null != filterConfig.getServletContext().getAttribute("Counter") ?
            (Integer) filterConfig.getServletContext().getAttribute("Counter") :
            new Integer(1);
        counter = new Integer(counter.intValue()+1);
        filterConfig.getServletContext().log("Number of hits is "+ counter);
        filterConfig.getServletContext().setAttribute("Counter", counter);
    }
    chain.doFilter(request, response);
}
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {

    if (filterConfig == null)
        return;

    synchronized (this) {
        Integer counter = (Integer) filterConfig.getServletContext().getAttribute("Counter");
        counter = new Integer(counter.intValue() + 1);
        filterConfig.getServletContext().log("Number of hits is " + counter);
        filterConfig.getServletContext().setAttribute("Counter", counter);

        chain.doFilter(request, response);
    }
}
Conclusion
Conclusion

- Tiers and Layers
- MVC
- JavaEE
  - Java Editions
  - JSP
  - Servlet
  - JSF
  - EJB
  - Listener
  - Filter
  - Servlet container file/folder structure
Which layer?
Client side or Server side?
Need container or App server?

- JSP
- JPA
- JSF
- Servlet
- Hibernate
- EJB
- Spring
- SpringMVC
- Web Service
- CSS
- HTML
- Applet
- Flash
- Struts
- JDBC
- Logging
- GWT
- Javascript
- JavaFX
- Silverlight
- AJAX
Exercise

- Write a JSP/Servlet application
- Contact List app
  - User login form
  - Data Entry
  - List
  - Add MVC pattern
- Use an IDE for development
  - NetBeans
  - Eclipse JavaEE IDE (Formerly named WTP)
  - IntelliJ IDEA
References and Material

- The Java EE 6 Tutorial, Oracle
- JavaCup Exercise (www.javacup.ir)
- J2EE Workshops, Seyyed Jamaleddin Pishvayi
  http://asta.ir
- Internet Engineering course, Amirkabir University of Technology, Dr. Bahador Bakhshi
  http://ceit.aut.ac.ir/~bakhshis/
Any questions?