


Java Servlets & Java Server Pages

Brad Rippe
Fullerton College 

What is a Servlet?

- A servlet is a java class that extends an application hosted on a web server.
- Handles the HTTP request-response process (for our purposes)
- Often thought of as an applet that runs on a server.
- Provides a component base architecture for web development, using the Java Platform
- The foundation for Java Server Pages (JSP).
- Alternative to CGI scripting and platform specific server side applications.

Common Uses

- Processing and/or storing data submitted by an HTML form.
- Providing dynamic content, e.g. returning the results of a database query to the client.
- Managing state information on top of the stateless HTTP, e.g. for an online shopping cart system which manages shopping carts for many concurrent customers and maps every request to the right customer.

Architectural Roles

- Servlets provide the middle tier in a three or multi-tier system.
- Servlets provide logic and/or traffic control for requests/response to the server.
- Servlets allow web developers to remove code from the client and place the logic on the server.

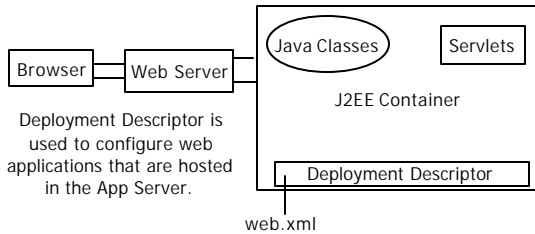
Architectural Roles

- Connection management
- Business rule enforcement
- Transaction management
- Mapping clients to a redundant set of servers
- Supporting different types of clients such as pure HTML, WML clients, other Java based clients

What is required?

- Servlets are not run in the same sense as applets and applications.
- Since they extend a web server, they require a servlet container to function.
- A servlet container is a part of a web server or application server that provides the network services over which request and response are sent.
- Contains and manages the servlets through there lifecycle.
- The container provides other services as well.


Web Container/Servlet engine



Web Application Directory Structure

- Application Root (htdocs/wwwroot)
- WEB-INF/web.xml
- WEB-INF/classes
 - Compiled servlet classes and other utility classes
- WEB-INF/lib
 - Any required third party jars
- All J2EE compliant App Servers require this directory structure

Common Containers Are Provided by the App Server

- Apache's Tomcat @ Jakarta 
 - <http://jakarta.apache.org/tomcat/index.html>
- BEA's WebLogic
- ATG Dynamo
- Allaire/MacroMedia's JRun
- IONA iPortal
- Sun's Java Web Server * available with the J2EE SDK
 - <http://java.sun.com/docs/books/tutorial/servletrunner/index.html>

Overview of the Servlet API

- All servlets implement the Servlet interface.
- The API provides two classes that implement this interface, GenericServlet and HttpServlet
- Servlet API is specified in two packages, javax.servlet and javax.servlet.http

Servlet Interface

- This is a contract between the servlet and the web container.
- Guarantees that all containers can communicate with a servlet.
- `public void init(ServletConfig)`
- `public void service(request, response)`
- `public void destroy()`
- `public ServletConfig getServletConfig()`
- `public String getServletInfo()`

HttpServlet

- Abstract class that extends **GenericServlet**
- Provides additional methods that are called by the **service()** method automatically.
- Both methods throw **ServletException** and **IOException**
- Methods of most concern:
 - `protected void doGet(HttpServletRequestRequest, HttpServletResponse)`
 - `protected void doPost(HttpServletRequestRequest, HttpServletResponse)`

Servlet Life Cycle

- Instantiation – web container creates an instance.
- Initialization – container calls the instance's **init()**.
- Service – if container has request, then it calls the **service()** method
- Destroy – before reclaiming the memory (destroying the servlet), the container calls the **destroy()** method.
- Unavailable – instance is destroyed and marked for GC.

ResourceExampleServlet

- How do we get it to work?
 - Using Tomcat 3.2.3
 - Created the appropriate web directory structure
 - In "Web-Apps" directory add the following:
 - cis228/
 - cis228/WEB-INF
 - cis228/WEB-INF/classes
 - cis223/WEB-INF/lib
 - Create Servlet (ResourceExampleServlet) which extends HttpServlet and saved in the classes directory
 - Create web.xml and save it in the WEB-INF directory

Configuration - web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.2/EN" "http://java.sun.com/j2ee/dtds/web-
app_2.2.dtd">
<web-app>
  <servlet>
    <servlet-name>resourceExample</servlet-name>
    <servlet-class>ResourceExampleServlet</servlet-class>
    <init-param>
      <param-name>dbUser</param-name>
      <param-value>testUser</param-value>
    </init-param>
    [ Repeat above for each parameter - init-param ]
  </servlet>
</web-app>
```

ResourceExampleServlet

- Define the following methods:
 - **init()** – initialize all member variables – safe?
 - **doGet(HttpServletRequest, HttpServletResponse)**
 - Note: You should define the doPost() method as well, even if it just calls doGet()
 - **destroy()**

What about the Code? – init()

```
public void init() throws ServletException {
    dbUser = getServletConfig().getInitParameter(
        "dbUser" );
    driver = getServletConfig().getInitParameter(
        "driver" );
    dbUrl = getServletConfig().getInitParameter( "dbUrl"
    );
    exampleMsg = getServletConfig().getInitParameter(
        "exampleMsg" );
}
```

What about the code? – doGet()

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();

out.println("<html>");
out.println("<head><title>Resource Example
Servlet</title></head>");
out.println("<body>");
out.println("<H3>Resource Example Servlet</H3>");
out.println("<p>I could really do some damage if I only
knew how to use those " +
" JDBC libraries!<br>I mean I have:<br>" );
out.println("<ul>");
out.println("<li>User: " + dbUser );
out.println("<li>URL: " + dbUrl );
out.println("<li>Driver: " + driver );
```

What about the code? – destroy()

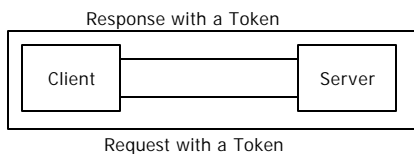
```
public void destroy() {  
    dbUser = null;  
    driver = null;  
    dbUrl = null;  
}
```

HttpServletRequest & HttpServletResponse

- Interfaces used for accessing request parameters and constructing client responses.
- The web container has concrete objects which implement these interfaces and passes into the servlet.

Servlet Sessions

- Since http protocol is stateless, Java API provides the following techniques for session tracking:
 - URL rewriting
 - Cookies
 - Hidden form fields



Servlet Sessions

- URL Rewrite – Facilitated through methods in the response interface
 - `http://bradsmachine.com?jsessionId=00988988`
- Hidden fields - `<input type="hidden" name="jsessionId" value="00988988">`
- Cookies
 - `Cookie c = new Cookie("uid", "brad");`
 - `c.setMaxAge(60);`
 - `c.setDomain("bradsmachine");`
 - `c.setPath("/");`
 - `response.addCookie(c);`
- Servlet API requires that web containers implement session tracking using cookies.

HttpSession

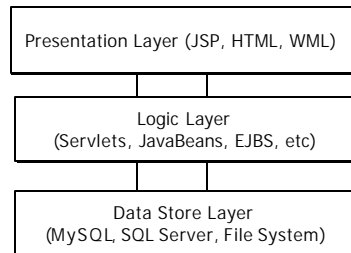
- Web containers provide an implementation of this interface to provide session tracking.
- Session objects can be retrieved from the `HttpServletRequest` object also provided by the container.
`HttpSession session = request.getSession();`
- If a session object does not exist for the client, one will be created.
- The duration of this session object can be configured in the `web.xml` file or in a servlet, `setMaxInactiveInterval(int interval);`

Session Example Servlet

Servlet Context

- Servlet API provides an interface for storing information that is relevant to all servlets being hosted in a web application.
- Common for facilities like application logging, access to other resources (Database Connection Pools), and request dispatching.
- There is one context per web application per JVM.
- Parameter information is configured in the web.xml file in name/value pairs.

Proposed Architecture of Web Applications



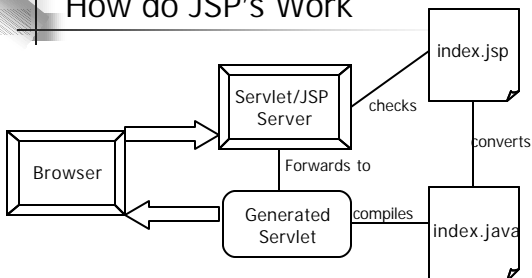
Highly Coupled Servlets

- High cost of maintenance
 - HTML and Java exist in the same file.
 - Web Designers don't understand java code, don't like the java code...
 - Programmers don't particularly like the messing with <HTML> code!!!!
 - Better to separate JAVA code from the HTML code.
- ```
if(developer == javaProgrammer)
 System.out.println("Stick to Java code!");
else if(developer == webDesigner)
 System.out.println("Stick to html code!");
```

## Java Server Pages

- Java Server Pages (JSPs) provide a way to separate the generation of dynamic content (java) from its presentation (html) (???)
- JSP specification builds on the functionality provided by the servlet specification.

## How do JSP's Work



## Basic JSP Syntax

- Contains html code like static html pages with the JSP tags and scripting included in the page.
- Three basic types of jsp tags
  - Scripting Elements
  - Directive Elements
  - Action Elements

## Scripting Elements

- Allow java code – variable or method declarations, scriptlet, and expressions.
- Declaration tag `<%! ... %>`
- Scriptlet tag `<% ... %>`
- Expression tag `<%= ... %>`

## Declaration Tag

- `<%! ... %>`
- Allows you to declare page wide variables and methods.
- `<%! int counter = 0; %>`
- `<%! Vector beanList = new Vector(); %>`
- Methods and variables have class scope
- Note, code must end with `;` like any java code

## Scriptlet Tag

- `<% ... %>`
- Used to include small pieces of Java code
- ```
<% for(Enumeration e = beanList.elements();  
e.hasMoreElements(); ) {  
    UserBean uBean = (UserBean) e.nextElement();  
    out.println( uBean.getUserName() );  
} %>
```

Expression Tag

- `<%= ... %>`
- Accepts any Java expression, evaluates the expression, converts to a String, and displays.
- `<%= counter %>`
- `<%= uBean.getUserName() %>`
- Short hand for:
■ `<% out.println(uBean.getUserName()); %>`

JSP Directives

- Directives provide global information to the JSP engine
- For example, a directive can be used to import java classes.
- Directive elements have a syntax of the form
- `<%@ directive ... %>`

Page Directives

- The page directive defines a number of page dependent attributes

```
<%@ page language="Java"  
[ extends="className" ]  
[ import="importList" ]  
[ session= "true|false" ]  
[ buffer="none|sizekb" ]  
[ autoFlush="true|false" ]  
[ isThreadSafe="true|false" ]  
... %>
```

Page Directive

- If language attribute is set, must be = "Java"
- Default import list is java.lang.*, javax.servlet.*, javax.servlet.jsp.* and javax.servlet.http.*.
- If session = "true" then default session variable of type javax.servlet.http.HttpSession references the current/new session for the page.

Include Directive

- The include directive is used to inline text and/or code at JSP page translation-time.
- `<%@ page include`
`file=""relativeURL" %>`
- `<%@ page include="/navbar.html"%>`

JSP Actions

- The syntax for action elements is based on XML(i.e. they have a start tag, a body, and an end tag).
- The JSP specification includes some action types that are standard.
- New action types are added using the taglib directive.

Standard Actions

- Web container implements these actions
- `<jsp:useBean>`
- `<jsp:setProperty>`
- `<jsp:getProperty>`
- `<jsp:include>`
- `<jsp:forward>`
- `<jsp:plugin>`
- `<jsp:param>`

Standard Actions

- `<jsp:useBean>`
- Associates an instance of a bean to a variable to use with in the jsp page
- `<jsp:useBean id="name"`
`scope="page|request|session|application"`
`class="className" type="typeName">`
- ...
- `</jsp:useBean>`

Standard Actions - useBean

- id – variable name to reference instance of class
- scope
 - page – javax.servlet.jsp.PageContext
 - Objects only accessible from within the page
 - request ServletRequest
 - Objects accessible in pages processing request where bean is created
 - session – HttpSession
 - Objects accessible from user session
 - application – ServletContext
 - Objects accessible in pages belonging to same application

Standard Actions - useBean

- Type (optional)
 - Allows scripting variable to be cast to another type from implementation class, java casting rules apply.
- Class
 - Fully qualified class name that defines the implementation of the object

Standard Actions - setProperty

- `<jsp:setProperty>`
- Sets the value of properties in a bean
- `<jsp:setProperty name="beanName" property="propertyName" (param="parameterName"|value="propertyValue")/>`
- Use `property="**"` to set all properties from the request

Standard Actions - setProperty

- Name
 - Variable name as defined in useBean
- Property
 - Name of the bean property
- Request
 - Name of the request parameter (if omitted same name as bean property name)
- Value
 - Value assign property (Performs necessary conversions)

Standard Actions - getProperty

- Gets the value of properties in a bean
- `<jsp:getProperty name="beanName" property="propertyName"/>`
- Name
 - Variable name as defined in useBean
- Property
 - Name of the bean property to retrieve

Bean Example

```
<html>
<title>Random JSP Test</title>
<body bgcolor="white">
<jsp:useBean id="rnd" scope="page"
  class="random.NumberGenerator"/>
<ul>
<li>Next number is: <jsp:getProperty
  name="rnd" property="nextInt"/>
</li>
</ul>
</body>
</html>
```

Bean Example

```
package random;
import java.util.*;
public class NumberGenerator {
    Random rnd = new Random();

    public int getNextInt() {
        return rnd.nextInt();
    }
}
```

Standard Actions – jsp:include

- `<jsp:include >`
- Allows you to include resources in the same context as the current page
- `<jsp:include page="url" flush="true"/>`
- `page:`
 - Relative url
- `flush:`
 - If true, buffer is flushed

Standard Actions – jsp:forward

- `<jsp:forward>`
- Allows you to dispatch the request to a different page
- `<jsp:forward page="url"/>`
- `page:`
 - Relative url

Standard Actions - <jsp:plugin>

- Creates HTML that contains OBJECT or EMBED constructs that result in Java Plugin download and execution of Applet
- `<jsp:plugin type="applet" code="applet.class" codebase="/html">`
- `<jsp:fallback>`
- `<p>Can't display applet</p>`
- `</jsp:fallback>`
- `</jsp:plugin>`
- Fallback is used if the plugin cannot be started

Standard Actions - jsp:param

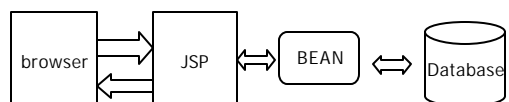
- `<jsp:param >`
- Used to provide key/value information for `<jsp:include >`, `<jsp:forward >`, and `<jsp:plugin >`
- `<jsp:param name="name" value="value"/>`
- Name and value are mandatory

Access Models

- Two approaches to building application with JSPs
- Model 1: JSP page processes all input
- Model 2: Servlet acts as a controller and directs http traffic to appropriate responses

Model 1

- JSP is responsible for processing incoming requests and replying to clients
- All data access is through beans

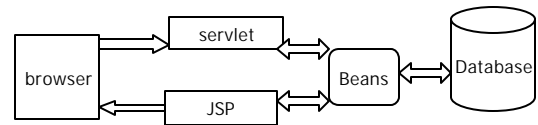


Model 1

- Suitable for simple applications
- Easier to understand
- May lead to lots of Java code embedded within JSP pages for complex applications
- (I've never used this model???)

Model 2

- Combine use of servlets and JSP
- Servlet acts as controller, JSP as presentation layer



It's a rap!

- JSP aren't total separation of logic from the presentation. Although it's a good start.
- Look into Custom tags to encapsulate your application logic and move it outside of the jsps. Creation of your own custom tag libraries help to eliminate java code from being embedded in the jsp.
- Frameworks like WebMacro, Struts, and Velocity provide additional features to

Resources

- Tomcath <http://jakarta.apache.org/tomcat>
- Struts <http://jakarta.apache.org/struts>
- Apache <http://www.apache.org>
- J2EE Web Site <http://java.sun.com/j2ee>
- Servlet 2.3 Spec <http://java.sun.com/products/servlet>
- JSP 1.2 Spec <http://java.sun.com/products/jsp>
- Free Development space for jsps/servlets
- Lecture available at <http://staffwww.fullcoll.edu/brippe/cis226>

Resources

- J2EE Developer's Guide – <http://java.sun.com/j2ee/j2sdkee/techdocs/guides/ejb/html/DevGuideTOC.html>
- J2EE Tutorial – http://java.sun.com/j2ee/tutorial/1_3-fcs/index.html
- Server Side Programming – <http://www.theserverside.com>