Introduction to J2EE Web Technologies

Kyle Brown
Executive Java Consultant
IBM WebSphere Services
RTP, NC
brownkyl@us.ibm.com

Overview

What is J2EE?
What are Servlets?
What are JSP’s?
How do you use them?
What is J2EE?

- API's for Server-side Java Programming
  - Comprehensive support for
    - Web Programming
    - Transactions
    - Distributed Objects
- Sun calls this "Java 2 Enterprise Edition" (J2EE)
- A direct competitor to Microsoft's .NET

Java 2 Platform, Enterprise Edition (J2EE)

- Standard Platform
  - Enables solutions for developing, deploying and managing n-tier server-centric enterprise application
- Application Programming Model
  - For developing multi-tier thin-client services
- Compatibility Test Suite
  - For verifying that a J2EE platform product complies with the J2EE platform standard
- Reference Implementation
  - Demonstrates the capabilities of J2EE and provides an operational definition of J2EE.
J2EE Concepts

- Containers
- Application Components
- Resource Managers (Drivers)
- Standard Services

Containers

- Containers provide the environment in which components execute
  - relatively simple APIs
  - Provide control by providing the calling framework

- Containers implement J2EE API's
  - Look the same to components
  - Have freedom to innovate internally
  - Allow vendors to differentiate themselves
Containers and Components

- Containers handle
  - Concurrency, Consistency, Security, Availability, Scalability, Administration, Integration, Distribution, Data Access

- Components handle:
  - Presentation, Business Logic, Data Access

J2EE Application Components

- Four types
  - Application Clients
  - Applets
  - Servlets and JavaServer Pages
  - Enterprise JavaBeans

- Three categories
  - Deployed, managed and executed on a J2EE server (EJB, JSP and Servlet)
  - Deployed and managed on a J2EE server but loaded and executed on a client machine (applets)
  - Not covered by J2EE spec (Application Clients)
Java and the Web

- Java Servlets are server-side Java classes that respond (typically) to client HTTP requests
  - Similar to a CGI program but faster

- Java Server Pages are a way for embedding Java code into server-side HTML pages
  - A template language for Java Beans and server-side HTML processing
**Java Servlets**

- A Java class that represents a single URL to the client
  - Defines a `service()` method that handles HTTP requests
  - `HttpServletRequest` -- access request data
  - `HttpServletResponse` -- reply to the client

- An instance of each class is a shared resource used by multiple threads
  - Each thread handles an HTTP request

**Generic Servlet Invocation**

- Client makes a request of a Servlet through an URL
- (Optional) Web Server forwards request to Web container
- Web Container locates an instance of a Servlet class
- Servlet engine calls Servlet's `service` method

![Generic Servlet Invocation Diagram]
The Java Servlet API

- The Servlet API includes two packages:
  - javax.servlet
  - javax.servlet.http

Servlet

- Represents a service
- Usually requested via URL
- Servlets are loaded by a Web Container
  - At initialization of Server (if preload)
  - At first client request
  - Upon servlet reload
Servlet Lifecycle

- The init() method is called at load time
  - One time behavior
- The service() method is invoked for each client request
- The destroy() method is called when it is unloaded

HttpServlet

- An HTTP-specific request handler
- Adds two HTTP specific methods:
  - doGet() -- handle a GET request (URL)
  - doPost() -- handle a POST request (HTML form)
- Subclasses override these two messages and may override init() and destroy()
- doGet() and doPost() do the work and are called by service()
Requests and Responses

- The service(), doGet() and doPost() methods each have two parameters:
  - HttpServletRequest -- provides access to request data (parameters), HttpSession information, etc.
  - HttpServletResponse -- provides services to allow the servlet to supply a reply to the requesting client

- Most servlet programming amounts to reading a request and writing a response

HttpServletRequest
- Represents communication channel back to client
- Allows servlet to return content and/or errors
- Set content header (type, length, ...)
- Redirect server to return a particular URL
**Response Protocol**

- `getWriter()`
  - Returns a PrintWriter for output

- `setContentType(String type)`
  - Set the content type for this response
  - Type is a MIME type

- `sendRedirect(String anURL)`
  - Redirect the browser to a new URL

---

**Simple Servlet**

```java
public class MyServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
        // get stream to output HTML on!
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        // send out a simple banner
        out.println("<HTML><BODY>");
        out.println("<h1>Hello World!</h1>");
        out.println("</BODY></HTML>");
    }
}
```
**HttpServletRequest**

- Represents client's request

- "Getters" for aspects of request, e.g.,
  - Request header, content type, length, method...
  - Request URL as a String
  - Servlet "path"
  - Client security type
  - Access request parameters (by name)
  - Scope for data sharing among participant objects in the request

---

**Request Protocol**

- `getParameterNames()`
  - Returns an Enumeration of parameters on the HTML page

- `getParameterValues(String name)`
  - Returns the value of a multi-valued parameter

- `getParameter(String name)`
  - Returns the value of a specific named parameter

- `getReader()`
  - Returns a BufferedReader to view input
Example HTML Form

Please fill out this form with your name.
Thanks!

```
<form method="POST" action="/servlet/NameServlet">
  <p>Please enter your name:</p>
  <p>First name: <input name="first" type="text" size="12" maxlength="20"></input>
  Surname: <input name="surname" type="text" size="15" maxlength="25"></input>
  <p>Thank you! <input type="submit"> <input type="reset"></input>
</form>
```

Reading a Post

```
public void doPost(HttpServletRequest req, HttpServletResponse res)
  throws ServletException, IOException
{
  Enumeration enum = req.getParameterNames();
  while (enum.hasMoreElements()) {
    String name = (String) enum.nextElement();
    String value = req.getParameter(name);
    //... do something with each pair...
  }
}
```
JavaServer Pages (JSP) is a standard HTML "template" language
- Embed standard JSP tags into an HTML page
- Embed Java code (scriptlets)

JSPs are converted to servlets at runtime
- Page compilation triggered by changes in the JSP
- JSP Source is parsed
- Java servlet code is generated
- This "JSP Servlet" is compiled, loaded and run

JSP Servlet Structure
- JSP file contents loaded into a buffer
- Subclass of HttpServlet created with overridden service method
- Service method is interleaving of:
  HTML copied to Response OutputStream
  Execution of Java equivalent of JSP notation
  ```java
  outstream.writeBytes(buf, 0, frontlen);
  outstream.print(new java.util.Date());
  outstream.writeBytes(buf, cntindx, rest);
  ```
JSP 1.1 Specification

- JSP elements
  - Directives
  - Scripting
  - Actions

- Scripting language support

JSP Directives

JSP directives are messages to the JSP engine

- Syntax in the form ..
  - `<%@ directive {attribute="value"} %>`
- 1.1 defines page, include, and taglib
  - `<%@ page language="java" %>`
  - `<%@ include file="companyBanner.html" %>`
  - `<%@ taglib uri="tagLibraryURI" prefix="tagPrefix" %>`
The `<%@ page %>` Directive

- Defines page-specific attributes
- Applies to complete translation unit

```<%@ page page_directive_attr_list %> where
    page_directive_attr_list :=
    language=language
    extends=className
    import=importList
    session=session
    buffer=buffer
    autoFlush=autoFlush
    isThreadSafe=isThreadSafe
    info=info
    errorPage=errorPage
    isErrorPage=isErrorPage
    contentType=contentType```

Scripting

- Declarations
  - `<%! declaration %>`
  - `jspInit()` and `jspDestroy()` methods may be defined here

- Scriptlets
  - `<% valid_code_fragment %>`
  - Java code makes up body of generated "method"

- Expressions
  - `<%= expression %>`
  - Semantics:
    1. The expression is evaluated
    2. Result is converted to a String and displayed
Within both Scriptlets and Expressions there are certain "implicit objects" available for use (without being declared first).

- **Implicit objects**
  - Request -- HttpServletRequest object
  - Response -- HttpServletResponse object
  - Session -- the current HttpSession
  - Out -- the JspWriter which writes into the output stream
  - PageContext, Application (ServletContext), Config (ServletConfig), Page
  - Exception -- Instance of Throwable (available to Error Pages)

Servlet and "back-end" supply dynamic content in a JavaBean

- JSP accesses object via `<jsp:usebean>` tag
- `<jsp:usebean>` tag specifies how to find or create a Bean

A Bean can be:
- Instantiated from serialized file or class file
- Retrieved from HttpSession, ServletRequest context or Application context
JSP Sample

```html
<html>
<head> ...
</head>
<body>
If this were a real application, you would confirm your information below and finalize your transaction.

```jsp:getProperty name="usr" property="firstName" /> <br/>
```jsp:getProperty name="usr" property="lastName" /> <br/>
```jsp:getProperty name="usr" property="street" /> <br/>
```jsp:getProperty name="usr" property="city" />,
```jsp:getProperty name="usr" property="state" />
```jsp:getProperty name="usr" property="zip" /> <br/>Data valid as of <%= new java.util.Date() %>
</body></html>
```

Enterprise Java and MVC

- The MVC Pattern is at the heart of Enterprise Java
- Model -- Represent the business logic
- View -- Represent a way of interacting with the model
- Controller -- Mediate between the two, and manage application flow
- Cleanly separates presentation (View) code from content (Model) code
1. clients make HTTP requests
2. servlet (controller) processes inputs and calls business logic (model)
3. business logic can be anything: beans, EJBs, JDBC, etc.
4. servlet calls the output JSP page (view)
5. output page processing produces HTML

---

**Calling JSPs From Servlets**

- Use a standard "RequestDispatcher" to forward requests to another web object.

```java
RequestDispatcher d;
request.setAttribute("beanName", theBean);
d = getServletContext().getRequestDispatcher("/aJSP.jsp");
dispenser.forward(request, response);
```
MVC Example

WebApps

A WebApp is a repository for application files. A web application may consist of:
- Servlets, JSP's, Utility Classes, Static html documents, Applets, etc.
- Descriptive meta information to tie all of the above together

A special subdirectory named "WEB-INF" contains:
- /WEB-INF/web.xml deployment descriptor
- /WEB-INF/classes/* directory for Java classes.
- /WEB-INF/lib/*.jar area for Java Archive files

Packaged using JAR into a .WAR (Web Archive) file
Web.XML file

```xml
<!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 id="WebApp">
  <display-name>NCSUDemoProject</display-name>
  <servlet>
    <servlet-name>ProcessRegistrationServlet</servlet-name>
    <servlet-class>com.ibm.ncsu.demo.servlet.ProcessRegistrationServlet</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>ShowAttendeeListServlet</servlet-name>
    <servlet-class>com.ibm.ncsu.demo.servlet.ShowAttendeeListServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ShowAttendeeListServlet</servlet-name>
    <url-pattern>ShowAttendeeListServlet</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>ProcessRegistrationServlet</servlet-name>
    <url-pattern>ProcessRegistrationServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

Example WAR Structure

/web-inf
/classes
   /com.ibm.ncsu/demo/domain/Attendee.class
   /com.ibm.ncsu/demo/domain/AttendeeBroker.class
   /com.ibm.ncsu/demo/servlet/ProcessRegistrationServlet.class
/web.xml
/index.html
/listattendees.jsp
/register.jsp
### The Session Problem

- Servlets must be stateless
  - They should not have instance variables
  - Are shared by multiple threads
- Temporary application state specific to a user must be stored outside the servlet
  - This is called Session State
  - Can be stored in an HttpSession object

### HttpSession

- Ask for a Session from a HttpRequest
  - HttpRequest.getSession(boolean create)
- Returns an HttpSession
  - If create is false, use a previously created session
- HttpSessions store user-specific information
  - void putValue(String, Object)
  - Object getValue(String)
Session Lookup

<table>
<thead>
<tr>
<th>Cookie name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;sessiid&quot;</td>
<td>12345</td>
</tr>
</tbody>
</table>

Application Server

<table>
<thead>
<tr>
<th>Id</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>&quot;session&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Customer&quot;</td>
<td>Customer</td>
</tr>
<tr>
<td>&quot;Name&quot;</td>
<td>&quot;Bob&quot;</td>
</tr>
</tbody>
</table>

Summary

- What Servlets and JSP's are
- How they are used
- What they are defined in J2EE