Introduction to J2EE Web Technologies

Kyle Brown
Senior Technical Staff Member
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?

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

Containers and Components

- Containers provide the environment in which components execute
  - Containers provide the implementation of the J2EE API's
    - All containers look the same to components
    - Give vendors the freedom to innovate internally and to differentiate themselves by providing different tools, quality of service, etc.
  - Provide control by providing the calling framework
  - 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

The Java Servlet API

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

```java
javax.servlet.Servlet
\triangleq \text{implements}
javax.servlet.GenericServlet
\triangleq \text{extends}
javax.servlet.http.HttpServlet
```
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
HttpServletResponse

- 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>\n        out.println("<h1>Hello World!</h1>\n        out.println("</BODY></HTML>\n    }
}
```

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

```html
<P>Please fill out this form with your name. Thanks!<br/>
<FORM METHOD="POST" ACTION="/servlet/NameServlet">
    <P>Please enter your name:<br/>
    <P>First name: <INPUT NAME="first" TYPE="TEXT" SIZE="12" MAXLENGTH="20">
    Surname: <INPUT NAME="surname" TYPE="TEXT" SIZE="15" MAXLENGTH="25">
    <P>Thank you! <INPUT TYPE="SUBMIT"> <INPUT TYPE="RESET">
</FORM>
```
Reading a Post

```java
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

- 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, or on first invocation
  - 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
  - Apply to complete translation unit
  - `<%@ page page_directive_attr_list %>` where `page_directive_attr_list`:

  language="scriptingLanguage"
  extends="className"
  import="packageList"
  session="true|false"
  buffer="none|sizekb"
  autoFlush="true|false"
  isThreadSafe="true|false"
  info="info_text"
  errorPage="error_url"
  isErrorPage="true|false"
  contentType="Type | Type; charset=CHARSET"
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:
  - The expression is evaluated
  - 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)
JavaServer Pages and JavaBeans

- 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>
<jsp:useBean id="usr" scope="request" type="com.ibm.demo.UserInfo"/>
<BODY>
If this were a real application, you would confirm your information below and finalize your transaction.
<p>
<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"/>
<br>
<jsp:getProperty name="usr" property="state"/>
<br>
<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

MVC Program Flow

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. output page processing produces HTML
5. servlet calls the output jsp page (view)
Calling JSPs From Servlets

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

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

MVC Example

- `index.html` 
- `register.jsp`
- `ListAttendeesServlet`
- `ProcessRegistrationServlet`
- `AttendeeBroker` 
- `Attendee`

- `void doPost()`
- `Collection parseAttendee()`
- `void doGet()`
- `void createAttendee(Attendee)`
- `void updateAttendee(Attendee)`
- `void deleteAttendee(Attendee)`
- `Collection getAllAttendees()`

- `String name`
- `String company`
- `String title`
- `String streetAddress`
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
  /com/ibm/ncsu/demo/domain/ListAttendeesServlet.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
HttpRequest

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

Session Lookup

<table>
<thead>
<tr>
<th>Browser</th>
<th>Application Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cookie List</td>
<td>id</td>
</tr>
<tr>
<td>cookie name</td>
<td>12345</td>
</tr>
<tr>
<td>&quot;sesessionid&quot;</td>
<td>12345</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>aSession</th>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Customer&quot;</td>
<td>aCustomer</td>
<td></td>
</tr>
<tr>
<td>&quot;Name&quot;</td>
<td>&quot;Bob&quot;</td>
<td></td>
</tr>
</tbody>
</table>
Scaling HttpSession

- While the previous (default) implementation works for one application server, it fails in a clustered environment where requests from a single user could be dispatched to more than one JVM over the course of a user session.
- There are several standard solutions to this:
  - Store the HttpSession information in a shared database
  - "Pin" each user to a specific JVM
  - Use a distributed notification mechanism to transfer session information between JVM's
- Each application server vendor will choose one or more of these solutions for clustered session state maintainance.

Advanced Topic: Servlet Filters

- One of the more interesting features added in the Servlet 2.3 specification was the notion of a Servlet Filter.
- Filters are similar to, but not the same as a servlet:
  - A Filter modifies a servlet request or response
  - Does not represent a user end-point
- The Filter interface defines a doFilter() method, which:
  - Examines the request headers
  - Customizes the request or response as needed
  - Passes control onto the next filter in the chain or blocks the response
Using Filters

- There are a number of different ways that filters can be used to change output or make general decisions based on input:
  - Logging and billing -- a logging or billing filter could be attached to all servlets (regardless of whether or not you have the source code) to log incoming HTTP requests and outgoing responses for audit purposes, or to record the userID and invocation information for later billing.
  - XSL -- an XSL Translation filter can take output formatted as XML and apply XSL Stylesheets to the output to translate it into HTML or some other format (like WML).
  - Internationalization -- an Internationalization filter could take abstract message ID's and replace them in the output with the proper national-language output text.
  - Content-filtering -- a content-filtering servlet could mask or remove output that a user is not authorized to see.

Summary

- We've seen:
  - What Servlets and JSP's
  - How they are used
  - What they are defined in J2EE