Servlets and Java Server Pages (JSP)
Andrew Gilbert
andrew.gilbert@sun.com
www.sun.com/developers/evangcentral

Senior Software Engineer
Sun Microsystems
Agenda

• What are Servlets?
• What are JSPs?
• Design issues
• Resources and Summary
What are Servlets?
What Are Servlets?

Servlets are Java’s answer to CGI/Perl...

* Server side Java™ program
  * That respond to web server requests
* Used to generate dynamic HTML
Servlets vs. CGI

- **Scalable**, uses **Lightweight** threads:
- Doesn’t start new process for each request, **Initialized once and persists in memory** for multiple requests, **cached**
- **Multi-threaded**
Why Servlets?

- Superior alternative to CGI:
  - Multi-threaded, Cached, better security
- Easy to program, Simple APIs for input arguments, cookies
- Written in Java
  - Can take advantage of JDBC, EJB, JMS, JavaMail, JavaIDL, RMI, APIs...
Servlet and JSP™ Run in Web Container
Servlet APIs

javax.servlet package

Servlet interface

GenericServlet

HttpServlet

HelloWorldServlet

Your Servlet
Servlet Example

Server

- init()
- service()
- destroy()

Request

Response

Session
Servlet Lifecycle

- **init()**: Called by the servlet engine once when the servlet is being loaded.
- **service()**: Called by the servlet engine to allow the servlet to respond to a request.
  - `doGet()`, `doPost()`: Specific methods for handling client requests.
- **destroy()**: Called when the servlet is removed.
Anatomy of a Request

- The servlet’s `service()` method is invoked with a `Request` and `Response` object.
- The servlet provides a response to the request.
Request and Response Parameters

**Request**: Encapsulates all **information from the client**

- **HTTP request** headers
- **InputStream** or reader
- **Input parameters**: Form data, cgi data

**Response**: Encapsulates all **communication back to the client**

- **HTTP response** headers
- **OutputStream** or **writer**
- Setting cookies, redirects, error pages
A Simple Servlet

Sample Code

```java
public class ExampleServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("Hello!<BR>”);
    }
}
```
What are Java Server Pages?
What is a Java Server Page?

JSP™ turns servlets inside out

– Presentation centric

– Java code embedded in a HTML page

– JSP™ is for formatting output such as HTML and XML
HTML to Java Bean Mapping

**JSP™ (Developer View)**

- HTTP Requests
- HTTP Response
- JSP™ Page
- Java Bean
- Java Bean
- Java Bean

**Check if page needs to be recompiled.**

- Web Server
- HTTP Requests
- HTTP Response
- JSP™ Page
- Compile page into servlet
- Java Bean
- Java Bean
- Java Bean

**Load if needed And run servlet**

- JAVA Servlet
- JSP™ (Physical View)
public class HelloServlet extends HttpServlet{
public void doGet(HttpServletRequest request,
HttpServletResponse response)  {
    response.setContentType("text/plain");
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("Hello World!");
    out.println("<br>");
    JspCalendar clock = new JspCalendar();
    out.println("Today is");
    out.println("<ul>");
    out.println("<li>Day of month: ");
    out.println("clock.getDayOfMonth()");
    out.println("</li>");
    out.println("</ul>");
    out.println("</html>");
}
Example JSP

<html>
    Hello World!<br>
    <jsp:useBean id="clock" class="calendar.JspCalendar" />
    Today is
    <ul>
        <li>Day of month:  
        <%= clock.getDayOfMonth() %>
    </ul>
</html>
Usage models of JSPs

Intent from the original JSP specifications:

- **Model 1:**
  - Browser calls JSP directly
  - JSP uses JavaBeans to encapsulate business logic
  - JSP delivers the response to the browser

- **Model 2:**
  - Browser calls servlet as coordinator
  - Servlet invokes contains business logic or delegates to JavaBeans
  - Servlet stores result of business logic in embedded objects in Response
  - JSP interrogates the Response and creates a page to send the browser
JSP Syntax Basics

• The JavaServer Pages specification includes:
  – Directives
  – Standard JSP™ Tags
  – Scripting elements
  – Implicit Objects
  – tag extensions
Example JSP standard tags:

```xml
<jsp:useBean id="clock"
    class="calendar.JspCalendar" />
<jsp:getProperty name="customer"
    property="name" />
<jsp:setProperty name="customer"
    property="name" param="username" />
```
JSP™ Standard Action Tags

• **Standard Action** tags used to access Java objects from the JSP™ page.

• Example JSP standard tag:

  ```jsp
  <jsp:useBean id="clock" class="calendar.JspCalendar" />
  ```

  Like `JspCalendar clock = new JspCalendar();`

  The JSP page accesses a bean object via a tag.
  
  – If a bean **doesn’t exist**, it is **instantiated**.
  
  – If bean **already exists**, it is **retrieved** from the **session** or the **request** context.
What is a JavaBean?

```java
public class AccountBean {
    private String firstName;
    private String lastName;
    public AccountBean() {}
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String f) {
        firstName = f;
    }
    ...
}
```

JavaBeans components are Java objects which follow a well-defined design/naming pattern.
JSP™ Standard Tags

Example JSP standard tags:

```html
<html>
<jsp:useBean id="accountBean" scope="session" class="AccountBean" />

<jsp:setProperty name="accountBean" property="*"/>

<form method=POST action=Account.jsp>
First Name
<input type=text name="firstName" />

<input type=submit name="submit" value="Submit">
</form>
</html>
```

**SetProperty:** Automatically execute all setters in accountBean that match values provided as input
JSP™ Standard Tags

Example JSP standard tags:

```html
<html>
<jsp:useBean id="accountBean" scope="session" class="AccountBean"/>
First Name =

getProperty: Retrieve value of named attribute

<jsp:getProperty name="accountBean" property="firstName"/>
</html>
```
JSP™ Scripting Elements

& **Declarations:** define page level variables and methods

  `<%! QuoteBean q; %>`

& **Scriptlets:** JSP Java code fragments

  `<% q = new QuoteBean(); %>`

& **Expressions:** results of evaluating the expression are converted to a string

  `<%= q.getQuote("SUNW"); %>`
Implicit Objects

• The JSP™ page has access to certain implicit objects that are always available

• They are the following:
  – request - HttpServletRequest
  – Response – HttpServletResponse
  – pageContext, HTTP session,
  – out, config, page, exception
JSP™ Sample

<%@ page info="My JSP™ example" import="myclasses.*" %>

<jsp:useBean id="event" class="com.sun.jspdemo.webImpl.Event" scope="session" />

<html>
<br>
<h3>The event name is: <jsp:getProperty name="event" property="name" /></h3>
<h3>The event description is: <%= event.getDescription() %> </h3>
<br>
<br>
<% if (event.getName().equalsIgnoreCase("appointment")) { %>
   event.processAppointment(request.getParameter("name")); %>
   <jsp:include page="appointment.html" /> 
<% } %>
else { %>
   <jsp:include page="regular.html" />
<% } %>

</html>
Portable Tag Extension Mechanism

Designing tag libraries allows content developers to use custom tags instead of java code.

```html
<%@ taglib uri="/WEB-INF/sqltaglib.tld" prefix="ora" %>
<HTML>
     <HEAD> <TITLE>SQL Tags Example </TITLE> </HEAD>
     <BODY BGCOLOR="#FFFFFF">
        <HR>
        <ora:dbOpen URL="jdbc:oracle:oci8:@" user="scott" password="tiger" connId="con1">
            password="tiger" connId="con1">
            <!-- generates HTML table -->
        select * from EMP
        </ora:dbQuery>
        <ora:dbClose connId="con1" />
        <HR>
     </BODY>
</HTML>
```
Design Issues
View Design Issues

• View Consists of two parts
  – Request Processing
    • (Part of the controller)
  – Response Generation
    • Presentation Layout
Tasks in Web Layer

• Validation of Input
  - Is form data complete and well-formed?

• Interaction with Business Logic

• Presentation of content
  - Get Data from Model Value Object
  - Present data to client
**View:** present User Interface (screen representation of model)

- **Request** → **Controller** → **Model** → **Updates**
- **Display** → **View** → **Selection** → **State Query (read only)**

- **View**
  - JSP
  - Custom JSP Actions
  - JavaBeans Components
When Servlets, When JSP?

• Use **servlets** for **dispatching** between the Web tier and EJB tier
  
  – **Request processing** => use **servlet** and/or **JavaBeans**

• Use **JSP™** technology for **response generation**
  
  – **Displaying HTML, XML** => use **JSP™** technology
How Do I Use All These?

• separation between content and application logic:
  - Move request processing out of JSP™ page.
  - JSP™ page is primarily presentation layout.

1. Parse Request

2. Prepare Java Beans

3. Forward Request to JSP™ Page
View Design Guidelines

Build View in the Web-tier

- Use JavaServer Pages™ (JSP™) components for presentation layout

- Custom tags and JavaBeans™ components for presentation logic
**All working together**

<table>
<thead>
<tr>
<th>Presentation</th>
<th>Application Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Logic</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Layout</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Business Services</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Business Data</strong></td>
<td></td>
</tr>
<tr>
<td><strong>main Servlet</strong></td>
<td></td>
</tr>
<tr>
<td><strong>View Selection</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Controller</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Session Bean</strong></td>
<td></td>
</tr>
<tr>
<td><strong>OrderEJB</strong></td>
<td></td>
</tr>
</tbody>
</table>

- **main Servlet**: Validates input, Calls session bean, ... Calls JSP™ passing Value Model object.
- **View Selection**: Formats HTML, Responds to client.
- **Controller**: Validates input, Calls session bean. ... Validates input, Calls JSP™ passing Value Model object.
- **Session Bean**: Purchases items(), controls process, Enforces transactions.
- **OrderEJB**: Entity Bean, Create(), Manages data.
- **Inventory**: Entity Bean, Update(), Manages data.
public void doPost (HttpServletRequest req, HttpServletResponse res) {
    String userId = req.getSession().getValue("userId");

    IAccountHome accountHome;
    IAccountBean account;

    Context initCtx = new InitialContext();
    accountHome = initCtx.lookup("com/AccountHome");
    account = accountHome.findByPrimaryKey(userId);
    AccountBean accountBean = account.getAccountDetails();

    Writer out = res.getWriter();
    out.println("<HTML><HEAD><TITLE>Your account</TITLE></HEAD>");
    out.println("<BODY>Name " + accountBean.getName() + ".<br>");
    out.println("Address: " + accountBean.getAddress() + ");;

}
JSPs are a presentation layer for Servlets...

```html
<HTML>
<HEAD><TITLE>Your account</TITLE></HEAD>
<BODY>
    <%AccountBean accountBean=(AccountBean)
        req.getAttribute("accountBean");
    %>
Name: <%=accountBean.getName()%> <br>
Address: <%=accountBean.getAddress()%> <br>
</BODY>
</HTML>
```
Example JSP

JSPs are a presentation layer for Servlets...

<HTML>
<HEAD><TITLE>Your account</TITLE></HEAD>

<BODY>
<jsp:useBean id="accountBean" scope="request" class="AccountBean"/>

Name: 
<jsp:getProperty name="accountBean" property="name" />

Address: 
<jsp:getProperty name="accountBean" property="address"/>

</BODY>

</HTML>
Example Servlet Dispatching to JSP

Servlets are Java's answer to CGI/Perl...

```java
public void doPost (HttpServletRequest req, HttpServletResponse res) {
    String userId = req.getSession().getValue("userId");
    IAccountHome accountHome;
    IAccountBean account;
    Context initCtx = new InitialContext();
    accountHome = initCtx.lookup("com/AccountHome");
    account = accountHome.findByPrimaryKey(userId);
    AccountBean accountBean= account.getAccountDetails();
    Writer out = res.getWriter();
    out.println("<HTML><HEAD><TITLE>Your account</TITLE></HEAD>");
    out.println("<BODY>Welcome “ + accountBean.getName + ".<br>
    Address:  " + accountBean.getAddress() );
}
```

Java Server Pages to the rescue!

```java
RequestDispatcher dispatcher;
ServletContext context = getServletContext();
```

Why do I care?

- Enables separation of presentation **logic** from presentation **layout**
- Enable your experts to play to their strengths
Resources and Summary
Resources

- http://java.sun.com/j2ee
- http://java.sun.com/j2ee/blueprints
- http://java.sun.com/j2ee/tutorial
- http://java.sun.com/products/servlet
Additional Resources

& J2EE
  μ java.sun.com/j2ee

& Sun ONE
  μ www.sun.com/sunone

& Java Community Process (JCP)
  μ jcp.org

& Java and XML
  μ java.sun.com/xml

& Web Services
  μ www.webservices.org
Summary

This Section Discussed design and implementation of the VIEW. You should now understand:

• **View Design Issues**

• Guidelines for Architecting the "View" or Presentation logic

• Servlets for presentation requests

• JSP™ s for presentation layout
Call for Action!

- Download J2EE 1.3 Reference Implementation
- Start developing Servlets and JSPs today!
www.sun.com/developers/evangcentral

In pursuit of the best software in the universe

- All presentations
- Audiocasts
- Codecamp materials
- Technology briefings
- code/articles-links/chats/resources
Andrew Gilbert

andrew.gilbert@sun.com