A Reusable Servlet Framework

Presented to the St. Louis Java User's Group

September 14, 2000

written by Eric Burke
burke_e@ociweb.com

last revised 01 Sep 2000

Contents

Typical Servlet

- Is a subclass of HttpServlet
  - some web applications use numerous Servlets
  - others use a single Servlet for numerous pages
- Intercepts HTTP requests in the doGet or doPost method
  - parameters must be validated
- Displays HTML or other content
  - generally a bunch of println() statements to formulate the HTML
  - other (better) approaches are possible:
    - **formatting XML using XSLT style sheets**
    - HTML class libraries for creating pages and forms using Java objects
      - Apache's ECS, for example
    - JSPs
    - Template engines

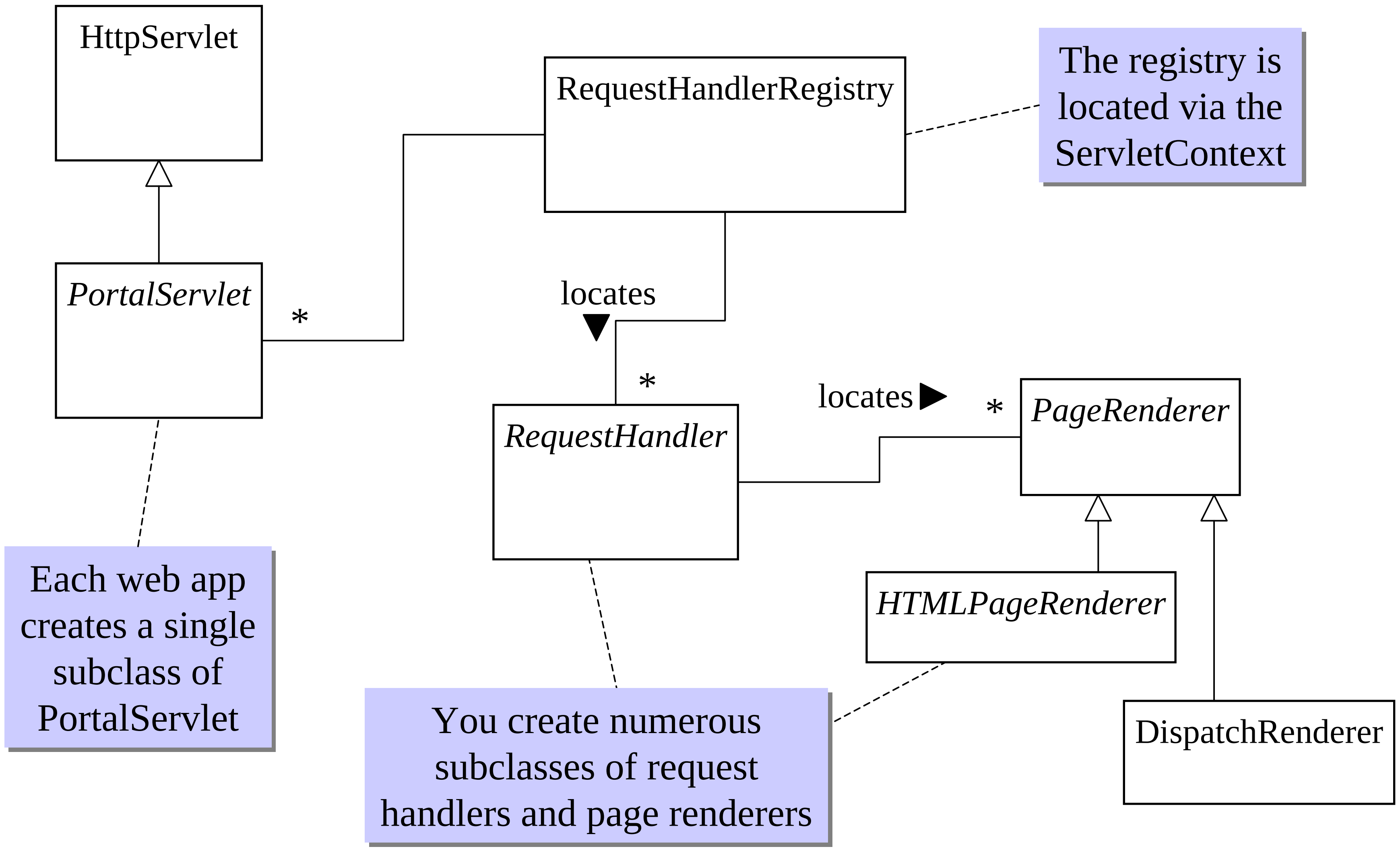
Issues

- HTTP is not amenable to sophisticated, multi-screen applications
  - instructions to a web application must be encoded as Strings
    - request parameters, extra path info, form elements
    - a simple typo could cause runtime errors - few compiler checks
    - **unless programmers are meticulous**, leads to very inconsistent code with lots of String comparisons
  - moving from page-to-page is not defined by the Servlet spec
    - each programmer does it differently
- Most Servlets handle too much
  - request parsing, page rendering, workflow
- XML and other approaches only address page rendering
  - **removes the need to embed ugly println() statements in the Servlet**

Framework Goals

- Concise and Simple
  - both for the framework itself and for applications that utilize the framework
  - consistent, well structured programming model for web apps
- Clear separation of responsibilities
  - parsing requests and selecting the next page
  - rendering "content"
    - may be HTML, XML, or even binary data
- Scalable
  - support apps with hundreds of pages
- Focused
  - the framework handles page-to-page interactions (workflow)
  - do not impose policies for page rendering, security, etc...
    - these problems have already been solved with other technologies

Class Diagram


HttpServlet
RequestHandlerRegistry
The registry is located via the ServletContext
PortalServlet
*
locates ▼
*
RequestHandler
locates ▶
*
PageRenderer
HTMLPageRenderer
DispatchRenderer
Each web app creates a single subclass of PortalServlet
You create numerous subclasses of request handlers and page renderers

Data Dictionary

- PortalServlet
  - abstract base class for Servlets
  - each web app creates a single subclass
  - `getDefaultRequestHandler` and `getRequestHandlers` are abstract methods
- RequestHandlerRegistry
  - responsible for locating `RequestHandler` subclasses
  - `PortalServlet` uses init/destroy methods to register/unregister
- RequestHandler
  - abstract base class for application specific request handlers
  - each subclass must provide a unique pathInfo identifier
  - responsible for selecting the next `PageRenderer` to display

Data Dictionary (Cont'd)

- PageRenderer
  - abstract base class for application-specific renderers
  - has a single method:

    ```
    public abstract void render(HttpServletRequest req,
                HttpServletResponse res);
    ```

  - subclasses can write any content to the response
- HTMLPageRenderer
  - convenience base class for rendering HTML
  - subclasses override the abstract method:

    ```
    public abstract String getHTML(HttpServletRequest req,
         HttpServletResponse res);
    ```

- DispatchRenderer
  - uses RequestDispatcher to forward to a JSP, static file, or Servlet

Typical Scenario

- The user types a URL into Netscape and hits Enter
  - the URL references a subclass of `PortalServlet`
- The doGet(...) method of `PortalServlet` is invoked
  - ask the `ServletContext` for the `RequestHandlerRegistry` instance
  - ask `RequestHandlerRegistry` for an appropriate `RequestHandler`
    - the default request handler is returned, since the URL did not contain any extra path information
  - invoke requestHandler.handleRequest(...)
    - this method will get a subclass of `PageRenderer`
    - it will then write HTML back out to the browser
      - any other content type is possible

Steps to Use the Framework

1. Create a subclass of PageRenderer for each page
   - implement the **render(...)** method
     - or create a subclass of HTMLPageRenderer
     - no subclass is necessary for DispatchRenderer
2. Create a subclass of RequestHandler for each page
   - you can also use a single request handler for multiple pages
   - request handlers must be stateless and thread-safe
3. Create a subclass of PortalServlet
   - override the two abstract methods
   - provide references to the request handlers
4. Package your app in a WAR file and deploy to the server
   - you can include smj_portal.jar in the WEB-INF/lib directory

PortalServlet.doService

Gets the registry from the ServletContext

```
protected void doService(HttpServletRequest req, HttpServletResponse res)
        throws IOException, ServletException {
    RequestHandlerRegistry rhReg = getRegistry();

    // if the requestHandlerID is null, the registry will return its
    // default handler
    RequestHandler rh = rhReg.getRequestHandler(this, req);
    rh.handleRequest(req, res);
}
```

The request handler will delegate rendering to a PageRenderer

RequestHandler.java

```
public abstract class RequestHandler {
   private String pathInfo;

   public RequestHandler(String pathInfo) {
       this.pathInfo = pathInfo;
   }

   public final void handleRequest(HttpServletRequest req,
           HttpServletResponse res) throws IOException, ServletException {
       PageRenderer pr = doHandleRequest(req);
       pr.render(req, res);
   }

   protected abstract PageRenderer doHandleRequest(HttpServletRequest req);

   public String getPathInfo() {
       return pathInfo;
   }
}
```

subclasses must provide a unique path, such as "logon"

delegate to the renderer

The only abstract method. Subclasses must validate incoming parameters and select the appropriate renderer.

Renderers

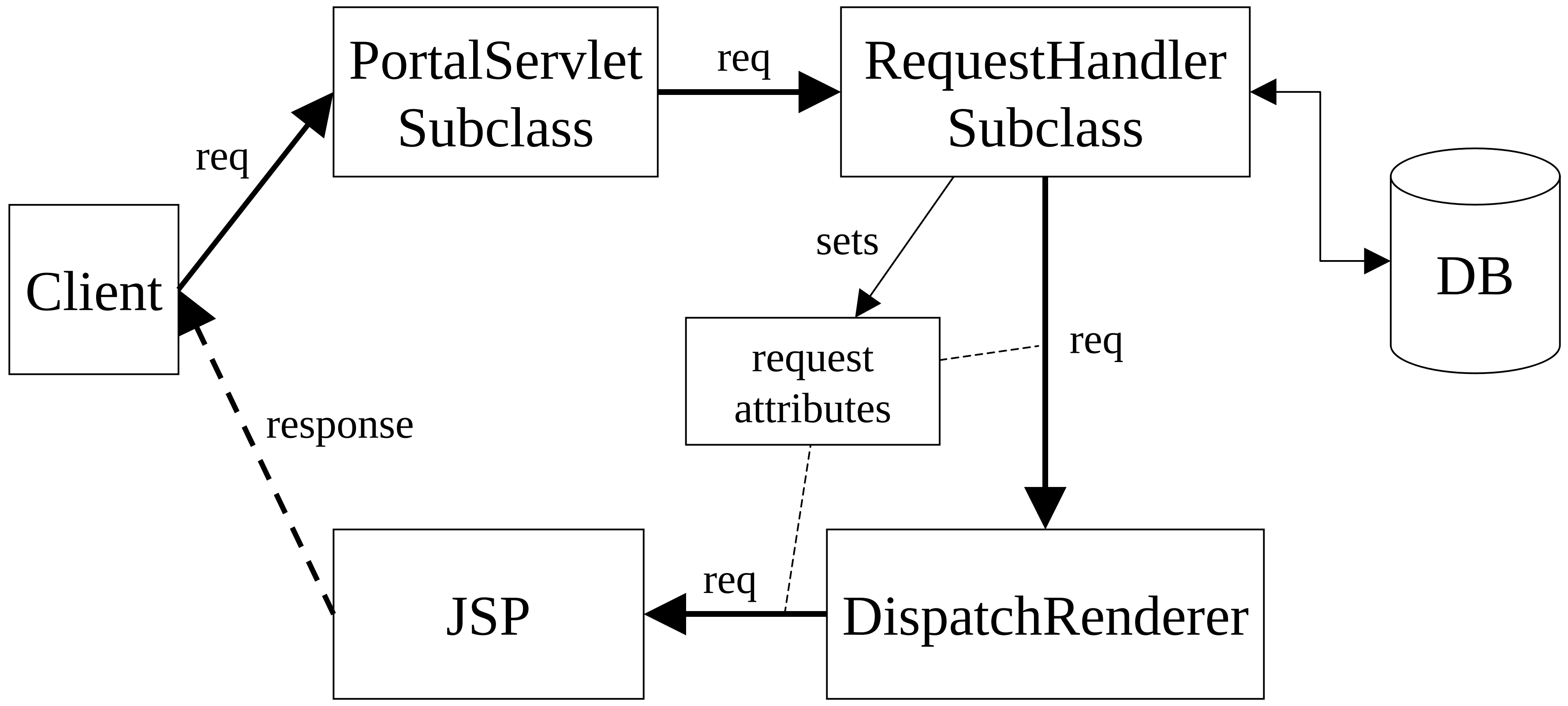
```
// abstract base class for all renderers
public abstract class PageRenderer {
  // the derived class can set the content type of the response
  // to anything, such as image/gif or text/html
  public abstract void render(HttpServletRequest req,
      HttpServletResponse res) throws IOException, ServletException;
}

// base class for HTML web pages
public abstract class HTMLPageRenderer extends PageRenderer {
  public void render(...) {
     ...
  }
  public abstract String getHTML(HttpServletRequest req,
      HttpServletResponse res) throws IOException, ServletException;
}
```

DispatchRenderer - Good for JSPs

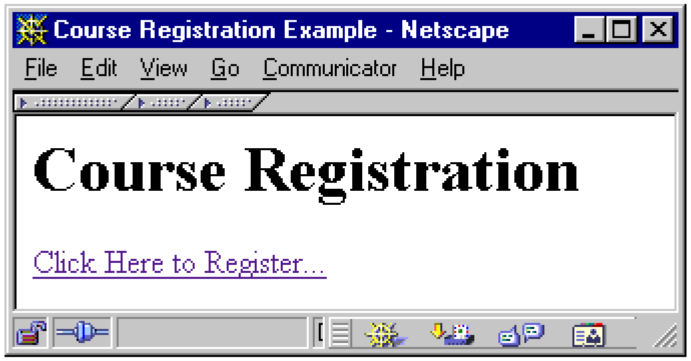
```
public class DispatchRenderer extends PageRenderer {
  private String path;

  /**
   * @param path the resource to forward (dispatch) to.
   */
  public DispatchRenderer(String path) {
      this.path = path;
  }

  public void render(HttpServletRequest req,
      HttpServletResponse res) throws IOException, ServletException {
      RequestDispatcher rd = req.getRequestDispatcher(path);
      rd.forward(req, res);
  }
}
```

JSP Flow

If the request handler needs to pass data to the JSP, it can add attributes to the request



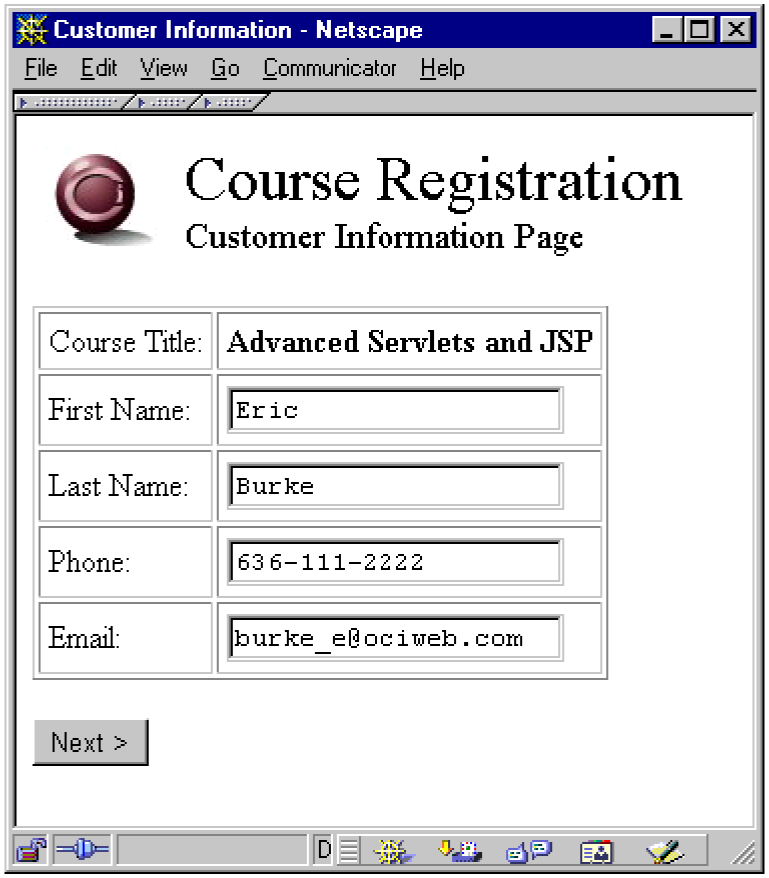
A Sample Application


Course Registration Example - Netscape
File Edit View Go Communicator Help

Course Registration

Click Here to Register...

- RegistrationServlet is the subclass of PortalServlet
  - DefaultReqHandler displays index.html, shown above
  - user clicks on link to begin
    - CustInfoReqHandler is invoked
      - extra path info contains "custInfo"
    - custInfo.jsp is returned to the browser
      - shown to the right


Customer Information - Netscape
File Edit View Go Communicator Help

Course Registration
Customer Information Page

Course Title:	Advanced Servlets and JSP
First Name:	Eric
Last Name:	Burke
Phone:	636-111-2222
Email:	burke_e@ociweb.com

RegistrationServlet.java

```
package com.ociweb.training;
import com.showmejava.portal.*;

public class RegistrationServlet extends PortalServlet {
    protected RequestHandler[] getRequestHandlers() {
        return new RequestHandler[] {
            new CustInfoReqHandler(),
            new SubmitReqHandler()
        };
    }

    protected RequestHandler getDefaultRequestHandler() {
        return new DefaultReqHandler();
    }
}
```

DefaultReqHandler.java

```
package com.ociweb.training;

import com.showmejava.portal.*;
import javax.servlet.http.*;

public class DefaultReqHandler extends RequestHandler {
    public DefaultReqHandler() {
        super("");  // not used for the default handler
    }

    protected PageRenderer doHandleRequest(
            HttpServletRequest req) {
        return new DispatchRenderer("/index.html");
    }
}
```

CustInfoReqHandler.java

```
public class CustInfoReqHandler extends RequestHandler {
    public static String PATH_INFO() {
        return "custInfo";
    }

    public CustInfoReqHandler() {
        super(PATH_INFO());
    }

    // a recommended convention is to define the URLs in the
    // request handlers using static methods
    public static String getURL(HttpServletRequest request,
            HttpServletResponse response) {
        // the context path is the name of the war file minus ".war"
        return response.encodeURL(request.getContextPath()
                + "/register/" + PATH_INFO());
    }
```

Using a static method is a recommended convention

CustInfoReqHandler.java (Cont'd)

```
// this method is a required part of the framework
protected PageRenderer doHandleRequest(HttpServletRequest req) {
    // see if the user hit the Next button
    if (req.getParameter("nextBtn") != null) {
        String firstName = req.getParameter("firstName");
        String lastName = req.getParameter("lastName");
        String phone = req.getParameter("phone");
        String email = req.getParameter("email");

        // create a registration object, which holds data and
        // does simple validation
        Registration reg = new Registration(firstName, lastName,
                phone, email);
```

CustInfoReqHandler.java (Cont'd)

```
                // store the registration info.  This makes it accessible
                // from the JSPs
                HttpSession session = req.getSession(true);
                session.setAttribute("registration", reg);
                if (reg.isValid()) {
                    // only move to the next page if the data is valid
                    return new DispatchRenderer("/confirm.jsp");
                }
            }

            // re-display the custInfo page if the user did not click
            // the next button, or if the data was invalid
            return new DispatchRenderer("/custInfo.jsp");
        }
    }
```

Request Handler Summary

- Tasks
  - extract data from HTML form elements
    - String firstName = req.getParameter("firstName");
  - validate that data
  - return the next renderer to display
    - on the previous example, custInfo.jsp was re-displayed in the event of an error. Otherwise, confirm.jsp was displayed.
- Conventions
  - provide static methods to formulate a URL to this handler
    - use HttpServletResponse.encodeURL to rewrite the URL for sessions
  - provide a static method called PATH_INFO() to return the unique path information for this handler

custInfo.jsp

```
<%@ page import="com.ociweb.training.*,com.showmejava.portal.*" %>

<html>
<head><title>Customer Information</title></head>
<body>

<jsp:include page="header.jsp" flush="true">
  <jsp:param name="heading1" value="Course Registration"/>
  <jsp:param name="heading2" value="Customer Information Page"/>
</jsp:include>
```

custinfo.jsp (Cont'd)

```
<% // this object is used to pre-fill text fields on the form
  Registration reg = (Registration)
          session.getAttribute("registration");

  String firstName = (reg != null) ? reg.getFirstName() : "";
  String lastName = (reg != null) ? reg.getLastName() : "";
  String phone = (reg != null) ? reg.getPhone() : "";
  String email = (reg != null) ? reg.getEmail() : "";

  if (reg != null && !reg.isValid()) {
      out.println("<h3>ERROR: All fields are required!</h3>");
  }
%>

<form method=POST action="<%= CustInfoReqHandler.getURL(request,
   response) %>">
```

custinfo.jsp (Cont'd)

```
<table cellpadding=4 border=1>
  <tr><td>Course Title:</td><td><b>Advanced Servlets and
   JSP</b></td></tr>
  <tr><td>First Name:</td><td><input type=text name=firstName
          value="<%= HTMLUtil.escape(firstName) %>"></td></tr>
  <tr><td>Last Name:</td><td><input type=text name=lastName
          value="<%= HTMLUtil.escape(lastName) %>"></td></tr>
  <tr><td>Phone:</td><td><input type=text name=phone
          value="<%= HTMLUtil.escape(phone) %>"></td></tr>
  <tr><td>Email:</td><td><input type=text name=email
          value="<%= HTMLUtil.escape(email) %>"></td></tr>
</table>
<br>
<input type=submit name=nextBtn value="Next &gt;">
</form>
</body></html>
```

JSP Summary

- The JSPs are only responsible for displaying content
  - try to keep Java code to a minimum
  - request handlers can pass data to JSPs in a couple of ways
    - by storing it in the HttpSession
    - by adding an attribute to the HttpServletRequest
- Factor out common code
  - the page header was repeated in every page
  - it is now included using <jsp:include .../> tag

The Deployment Descriptor

```
<?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>rs</servlet-name>
    <servlet-class>com.ociweb.training.RegistrationServlet</servlet-
   class>
  </servlet>

  <servlet-mapping>
    <servlet-name>rs</servlet-name>
    <url-pattern>/register/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Deployment (using Tomcat)

- This framework has been packaged into a JAR file
  - smj_portal.jar
- Example code has been packaged into a WAR file
  - smj_portal.jar is also included in this WAR file, under the WEB-INF/lib directory
  - simply copy portaltest.war to Tomcat's webapps directory
- To test:
  - http://localhost:8080/portaltest
    - this will cause index.html to be displayed
    - index.html then links to:
      - http://localhost:8080/portaltest/register/custInfo

the web application context

Extra path info that locates the request handler

alias for RegistrationServlet

Recommended Reading and Web Sites

- Java Servlet Programming
  - Jason Hunter (O'Reilly)
  - A little out of date; does not cover web applications or newer Servlet features
- Web Development with JavaServer Pages
  - Duane K. Fields and Mark A. Kolb (Manning Publications)
  - Covers latest specs, has discussion of similar architectures
- Web sites
  - jakarta.apache.org - the home of Tomcat
  - www.showmejava.com - download the Servlet framework
  - java.sun.com/products/servlet/ - Sun's Servlet pages

Advanced Servlet and JSP Programming

- This material is a small subset of OCI's new course: "Advanced Servlet and JSP Programming"
  - 3 days of advanced Servlet and JSP material
    - our 2 day "Java Servlet Programming" course is a prerequisite
  - contact training@ociweb.com for more information
    - or call Dr. Peter Maher, Director of Training, at 314-579-0066
- Topics include
  - Tomcat configuration, web applications, and WAR files
  - app to Servlet communication (mostly HTTP, with comparisons to other techniques)
  - alternate media types (PDF, Images, XSL-fo, etc...)
  - comprehensive coverage of JSP, including custom tag libraries
  - Servlet framework design, application design techniques
  - example application walkthru (shopping cart)
  - misc topics, covering recent developments in Servlets