# Lecture 9: The Servlet/JSP Version of the Demo Information Processing System

(04-15-2020)

(Reading: Lecture Notes)

Lecture Outline

## 1. Introduction

(1) The advantages and problems of CGI scripts based information processing

  a. Main advantages

   (a) Flexible CGI scripts: each script can be easily modified and updated without affecting other parts.

   (b) The backend programs can be written in many different types of languages efficiently.

  b. Disadvantages

   (a) Complex CGI scripts.

   (b) Non-trivial interactions between CGI scripts and the backend programs.

(2) Advantages of Java Servlets

  a. Unified interface and backend program.

  b. Portability of Java makes it more appealing.

(3) Levels of networking utilities

  a. Basic networking APIs: BSD socket API. In Java, the corresponding API is provided by the classes and interfaces of the **java.net** packages.

  b. Higher level networking facilities: RPC and RMI

  c. Servlets: special high-level facilities that extend server capabilities in client-server applications.

(4) Servlets

a. The *client-server model* of applications
b. The *request-response* model of communications of the client-server model is the foundation of servlets.
c. The web client and server applications: a typical client-server model applications using the request-response model of communications.
d. Servlet technique extends server applications by providing application specific functionalities.
    (a) Example: for web applications, Java Servlets provide facility to interact with web servers and client browsers to facilitate their interactions.
e. Servlet container (servlet engine): the server that executes servlets.

2. Java Servlet, JSP, and JDBC

(1) Java servlets

a. Java Servlets is an implementation of of the generic servlet concepts.
b. Java Servlets support both general servlet applications and web based (i.e. HTTP) applications.

(2) JavaServer Pages (JSP)

a. JSP is an extension of Java Servlets technology
    (a) JSP provides more flexible and powerful *markup* capabilities (than Java Servlets).
    (b) JSP normally is more suitable for presenting static markups. On the other hand, Java Servlets provide more powerful facilities for information processing and database accessing/updating.
b. Java Servlets and JSP have many overlapping functionalities. The main difference lies on the degree of easiness of use.

(3) JDBC (Java database connectivity)

a. JDBC is Java's API for communicating and manipulating data in relational databases.
b. JDBC supports SQL compatible statements for database access and manipulation.
c. Java supports a number of JDBC drivers that implement interfaces to some popular databases (such as Oracle and Microsoft Access). Third-party drivers for a large number of databases are available.

3. Java Servlet architecture (Chapter 9, Deitel)

(1) Interface **Servlet**

a. Each servlet must implement the interface **Servlet**.

b. Methods of **Servlet** interface (Fig.9.1,p.535)

(a) **void init (ServletConfig config)**
Automatically called once during a servlet's execution cycle to initialize the servlet. The only argument is supplied by the servlet container.

(b) **ServletConfig getServletConfig()**
Returns a reference to an object that implements interface **ServletConfig**. This object provides access to the servlet's configuration (such as servlet initialization parameters and the servlet's **ServletContext**, which provides the servlet with access to its environment).

(c) **String getServletinfo()**
Defined by a servlet programmer to return a **String** containing servlet info such as the servlet's author, version, or any other annotation.

(d) **void service (ServletRequest request, ServletResponse response)**
The servlet container calls this method to respond to a client request to the servlet.

(e) **void destroy ()**
Called by the container to *cleanup* the terminating servlet.

(2) **GenericServlet** and **HttpServlet** classes.

a. The servlet packages define two **abstract** classes – **GenericServlet** and **HttpServlet** – that implement the interface **Servlet**.

∗ These two classes provide all default implementations of all the **Servlet** methods.

∗ Most servlets extend one of these two classes and override some or all of their methods.

b. The interface **HttpServlet** defines enhanced processing capabilities for servlets that extend the functionality of an HTTP server.

c. For **HttpServlet**, the most important method is the **service**, which allows a servlet to receive both a **ServletRequest** object and a **ServletResponse** object.

(a) These two objects provide access to input and output streams that allow the servlet to read data from the client and send data to the client.

(b) The streams can be either byte based or char based.

(c) **ServletException** and **IOException** can be thrown to handle errors of a servlet execution.

(3) The **HttpServlet** class

a. This class is extended by web-based servlets. This class overrides method **service** to distinguish between the typical requests received from a client web browser.

b. **doGet** and **doPost** methods in **HttpServlet** class: correspond to the **get** and **post** methods in HTTP protocol.

   (a) **protected void doGet(HttpServletRequest req,**
   **HttpServletResponse resp) throws ServletException,**
   **IOException**
   Called by the server (via the **service** method) to allow a servlet to handle a GET request.

   (b) **protected void doPost(HttpServletRequest req,**
   **HttpServletResponse resp) throws ServletException,**
   **IOException**
   Called by the server (via the **service** method) to allow a servlet to handle a POST request. (Recall: The HTTP POST method allows the client to send data of unlimited length to the Web server a single time and is useful when posting information such as credit card numbers.).

c. Other methods in **HttpServlet** (there are a total ten methods): Fig.9.2, p.537

   (a) **doDelete**: called in response to an HTTP **delete** request to delete a file from an HTTP server. Some HTTP servers may not support the **delete** request due to its potential security problem.

   (b) **doOptions**: called in response to an HTTP **options** request to obtain option information supported bya an HTTP server. Example options include the supported methods, and version number.

   (c) **doPut**: called in response to an HTTP **put** request to store a file on an HTTP server. Again, **put** request may not be supported by some HTTP servers due to security concern.

   (d) **doTrace**: called in response to an HTTP **trace** request to provide debugging information. An implementation of this request will automatically return (to the requesting client) an HTML document that contains the request header info.

(4) The **HttpServletRequest** interface

a. Every call to **doGet** (or **doPost**) returns an object that implements interface **HttpServletRequest**.

   (a) The HTTP server that executes the servlet (hence a **doGet** or **doPost**) creates an **HttpServletRequest** object and passes this object to the servlet's **service** method, which in turn passes it to **doGet** or **doPost**.

   (b) This object contains the request from the client (in CGI terms, this object contains the CGI string (name=value pairs), among others).

(c) The **HttpServletRequest** interface implements a number of methods to facilitate the servlet processing the request. Some of these methods are from the interface **ServletRequest**, which the interface **HttpServletRequest** extends.

(d) A complete list of **HttpServletRequest** methods can be found from Java's document:

http://java.sun.com/j2ee/j2sdkee/techdocs/api/javax/servlet/http/HttpServletRequest.html

b. Some important **HttpServletRequest** methods (Fig. 9.3, p.537,538)

(a) **String getParameter (String name)**: obtains the value of a parameter (name) sent to the servlet as part of a **get** or **post** request. I.e. it returns the **value** of the **name** in a **name=value** pair.

(b) **Enumeration getParameterNames ()**: returns the names of all the parameters sent to the servlet as part of a **post** request.

(c) **String[] getParameterValues (String name)**: For a parameter with multiple values, this method returns an array of **String**s consisting of all values of the given name.

(d) **Cookie[] getCookies ()**: returns an array of **Cookie** objects stored on the client by the server. Cookies are used to identify clients.

(e) **HttpSession getSession (boolean create)**: returns an **HttpSession** object associated with the client's current browsing session.

(5) The **HttpServletResponse** interface

a. Every call to **doGet** (or **doPost**) returns an object that implements interface **HttpServletReponse**.

(a) The HTTP server that executes the servlet (hence a **doGet** or **doPost**) also creates an **HttpServletResponse** object and passes this object to the servlet's **service** method, which in turn passes it to **doGet** or **doPost**.

(b) The **HttpServletResponse** interface implements a number of methods to facilitate the servlet to formulate response to the client. Some of these methods are from the interface **ServletResponse**, which the interface **HttpServletResponse** extends.

b. Some important **HttpServletResponse** methods (Fig. 9.4, p.538,539)

(a) **void addCookie (Cookie cookie)**: adds a **Cookie** to the header of the response to the client.

(b) **ServletOutputStream getOutputStream ()**: obtains a byte-based output stream for sending binary data to the client.

(c) **PrintWriter getWriter ()**: obtains a character-based output stream for sending text data to the client.

(d) **void setContentType (String type)**: specifies the MIME type of the response to the browser.

4. Handling servlet get and post requests

(1) Handling HTTP get request, Example 1: WelcomeServlet.java, Fig.9.5, p.540

a. Lines 5-7 import appropriate packages. The package **javax.servlet** contains the general servlet package; while the package **javax.servlet.http** provides superclass **HttpServlet** for servlets that can handel HTTP get request and other requests.

b. Each HTTP servlet application must **extends** the superclass **HttpServlet**: line 9.

c. Line 12-44 declares a **doGet** method that overrides the **doGet** method in the superclass.

d. Line 16 specifies the content type of the HTTP data to be sent, using the **setContentType** method of the **response** object, which is passed as the 2nd parameter.

e. Line 17 obtains a reference to the **PrintWriter** object using the **getWriter** method of the **response** object. This reference enables the servlet to send (HTTP) content to the client.

f. Lines 22-44 create and send the body of HTTP response.

g. The HTML file that works with the servlet: file WelcomeServlet.html, Fig.9.6, p.541

```
1  // Fig. 9.5: WelcomeServlet.java
2  // A simple servlet to process get requests.
3  package com.deitel.advjhtp1.servlets;
4
5  import javax.servlet.*;
6  import javax.servlet.http.*;
7  import java.io.*;
8
9  public class WelcomeServlet extends HttpServlet {
10
11    // process "get" requests from clients
12    protected void doGet( HttpServletRequest request,
13       HttpServletResponse response )
14          throws ServletException, IOException
15    {
      response.setContentType( "text/html" );
       PrintWriter out = response.getWriter();

       // send XHTML page to client

      // start XHTML document
      out.println( "<?xml version = \"1.0\"?>" );

      out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
         "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
         "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );

      out.println(
         "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );

      // head section of document
      out.println( "<head>" );
      out.println( "<title>A Simple Servlet Example</title>" );
      out.println( "</head>" );

      // body section of document
      out.println( "<body>" );
      out.println( "<h1>Welcome to Servlets!</h1>" );
      out.println( "</body>" );

      // end XHTML document
      out.println( "</html>" );
      out.close();  // close stream to complete the page
   }
}
```

```
<?xml version = "1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Fig. 9.6: WelcomeServlet.html -->

<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
   <title>Handling an HTTP Get Request</title>
</head>

<body>
```

```
   <form action = "/advjhtp1/welcome1" method = "get">

      <p><label>Click the button to invoke the servlet
         <input type = "submit" value = "Get HTML Document" />
      </label></p>

   </form>
</body>
</html>
```

(2) Example 2: Handling HTTP get requests containing data, WelcomeServlet2.java, Fig.9.11, p.549, 550

a. Line 16 obtains the *value* part of a *name=value* pair in an HTTP request string by using the **getParameter** method of the **response** object.

  (a) Note: for each **get** request, a web browser will construct a request string of the form

  URL-Prefix?*name1=value1&name2=value2...*

  The *name=value* portion is called *query string*. The char ? indicates the beginning of the query string.

b. The HTML file that works with the servlet: file WelcomeServlet2.html, Fig.9.12, p.551

```
1  // Fig. 9.11: WelcomeServlet2.java
2  // Processing HTTP get requests containing data.
3  package com.deitel.advjhtp1.servlets;
4
5  import javax.servlet.*;
6  import javax.servlet.http.*;
7  import java.io.*;
8
9  public class WelcomeServlet2 extends HttpServlet {
10
11    // process "get" request from client
12    protected void doGet( HttpServletRequest request,
13       HttpServletResponse response )
14          throws ServletException, IOException
15    {
16       String firstName = request.getParameter( "firstname" );
17
18       response.setContentType( "text/html" );
19       PrintWriter out = response.getWriter();
```

```java

      // send XHTML document to client

      // start XHTML document
      out.println( "<?xml version = \"1.0\"?>" );

      out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
         "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
         "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );

      out.println(
         "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );

      // head section of document
      out.println( "<head>" );
      out.println(
         "<title>Processing get requests with data</title>" );
      out.println( "</head>" );

      // body section of document
      out.println( "<body>" );
      out.println( "<h1>Hello " + firstName + ",<br />" );
      out.println( "Welcome to Servlets!</h1>" );
      out.println( "</body>" );

      // end XHTML document
      out.println( "</html>" );
      out.close();  // close stream to complete the page
   }
}
```

```html
<?xml version = "1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Fig. 9.12: WelcomeServlet2.html -->

<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
   <title>Processing get requests with data</title>
</head>

<body>
```

```
   <form action = "/advjhtp1/welcome2" method = "get">

      <p><label>
         Type your first name and press the Submit button
         <br /><input type = "text" name = "firstname" />
         <input type = "submit" value = "Submit" />
      </p></label>

   </form>
</body>
</html>
```

(3) Example 3: Handling HTTP post requests, WelcomeServlet3.java, Fig.9.14, p.553, 554

a. Line 12-48 declares a **doPost** method that overrides the **doPost** method in the superclass.

b. The HTML file that works with the servlet: file WelcomeServlet2.html, Fig.9.15, p.554

```
1  // Fig. 9.14: WelcomeServlet3.java
2  // Processing post requests containing data.
3  package com.deitel.advjhtp1.servlets;
4
5  import javax.servlet.*;
6  import javax.servlet.http.*;
7  import java.io.*;
8
9  public class WelcomeServlet3 extends HttpServlet {
10
11    // process "post" request from client
12    protected void doPost( HttpServletRequest request,
13       HttpServletResponse response )
14          throws ServletException, IOException
15    {
16       String firstName = request.getParameter( "firstname" );
17
18       response.setContentType( "text/html" );
19       PrintWriter out = response.getWriter();
20
21       // send XHTML page to client
22
23       // start XHTML document
24       out.println( "<?xml version = \"1.0\"?>" );
```

```

      out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
         "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
         "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );

      out.println(
         "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );

      // head section of document
      out.println( "<head>" );
      out.println(
         "<title>Processing post requests with data</title>" );
      out.println( "</head>" );

      // body section of document
      out.println( "<body>" );
      out.println( "<h1>Hello " + firstName + ",<br />" );
      out.println( "Welcome to Servlets!</h1>" );
      out.println( "</body>" );

      // end XHTML document
      out.println( "</html>" );
      out.close();  // close stream to complete the page
   }
}
```

```
<?xml version = "1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Fig. 9.15: WelcomeServlet3.html -->

<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
   <title>Handling an HTTP Post Request with Data</title>
</head>

<body>
   <form action = "/advjhtp1/welcome3" method = "post">

      <p><label>
         Type your first name and press the Submit button
         <br /><input type = "text" name = "firstname" />
```

```
            <input type = "submit" value = "Submit" />
         </label></p>

      </form>
</body>
</html>
```

5. Java JSP overview and architecture (Chapter 10, Deitel)

(1) What is JSP (JavaServer Page)?

a. Java JSP is an extension of the Java Servlet technology.
   (a) It simplies the delivery of dynamic web contents.
   (b) It enables creation of dynamic contents by reusing predefined components and by interacting components using server-side scripting.

b. The **javax.servlet.jsp** and **javax.servlet.jsp.tagext** packages are needed (in addtion to **javax.servlet** and **javax.servlet.http**) contains classes and interfaces specific to JSP programming.

c. JSP container (JSP engine): the server that executes JSP. JSPs are normally executed as part of a JSP container.

(2) Overview.

a. JSPs has four key components:
   (a) *directives*: they are messages to JSP container that allow a programmer to specify page settings, to include content from other sources, and to specify custom tag libraries for use in a JSP.
   (b) *actions*: they encapsulate functionality in predefined tags that programmers can embed in a JSP.
      i. As the case of *actions* in normal HTML page, actions are usually performed in response to a client request and are based on the information in a request.
      ii. Actions can also create Java objects for use in JSP scriptlets.
   (c) *scriptlets*: they are also called *scripting elements*. They enable programmers to insert Java code that interacts with components in a JSP (and possibly other Web application components) to perform request processing.
   (d) *tag libraries*: they are part of the so called *tag extension mechanism* that enables programmers to create custom tags, which enables programmers to manipulate JSP components.

b. Mark-up flavor of JSP files

    (a) JSP can directly include XHTML or XML documents (it can also include HTML documents). JSPs normally include XHTML or XML markups, which are called *fixed-template data* or *fixed-template-text*.
    (b) Fixed-tempplate-data allows programmers to decide whether to use servlet or JSP.
        i. When the majority of the reply content is fixed-template-data, JSP is preferred.
        ii. Otherwise servlets may be more appropriate.

  c. Processing JSP request – close interactions with servlets
    (a) When a JSP request is received by a JSP container, the container first translates the JSP into a Java servlet that handles the current and future requests to the JSP.
    (b) The JSP container places the Java servlet statements that implement the JSP's response in method **_jspService** at translation time.
    (c) The JSP may respond to request directly, or may invoke other processing components before responding.

(3) First JSP example: clock.jsp, Fig.10.1, p.596,597.

  a. The file consists of a large portion of XHTML markups.
    (a) There are only two lines that produce output: line 12, which is a normal XHTML tag pair, and line 31 (see below)
    (b) The rest of the document specify style, font, and other markup info.
  b. JSP expressions are delimited by <%= and %> pairs.
    (a) Line 31 is a JSP expression that creates a new instance of the class **Date** from the **java.util** package.
    (b) When a client requests this JSP, a string representation of the date and time returned by the **Date** class instance will be inserted as part of the HTTP response.
  c. Line 10 is an XHTML **meta** element that sets refresh interval of this document.
  d. This JSP can be accessed using the URL:

http://localhost:8080/advjhtp1/jsp/clock.jsp

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 10.1: clock.jsp -->
```

```

<html xmlns = "http://www.w3.org/1999/xhtml">

<head>
   <meta http-equiv = "refresh" content = "60" />

   <title>A Simple JSP Example</title>

   <style type = "text/css">
      .big { font-family: helvetica, arial, sans-serif;
             font-weight: bold;
             font-size: 2em; }
   </style>
</head>

<body>
   <p class = "big">Simple JSP Example</p>

   <table style = "border: 6px outset;">
      <tr>
         <td style = "background-color: black;">
            <p class = "big" style = "color: cyan;">

               <!-- JSP expression to insert date/time -->
               <%= new java.util.Date() %>

            </p>
         </td>
      </tr>
   </table>
</body>

</html>
```

(4) Implicit objects

a. Implicit objects provide many servlet capabilities to progammers in the JSP context. There are four scopes of JSP implicit objects: *application, page, request*, and *session*.

   (a) Application scope implicit objects: can be manipulated by any servlet or JSP.

   (b) Page scope implicit objects: they exist only in the page that defines them. Each page has its own instance of the page-scope implicit objects.

   (c) Request scope implicit objects: they exist in the duration of a request.

(d) Session scope implicit objects: they exist in the entire client session.

b. Fig.10.2: JSP implicit objects and their scope:

(a) *Application scope*

· bf application: This **javax.servlet.ServletContext** object represents the container in which the JSP executes.

(b) *Page scope*

· **config**: This **javax.servlet.ServletConfig** object represents the JSP configuration options. As with servlets, JSP configuration options can be specified in a Web application descriptor.

· **exception**: This **javax.servlet.Throwable** object represents the Java exception that is passed to the JSP error page. This object is available only in a JSP error page.

· **out**: This **javax.servlet.jsp.JspWriter** object writes text as part of the response to a request. This object is used only with JSP expressions and actions that insert string content in a response.

· **page**: This **javax.lang.Object** object represents the **this** reference for the current JSP instance.

· **pageContext**: This **javax.servlet.jsp.PageContext** object hides the implementation details of the underlying servlet and JSP container and provides JSP progammers with access to the implicit objects discussed in this table.

· **response**: This object represents the response to the client. This object normally is an instance of a class that implements **HttpServletResponse** (package **javax.servlet.http**). If a protocol other than HTTP is used, this object is an instance of a class that implements **javax.servlet.ServletResponse**.

(c) *Request scope*

· bf request: This object represents the client request. The object normally is an instance of a class that implements **HttpServletRequest** (package **javax.servlet.http**). If a protocol other than HTTP is used, this object is an instance of a class that implements **javax.servlet.ServletRequest**.

(d) *Session scope*

· bf session: This **javax.servlet.http.HttpSession** object represents the client session info if such a session has been created. This object is available only in pages that participate in a session.

(5) Scripting

a. JSP scripting is part of JSP that performs the operations to produce dynamic components of a response. In other words, it does the info processing part.

(a) JSP scripting components include *scriplets, comments, expressions, declarations*, and *escape sequences*.

(b) JSP scriplets are block of Java code delimited by <% and %> pairs.

(c) JSP supports three style of comments:

  i. Comments can be delimited by <%−− and −−%> pairs. Such comments can be placed anywhere in a JSP, except within scriplets.

  ii. XHTML style comments delimited by <!−− and −−> can also be placed anywhere in a JSP, except within scriplets either.

  iii. Comments can also be Java's style (delimited by //) or multiline style (delimited by /∗ and ∗/ pairs. As they are Java style of comments, these two style of comments can only be placed inside scriplets.

(d) JSP expressions are of Java expressions delimited by <%= and %> pairs.

  i. Such a pair enclose a Java expression that is evaluated whena client requests the JSP containing the expression.

  ii. The JSP container converts the result of a JSP expression to a **String** object first, then outputs the **String** object as part of the response to the client.

(e) JSP declarations are delimited by <%! and %> pairs. They allows programmers to define variables and methods. Such as

<%! int counter = 0; %>

(f) JSP escape sequences allows inclusion of special letters (those used as delimiters) in JSP documents: Cf. Fig.10.3, p.601).

b. Scripting example: Fig.10.4, p.601,602, welcome.jsp

(a) The example consists of a single JSP file.

(b) At line 19, the JSP obtains the value of the "firstName" parameter. If it does obtain a non-NULL value (line 21), it will display the value at lines 25-28.

(c) If it does not obtain a non-NULL value (line 33), the JSP asks input of first name at lines 38-43, and then invokes itself as the action. Therefore this JSP is recursive!

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 10.4: welcome.jsp -->
6  <!-- JSP that processes a "get" request containing data. -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
```

```

<!-- head section of document -->
<head>
   <title>Processing "get" requests with data</title>
</head>

<!-- body section of document -->
<body>
   <% // begin scriptlet

      String name = request.getParameter( "firstName" );

      if ( name != null ) {

   %> <%-- end scriptlet to insert fixed template data --%>

         <h1>
            Hello <%= name %>, <br />
            Welcome to JavaServer Pages!
         </h1>

   <% // continue scriptlet

      }  // end if
      else {

   %> <%-- end scriptlet to insert fixed template data --%>

         <form action = "welcome.jsp" method = "get">
            <p> Type your first name and press Submit</p>

            <p><input type = "text" name = "firstName" />
               <input type = "submit" value = "Submit" />
            </p>
         </form>

   <% // continue scriptlet

      }  // end else

   %> <%-- end scriptlet --%>
</body>

```

```
52 </html>  <!-- end XHTML document -->
```

(6) Standard actions: Fig.10.5, p.604.

a. There are seven standard actions:

   (a) **jsp:include**: Dynamically includes another resource in a JSP. As the JSP executes, the referenced resource is included and processed.

   (b) **jsp:forward**: Dynamically forwards request to another JSP, servlet, or static page. This action terminates the current JSP's execution.

   (c) **jsp:plugin**: Allows a plug-in component to be added to a page in the form of a browser specific object or embed HTML element. In the case of a Java applet, this action enables the downloading and installation of the Java Plug-in (if it is not already installed on the client computer). request to another JSP, servlet, or static page. This action terminates the current

   (d) **jsp:param**: Used with the **include, forward, plug-in** actions to specify additional name/value pairs of the info for use by these actions.
   Note: This action specifies *additional* name=value pair. This pair is *in addition* to the original name=value pairs in the the original request string of the initiating JSP!

   There are three other JavaBean manapulation actions.

b. <**jsp:include**> action

   (a) There are two *include* mechanisms: the <**jsp:include**> action and the **include** *directive*.

      i. The <**jsp:include**> action enables dynamically inclusion of resource in a JSP.

      ii. The **include** directive copies the specified content once at translation time.

   (b) Attributes of <**jsp:include**> action: Fig.10.6, p.605

      **page** Specifies the relative URL path of the resource to include. The resource must be part of the same Web application.

      **flush** Specifies whether the buffer should be flushed after the **include** is performed. In JSP 1.1 this attribute is required to be **true**.

c. <**jsp:include**> action example: Fig.10.7, 10.8, 10.9, 10.10, p.606-609.

   (a) The example consists of four files: two (X)HTML files and two JSP files. The JSP file include.jsp uses the **include** action to include the other three files.

   (b) Fig.10.7, p.606, 607. This html file named banner.html is included across the top of the XHTML document created by Fig.10.10.

(c) Fig.10.8, p.607,608. This html file lists the table of contents of the supported web page. It is included as the left side of the XHTML document created by Fig.10.10.

(d) Fig.10.9, p.608. This JSP file named clock2.jsp is included as the main content in the XHTML document created by Fig.10.10.

(e) Fig.10.10, p.608, 609. This jsp file named include.jsp includes all the three previous files.

```
<!-- Fig. 10.7: banner.html                -->
<!-- banner to include in another document -->
<div style = "width: 580px">
   <p>
      Java(TM), C, C++, Visual Basic(R),
      Object Technology, and <br /> Internet and
      World Wide Web Programming Training&nbsp;<br />
      On-Site Seminars Delivered Worldwide
   </p>

   <p>
      <a href = "mailto:deitel@deitel.com">
         deitel@deitel.com</a><br />

      978.579.9911<br />
      490B Boston Post Road, Suite 200,
      Sudbury, MA 01776
   </p>
</div>
```

```
<!-- Fig. 10.8: toc.html                     -->
<!-- contents to include in another document -->

<p><a href = "http://www.deitel.com/books/index.html">
   Publications/BookStore
</a></p>

<p><a href = "http://www.deitel.com/whatsnew.html">
   What's New
</a></p>

<p><a href = "http://www.deitel.com/books/downloads.html">
   Downloads/Resources
</a></p>

```

```
<p><a href = "http://www.deitel.com/faq/index.html">
   FAQ (Frequently Asked Questions)
</a></p>

<p><a href = "http://www.deitel.com/intro.html">
   Who we are
</a></p>

<p><a href = "http://www.deitel.com/index.html">
   Home Page
</a></p>

<p>Send questions or comments about this site to
   <a href = "mailto:deitel@deitel.com">
      deitel@deitel.com
   </a><br />
   Copyright 1995-2002 by Deitel &amp; Associates, Inc.
   All Rights Reserved.
</p>
```

```
<!-- Fig. 10.9: clock2.jsp                       -->
<!-- date and time to include in another document -->

<table>
   <tr>
      <td style = "background-color: black;">
         <p class = "big" style = "color: cyan; font-size: 3em;
            font-weight: bold;">

            <%-- script to determine client local and --%>
            <%-- format date accordingly              --%>
            <%
               // get client locale
               java.util.Locale locale = request.getLocale();

               // get DateFormat for client's Locale
               java.text.DateFormat dateFormat =
                  java.text.DateFormat.getDateTimeInstance(
                     java.text.DateFormat.LONG,
                     java.text.DateFormat.LONG, locale );

            %>  <%-- end script --%>

            <%-- output date --%>
            <%= dateFormat.format( new java.util.Date() ) %>
         </p>
      </td>
   </tr>
</table>
```

```
<?xml version = "1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Fig. 10.7: include.jsp -->

<html xmlns = "http://www.w3.org/1999/xhtml">

  <head>
      <title>Using jsp:include</title>

      <style type = "text/css">
         body {
            font-family: tahoma, helvetica, arial, sans-serif;
         }

         table, tr, td {
            font-size: .9em;
            border: 3px groove;
            padding: 5px;
            background-color: #dddddd;
         }
      </style>
   </head>

   <body>
      <table>
         <tr>
            <td style = "width: 160px; text-align: center">
               <img src = "images/logotiny.png"
                  width = "140" height = "93"
                  alt = "Deitel & Associates, Inc. Logo" />
            </td>

            <td>

               <%-- include banner.html in this JSP --%>
               <jsp:include page = "banner.html"
```

```
39                    flush = "true" />
40
41              </td>
42           </tr>
43
44           <tr>
45              <td style = "width: 160px">
46
47                 <%-- include toc.html in this JSP --%>
48                 <jsp:include page = "toc.html" flush = "true" />
49
50              </td>
51
52              <td style = "vertical-align: top">
53
54                 <%-- include clock2.jsp in this JSP --%>
55                 <jsp:include page = "clock2.jsp"
56                    flush = "true" />
57
58              </td>
59           </tr>
60        </table>
61     </body>
62  </html>
```

d. <**jsp:forward**> action example: Fig.10.11, 10.12, p.611-613.

(a) The example consists of two JSP files: the JSP file **forward1.jsp** is a modified version of **welcome.jsp** (Fig.10.4). This JSP will now forward the request to the 2nd JSP **forward2.jsp**.

(b) The JSP forward1.jsp now tests if the firstname parameter has a non-NULL value and if so it forwards the request to forward2.jsp at lines 22-25.

(c) Pay attention to the **¡jsp:param¿** action at lines 23-24. It passes a name/value pair to forward2.jsp.

   i. Every **¡jsp:param¿** action requires both a **name** attribute and a corresponding **value** attribute.

   ii. If the name specified in a ¡jsp:param¿ action conflicts with the name of in the request, the new value for that name parameter will take precedence over the original value.

(d) The JSP forward2.jsp is ordinary. At line 23, it obtains the value of "firstname" and value of "date" parameters at lines 23 and 31, respectively.

```
1  <?xml version = "1.0"?>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Fig. 10.11: forward1.jsp -->

<html xmlns = "http://www.w3.org/1999/xhtml">

<head>
   <title>Forward request to another JSP</title>
</head>

<body>
   <% // begin scriptlet

      String name = request.getParameter( "firstName" );

      if ( name != null ) {

   %> <%-- end scriptlet to insert fixed template data --%>

         <jsp:forward page = "forward2.jsp">
            <jsp:param name = "date"
               value = "<%= new java.util.Date() %>" />
         </jsp:forward>

   <% // continue scriptlet

      }  // end if
      else {

   %> <%-- end scriptlet to insert fixed template data --%>

         <form action = "forward1.jsp" method = "get">
            <p>Type your first name and press Submit</p>

            <p><input type = "text" name = "firstName" />
               <input type = "submit" value = "Submit" />
            </p>
         </form>

   <%  // continue scriptlet

      }  // end else

   %> <%-- end scriptlet --%>
</body>

</html>  <!-- end XHTML document -->
```

```
<?xml version = "1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- forward2.jsp -->

<html xmlns = "http://www.w3.org/1999/xhtml">

<head>
   <title>Processing a forwarded request</title>

   <style type = "text/css">
      .big {
         font-family: tahoma, helvetica, arial, sans-serif;
         font-weight: bold;
         font-size: 2em;
      }
   </style>
</head>

<body>
   <p class = "big">
      Hello <%= request.getParameter( "firstName" ) %>, <br />
      Your request was received <br /> and forwarded at
   </p>

   <table style = "border: 6px outset;">
      <tr>
         <td style = "background-color: black;">
            <p class = "big" style = "color: cyan;">
               <%= request.getParameter( "date" ) %>
            </p>
         </td>
      </tr>
   </table>
</body>

</html>
```

6. Apache Tomcat Java Servlet container

   (1) Getting the software

   (2) Structure of the container

      a. The configuration files
      b. The application directory: *webapps*
         (a) Location of HTML files
         (b) Location of Java Servlet classes
         (c) Location of JSP scripts

7. Servlet version of the demo on CentOS Linux

8. Servlet version of the demo on Windows