Chapter 17
Web-Based User Interface
Design and Programming

© Pearson Education Limited 2005

Topics

- Terminology and technologies
- Enabling technologies for web client tier
  - HTML
  - scripting languages
  - applets
- Enabling technologies for web server tier
  - servlets
  - beans
  - tags
  - JSP
  - Struts
- Additional references
**Concepts and technologies**

- **Web-based application** – an Internet browser manages the rendering of UI content, but the business logic and database state exists on a server.

- **HyperText Markup Language (HTML)**
  - A HTML-formatted web page is a mix of the presentation content (e.g. some text) and rendering instructions (e.g. font size).
  - HTML can be used to apply rendering instructions on a text in standard UI components, including Swing components.

- **JavaScript** – a scripting language used in HTML document to perform quick validation of inputs, simple animations, and other activities to make the web page “alive”.

- **Applet** – technology that makes the UI dynamic (in layout and content) within a browser.
  - JApplet component in Swing provides a UI container that can be placed in a web page.
  - HTML uses the `<object>` tag to download an object (e.g. an applet) that lives in the URL space to the client.
  - Applet lives in a sandbox environment – it has limited and restricted access to system resources.
  - Applet can be digitally signed – it can access system resources if an explicit permission is given to it → trusted applet.

**Concepts and technologies (cont.)**

- **Servlet (server applet)** – Java code, which Java-based web server (such as Tomcat) uses to handle requests (web server associates certain URLs with servlets rather than with physical web pages).
  - can create HTML pages (all the HTML content and formatting must come from a servlet).
  - can connect to a database and maintain its connection for more than one client.
  - can pass a client request to another servlet (servlet chaining).

- **Java Server Page (JSP)** – a HTML page with embedded Java code (tags and/or scriptlets) to manage the dynamic content of the page and to supply data to it.
  - with JSP, web designers do not have to be also Java developers.
  - JSP is dynamically compiled to a servlet prior to running – hence, JSP is really a servlet with HTML inside.

- **Scriplet** – a mix of Java code with web page content (not really reusable as it combines presentation and control code).

- **Tag** – placeholder for Java code designed to be reusable.
  - Java Standard Tag Library (JSTL).

- **Struts** – a framework for building web apps based on:
  - MVC architecture
  - set of JSP custom tags for building JSP pages.
JavaScript by example

- JavaScript can access the current web page via its objects
  - This is shown on Line 6 where the script checks the value of text field named percent (declared on Line 20 – next slide)
  - The value is returned as a string which needs to be converted into an integer via the call to a built-in function parseInt()
  - If the input value is valid (Lines 8 – 9), then the script returns true, otherwise it displays a message box indicating the wrong input and returns false

```javascript
<SCRIPT LANGUAGE="JavaScript">
  function checkIt(int lower, int upper) {
    var strval = document.sampleForm.percent.value;
    var intval = parseInt(strval);
    if ( lower < intval && intval < upper ) {
      return( true );
    } else {
      alert("Input " + strval + " is out of "+
           lower+"-"+upper+" range");
      return( false );
    }
  }
</SCRIPT>
```

Calling JavaScript from HTML

- The server URL, which the form will call, is placed in the action attribute
  - the form processor is handled by a servlet called SearchMovie, located on the root of the server (indicated by /)
  - the script is “fired” once the user clicks on the submit button, as indicated by the onSubmit event handler
  - The checkIt function (Line 5) checks the value of input text, presented by Line 20, to see if it has the correct value (as demanded by the lower and upper parameters of checkIt())

```html
<FORM NAME="sampleForm" METHOD="post"
       ACTION="/SearchMovie" onSubmit="checkIt(0,100)"
><P>Percentage given:
  <INPUT TYPE="text" NAME="percent" VALUE="1">
  <BR><INPUT TYPE="submit">
</FORM>
```
Applet in a sandbox

- **Applet**
  - program that runs inside a web browser
  - runs within a sandbox determined by Java run-time security system
  - built using an application framework (library) rooted at JApplet

- **Applet is not allowed:**
  - read from a file system of the host machine (including files, properties, etc)
  - write or delete a file
  - connect to a network port on any machine except the HTTP server it comes from
  - execute or load another programs/library/DLL code

Applet by example

- Applets are not required to have a `main( )` – you put any startup code in `init( )`

- The `init( )` method is responsible for putting all the components on the form using the `add( )` method
  - In this program, the only activity is putting a text label on the applet, via the JLabel class
  - Swing requires to add all components to the "content pane" of a form, and so you must call `getContentPane( )` as part of the `add( )` process

```java
public class Applet1 extends JApplet {
    public void init() {
        getContentPane().add(new JLabel("Applet!"));
    }
}
```

[Eckel]
Applet method summary

<table>
<thead>
<tr>
<th>Applet Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>void destroy()</td>
</tr>
<tr>
<td>Called by the browser or applet viewer to inform this applet that it is being reclaimed and that it should destroy any resources that it has allocated.</td>
</tr>
<tr>
<td>void init()</td>
</tr>
<tr>
<td>Called by the browser or applet viewer to inform this applet that it has been loaded into the system.</td>
</tr>
<tr>
<td>void start()</td>
</tr>
<tr>
<td>Called by the browser or applet viewer to inform this applet that it should start its execution.</td>
</tr>
<tr>
<td>void stop()</td>
</tr>
<tr>
<td>Called by the browser or applet viewer to inform this applet that it should stop its execution</td>
</tr>
</tbody>
</table>

Running applet inside browser

- Unfortunately not as simple as:

```
<applet code=Applet1 width=100 height=50>
</applet>
```

- Because of the browser and language wars, Sun could not expect browsers to support the correct flavor of Java, and the only solution was to provide some kind of add-on that would conform to a browser’s extension mechanism.

- With Internet Explorer, the extension mechanism is the ActiveX control, and with Netscape it is the plug-in.
  - In your HTML code, you must provide tags to support both, but you can automatically generate the necessary tags with the HTMLconverter tool that comes with the JDK download.

[Eckel]
Running applet from command line

- To create an applet that can be run from the console command line, you simply add a `main()` to your applet that builds an instance of the applet inside a `JFrame`
- `Applet1b.java` modified to work as both an application and an applet (the applet is created and added to a `JFrame` so that it can be displayed):

```
public class Applet1c extends JApplet {
    public void init() {
        getContentPane().add(new JLabel("Applet!"));
    }
    public static void main(String[] args) {
        JApplet applet = new Applet1c();
        JFrame frame = new JFrame("Applet1c");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().add(applet);
        frame.setSize(100,50);
        applet.init();
        applet.start();
        frame.setVisible(true);
    }
}
```

Servlet (Server Applet)

- The basic computational unit that a Java-based Web server uses to handle multiple clients (HTTP requests)
  - request is processed and data returned in a stateless transaction (normally in HTML output)
- There is no client code as such in the servlet. The client is a HTML page or other pages, such as JSP or ASP.
**Servlet by example**

- All page content and formatting must come from the servlet (lots of simple print statements)

```java
public class BaseballStatServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws java.io.IOException {
        resp.setContentType("text/html");
        java.io.PrintWriter html = resp.getWriter();
        String player = (String) req.getParameter("player");
        html.println("<HTML><HEAD><TITLE>Player Stats</TITLE></HEAD>");
        html.println("<BODY>");
        if ((player == null) || (player.length() == 0)) {
            html.println("<H1>No Player Requested</H1>");
        } else {
            if (player.equals("Derek Lowe")) {
                html.println("<H1>... </H1>");
            } else {
                html.println("<H1>... </H1>");
            }
        }
        html.println("</BODY></HTML>");
    }
}
```

[Turner and Bedell]

---

**Java Server Pages (JSP)**

- JSP separates the business logic from the presentation.
- The business logic is represented by a servlet whereas the presentation is the JSP.

1) send request to JSP
   - Servlet Engine will then
     - parse the JSP
     - generate servlet code
     - compile the servlet
     - instantiate the servlet

2) invoke the servlet

- Web server
- JSP request
- Web browser
- HTML output
- Servlet in a Servlet Container
- DB

© Pearson Education 2005  Chapter 17 (Maciaszek, Liong - PSE)
Beans

- **JSP** is able to use **JavaBeans** and **Enterprise JavaBeans (EJB)**
- **JavaBeans** are a set of Java classes that follow predefined rules to allow inspections and modifications on their properties.
- **EJB** is similar to JavaBeans but it supports the capability of accessing shared business logic and a shared database. EJB is fully managed by a server container (usually an application server).
- Beans conventions:
  - Each attribute of the bean that will be exposed publicly should have at least a method called `getX()`.
  - If the bean will allow the attribute to be modified, it needs to provide a method called `setX()`.
  - The `getX()` method should return the same type value that the `setX()` method takes as an argument.
  - If a value is Boolean, it uses the accessor `isX()` rather than `getX()`.
- JSP supports an introspection mechanism on beans that allows form values to be automatically populated into beans from a JSP page by using the `jsp:setProperty` tag.

JSP by example – input form

**Input an Animal**

- Common Name: Grey Wolf
- Species Name: Canis lupus
- Adult Height: 0.73
- Adult Weight: 66
- Top Speed: 66
- Description:
  
  Although most wolves have basically grey coats, hence the common name, the coats usual have a lot of bare yellow interspersed between the salt-and-pepper grey and black hair. Wolves anywhere can have coats that grade from almost pure white.

Submit Query
JSP by example - bean

A simple bean (Animal.java) that implements a few properties of an animal:

```java
public class Animal {
    String commonName = null;
    String speciesName = null;
    float adultHeight = 0;
    float adultWeight = 0;
    int topSpeed = 0;
    String description;
    public String getCommonName() {
        return this.commonName;
    }
    public void setCommonName(String commonName) {
        this.commonName = commonName;
    }
    public String getSpeciesName() {
        return this.speciesName;
    }
    ...
```

JSP by example – HTML page

This page is actually straight HTML, defining a standard form that takes the various properties of an animal.

This page, after it's filled out, has been shown on Slide 16.

When the submit button is clicked, the values are sent to a second page (next slide).

```html
<html> <head> <title>Input an Animal</title> </head> <body> <h1>Input an Animal</h1> <FORM action="animaldisplay.jsp" method="POST"> Common Name: <INPUT TYPE="text" name="commonName"><BR> Species Name: <INPUT TYPE="text" name="speciesName"><BR> Adult Height: <INPUT TYPE="text" name="adultHeight"><BR> Adult Weight: <INPUT TYPE="text" name="adultWeight"><BR> Top Speed: <INPUT TYPE="text" name="topSpeed"><BR> Description:<BR> <TEXTAREA rows="5" cols="50" name="description"> </TEXTAREA><BR> <INPUT TYPE="SUBMIT"> </FORM> </body> </html>
```
JSP by example – display form

Display an Animal

Grey Wolf

Species Name: Canis lupis
Adult Weight: 56.0 Kg (123.2000000000002 lbs)
Adult Height: 0.78 m (2.558399906158447 ft)
Top Speed: 56 kph (34.77599999999996 mph)
Description:
Although most wolves have basically grey coats, hence
the common name, the coats usual have a lot of base
yellow interspersed between the salt-and-pepper fay and
black hair. Wolves anywhere can have coats that grade
from almost pure white to black.

JSP by example – “action” page

- `<jsp:useBean>` creates an instance of the `Animal` class, and
  associates it with the ID `animal`.
- `<jsp:setProperty>` tag looks at all the values available on
  the form that was just submitted, and then uses introspection
to determine whether any of the property names match up
with bean property names in the object specified by the name
argument.
  - The result of this is that the newly created animal bean is
    populated with the values from the previous page.

```html
<html> <head> <title>Display an Animal</title> </head> <body> <h1>Display an Animal</h1> <jsp:useBean id="animal" scope="request" class="demo.Animal"/> <jsp:setProperty name="animal" property="*"/> ... (next slide)```

[Turner and Bedell]
JSP by example – “action” page

The display code uses both the \texttt{jsp:getProperty} tag and the raw \texttt{get\_X()} calls to the object itself.

\begin{itemize}
  \item We need to use the raw calls to compute the English unit equivalents of the metric values.
\end{itemize}

\begin{verbatim}
\begin{html}
\texttt{<H2>jsp:getProperty name="animal" property="commonName"/</H2>}
Species Name: \texttt{<jsp:getProperty name="animal" property="speciesName"/>}<BR>
Adult Weight: \texttt{<jsp:getProperty name="animal" property="adultWeight"/> Kg}
  \texttt{(\%= animal.getAdultWeight() * 2.2 \%= Lbs)<BR>}
Adult Height: \texttt{<jsp:getProperty name="animal" property="adultHeight"/> m}
  \texttt{(\%= animal.getAdultHeight() * 3.28 \%= ft)<BR>}
Top Speed: \texttt{<jsp:getProperty name="animal" property="topSpeed"/> kph}
  \texttt{(\%= animal.getTopSpeed() * 0.621 \%= mph)<BR>}
Description:<BR>
\texttt{<jsp:getProperty name="animal" property="description"/>}
\end{html>
\end{verbatim}

JSP custom tags

\begin{itemize}
  \item To \texttt{reuse} common functionalities that you repeatedly have to code in Java on the JSP page.
    \begin{itemize}
      \item For example, you might need to present metric values in English units, as in the previous example.
    \end{itemize}
  \item JSP syntax can be extended by adding new \texttt{custom tag libraries} to JSP.
  \item There are two pieces to a JSP tag library (\texttt{taglib}):
    \begin{itemize}
      \item a Java class that actually handles the JSP
      \item a tag library descriptor file (TLD) that lets JSP know about the new tags
    \end{itemize}
  \item \textbf{Steps}:
    \begin{itemize}
      \item define a Java class that extends \texttt{BodyTagSupport}
      \item inform JSP (via a TLD file) that the new tag is available
      \item with the TLD file placed in the WEB-INF subdirectory of your application, you can write a JSP file that uses it
    \end{itemize}
\end{itemize}
Struts builds applications that are **compliant Web applications** (Webapps), which means:

- A standard directory structure
- Certain standard configuration files (web.xml and so on)
- Dynamic functionality deployed as Java classes and .jsp pages
- A standard Web Archive (.war file) format for deployment

The Struts framework is based on two primary components:

- a **Model-View-Controller** architecture that makes it easy to build flexible applications and
- a set of **JSP custom tags** for building JSP pages.

---

**Struts – MVC**

- **Model:**
  - `Address.java`—A programming model of the user’s address (`PMAddress.java`).
    - `Address.java` would provide a simple way of setting and getting components of the user’s address, and reading it from or storing it to a more permanent storage location (such as a database).
  - `AddressView.jsp`—A view to be used to display the user’s information (`PVAddress.jsp`).
    - `AddressView.jsp` would contain very little in the way of conditional logic; it simply takes the values in `Address.java` and displays them.

- **View:**
  - `AddressView.jsp`—A view to be used to display the user’s information (`PVAddress.jsp`).

- **Controller:**
  - `AddressAction.java`—A controller to assist in validating the user’s entries and choosing the right view to display the results (`PCAddress.java`).
    - If errors are made in entering the data, `AddressAction.java` makes the decision to display an error page or send the user back to the original entry page.
**Struts – tag libraries**

- Used for creating View components.
- The Struts tag libraries provide a set of JSP custom tags that are generally understandable by both JSP developers and page designers.
  - The custom tags have names like `<html:text>` and `<logic:iterate>`.
- The tags include:
  - HTML tags
  - bean tags
  - logic tags
  - The HTML custom tags are used in JSP files for:
    - generating HTML elements,
    - coordinating form processing, and
    - linking the JSP pages (View components) into the rest of the Struts framework.
- It is possible (though not recommended) to use Struts without using any of the Struts custom tags.
- In time, JSTL (Java Standard Tag Library) and JSF (JavaServer Faces) will make the proprietary Struts tags obsolete.

---

**Struts – MVC for webapps**

- Model components (beans) provide access to the information from which the user presentation is built.
- View components are the JSP files that render the HTML to be sent to the user.
- A Controller component performs several primary activities:
  - Validates that the data entered by the user was valid
  - Makes decisions about which Model components need to be accessed or updated and manages these activities
  - Collects the data that the View component will need for display
  - Decides which View component should be displayed to the user
Summary

- **A web-based application** means that an Internet browser manages the rendering of UI content, but the business logic and database state exists on a server.
  - In other than *applet* technology, the presentation logic is also on the server.
- **Scripting languages** can be used in HTML documents to enrich user interactions.
- **Servlet** is a dedicated service running on a server machine to serve multiple clients.
- **Java Server Pages (JSP)** is a variation of the servlet technology. JSP separates the business logic from the presentation.
  - JSP supports reuse of components from previously implemented web pages via the *Custom Tag Library*.
- **Struts** – an open-source project from Apache – provides an additional level of decoupling between the presentation, the business logic and the data obtained from business objects.