## Chapter 3: Interactive Web Applications

### 3.1 Web Server Interfaces

### 3.2 Server-Side Scripting (PHP)

### 3.3 Database Integration

### 3.4 Integration of Client-Side and Server-Side Scripts (AJAX)

### 3.5 Web Programming with Java (Applets, Servlets, Java Server Pages)

- **Applets**
- **Servlets**
- **Java Server Pages**
Server-Side vs. Client-Side Realisation

- **Client-side realisation:**
  - Browser contains execution engine for scripts
  - Web server does not need to execute scripts
  - Script is sent to client as part of server response
  - Examples: JavaScript, Java Applets

- **Server-side realisation:**
  - Web server contains execution engine for scripts
  - Browser does not need to execute scripts
  - Script is executed on server and computes response to client
  - Examples: PHP, Java Servlets, Java Server Pages
Applets

• **Applet:**
  – “application snippet”
  – Java program, embedded in HTML page
  – Executed by browser software
    » directly or via plugin
  – Does not contain a "main()" method!

• **Application:**
  – *Stand-alone* Java program
  – Contains a static "main()" method
Example: Hello-World Applet (1)

import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorldApplet extends Applet {
    public void paint(Graphics g) {
        g.setFont(new Font("SansSerif", Font.PLAIN, 48));
        g.drawString("Hello world!", 50, 50);
    }
}

• Class for applet derived from Applet
• Applet derived from Component
  – Calls paint method
  – Redefining the paint method means it is executed at display time
• Similar to Java Swing, Java 2D
Example: Hello-World Applet (2) – Old Version

```html
<html>
  <head>
    <title> Hello World </title>
  </head>
  <body>

    The Hello-World example applet is called: <br>

    <applet code="HelloWorldApplet.class" width=300>
    </applet>

  </body>
</html>
```

This is frequently used but deprecated HTML syntax!
Example: Hello-World Applet (2) – New Version

```html
<html>
<head>
    <title> Hello World </title>
</head>
<body>

The Hello-World example applet is called: <br>
<object type="application/x-java-applet"
    height="100" width="400">
    <param name="code" value="HelloWorldApplet" />
</object>

</body>
</html>
```

Modern HTML5 syntax
(Note: "classid" not supported in HTML5!)
Assuming "HelloWorldApplet.class" exists
Parameter Passing in HTML – Old Version

Applet:
public class HelloWorldAppletParam extends Applet {
    
    public void paint(Graphics g) {
        String it = getParameter("insertedtext");
        g.setFont(new Font("SansSerif", Font.PLAIN, 48));
        g.drawString("Hello \textcolor{red}{"+it+"} world!", 50, 50);
    }
}

HTML:
<html>
    ...
    <br>
    <applet code="HelloWorldAppletParam.class" width="800">
        <param name="insertedtext" value="wonderful">
    </applet>
    ...
</html>
Parameter Passing in HTML – New Version

Applet:

```java
public class HelloWorldAppletParam extends Applet {
    public void paint(Graphics g) {
        String it = getParameter("insertedtext");
        g.setFont(new Font("SansSerif", Font.PLAIN, 48));
        g.drawString("Hello " + it + " world!", 50, 50);
    }
}
```

HTML:

```html
<html>
    ...
    <br>
    <object type="application/x-java-applet"
            height="100" width="800">
        <param name="code" value="HelloWorldAppletParam" />
        <param name="insertedtext" value="wonderful">
            Java Applets not supported.
        </param>
    </object>
    ...
</html>
```

This is modern HTML5.
Applet Life Cycle

Callback methods:

```java
public class ... extends Applet {
    . . .
    public void init() { . . . }
    public void start() { . . . }
    public void stop() { . . . }
    public void destroy() { . . . }
    . . .
}
```
User Interaction in Applets

• Applets are able to react to user input
  – Define an event handler
  – Register during applet initialisation (init())

• Applets are executed locally, and therefore have full access to local input
  – Mouse movements, key press, …
  – This is not possible with server-side code!

• Applets can make use of graphics libraries
  – For instance Java 2D
  – This is not easily possible with server-side code!
Example: Mouse Interaction in an Applet

public class ClickMe extends Applet implements MouseListener {
    private Point spot;
    private static final int RADIUS = 7;

    public void init() {
        addMouseListener(this);
    }

    public void paint(Graphics g) {
        g.setColor(Color.red);
        if (spot != null) {
            g.fillOval(spot.x - RADIUS, spot.y - RADIUS,
                        RADIUS * 2, RADIUS * 2);
        }
    }

    public void mousePressed(MouseEvent event) {
        if (spot == null)
            spot = new Point();
        spot.x = event.getX();
        spot.y = event.getY();
        repaint();
    }
}

Swing Applets

- Class `javax.swing.JApplet`
  - Derived from `Applet`
  - Is a top level Swing Container

- All Swing GUI components can be used

- Particularities of Swing Applets:
  - Add panels, layout managers etc as with `JFrame`
  - Default layout manager is `BorderLayout`
  - Direct drawing into a Swing applet is not recommended!
  - Redefine method `paintComponent()`
  - Call parent method:
    ```java
    public void paintComponent(Graphics g){
        super.paintComponent(g);
        . . .
    }
    ```
Example: Counter as Swing-Applet (1)

```java
public class CounterSwingApplet extends JApplet {
    CounterPanel counterPanel;

    public void init() {
        counterPanel = new CounterPanel();
        add(counterPanel);
    }
}
// The View
class CounterPanel
    extends JPanel implements Observer {
    private Counter ctr;

    JPanel valuePanel = new JPanel();
    JTextField valueDisplay = new JTextField(10);

    JButton countButton = new JButton("Count");
    JButton resetButton = new JButton("Reset");
    JPanel buttonPanel = new JPanel();

    ...
```
Example: Counter as Swing Applet (2)

```java
public CounterPanel () {
    ctr = new Counter();
    valuePanel.add(new Label("Counter value"));
    . . .
    add(valuePanel, BorderLayout.NORTH);
    countButton.addActionListener(new ActionListener() {
        public void actionPerformed (ActionEvent event) {
            ctr.count();
        }
    });
    . . .
    ctr.addObserver(this);
}

public void update (Observable o, Object arg) {
    valueDisplay.setText(String.valueOf(ctr.getValue()));
}

public void paintComponent(Graphics g){
    super.paintComponent(g);
}
}

class Counter extends Observable {
    . . .}
```
Organisation of Bytecode Files

• `<applet>` and `<applet>` tags allow
  – Declaration of a "codebase" directory (attribute `codebase`)
  – Declaration of a Java archive (JAR) file (attribute `archive`)

• Advantages of codebase:
  – Java bytecode concentrated at one location
  – Fits with Java file conventions

• Advantages of archives:
  – Less files, less HTTP connections, better performance
  – Lower bandwidth requirements due to (LZW) compression
Applets and Security

• "Sandbox security": An applet is not allowed to
  – Open network connections (except of the host from which it was loaded)
  – Start a program on the client
  – Read or write files locally on the client
  – Load libraries
  – Call "native" methods (e.g. developed in C)

• "Trusted" Applets
  – Installed locally on the client, or
  – Digitally signed and verified
  – Such applets may get higher permissions, e.g. for reading/writing files
Advantages and Disadvantages of Java Applets

• Advantages:
  – Interaction
  – Graphics programming
  – No network load created during local interactions
  – Decentally executed – good scalability

• Disadvantages:
  – Dependencies on browser type, browser version, Java version
  – Generally known as a not very reliable technology
  – Debugging is problematic
Chapter 3: Interactive Web Applications

3.1 Web Server Interfaces

... 

3.5 Web Programming with Java (Applets, Servlets, Java Server Pages)

Applets
Servlets
Java Server Pages

Literature:
http://java.sun.com/products/servlet/docs.html
http://tomcat.apache.org/
Basic Principle: Server-Side Execution

1. User fills form
2. Form is sent as HTTP request to server
3. Server determines servlet program and executes it
4. Servlet computes response as HTML text
5. Response is sent to browser
6. Response, as generated by servlet, is displayed in browser
Java-Enabled Web Server

- Servlets are part of Java *Enterprise* Edition (Java EE)
- Prerequisite:
  - Web server must be enabled for Java *servlets*
    - Recognize servlet requests
    - Administer servlets
    - Execute servlets (*servlet container*)
- Before doing any experiments:
  - Install Servlet Container software
  - E.g. Apache Tomcat
Java Servlets

• Java Servlet Specification (JSS):
  – First version: 1996 (Java: 1995)
  – Current version: 2.5 (with Java EE 5)

• Java Server Pages: 1997-1999

• Important reference implementation:
  – "Jakarta" project of the "Apache" group
    » Apache: OpenSource Web server
  – "Tomcat":
    » Supports Servlets and JSP
    » Separate server (used for the examples) or as module for Apache
    » Some development environments include a servlet container

• Basic principle:
  – Web server calls servlet code on request from client (pattern Template Method)
  – Servlet determines response to client by manipulating data structures
Servlet-API: Basics

- abstract class `javax.servlet.GenericServlet`
  - Declares method `service()`
- abstract class `javax.servlet.http.HttpServlet`
  - Subclass of GenericServlet for HTTP servlets
  - Defines standard implementation for method `service()`, calls
    » `doPost()`, `doGet()`, `doPut()`

```java
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
protected void doPost(HttpServletRequest req, HttpServletResponse resp)
```

- interface `javax.servlet.http.HttpServletRequest`
  - Provides information about request, method examples:
    `getAttribute()`, `getParameter()`, `getReader()`
- interface `javax.servlet.http.HttpServletResponse`
  - Access to response construction, method examples:
    `setContentType()`, `getWriter()`
import java.io.*;
import javax.servlet.**;
import javax.servlet.http.**;

public class HelloWorld extends HttpServlet {

    public void doGet(HttpServletRequest request,
            HttpServletResponse response)
            throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Hello World!</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello World!</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
Example: Very Simple Dynamic Servlet

HTML page showing current date and time

```java
public class DateServlet extends HttpServlet {
    public void doGet (HttpServletRequest request,
                        HttpServletResponse response)
        throws ServletException, IOException {
        String title = "Date Servlet Page";
        response.setContentType("text/html");

        PrintWriter out = response.getWriter();
        out.println("<html><head><title>");
        out.println(title);
        out.println("</title><head><body>");
        out.println("<h1>" + title + "</h1>");
        out.print("<p>Current time is: ");
        out.println(new java.util.GregorianCalendar().getTime());
        out.println("</p>");
        out.println("</body><html>");
        out.close();
    }
}
```

Java  HTML
Deployment of Servlet Application

- Servlet is a Java code file (myDate.java)
  - Needs to be compiled
  - Needs to be made known to the Servlet Container

- Deployment:
  - Installation of new server-side java code in the server software
  - Provide a location (directory), called context path
  - Provide metadata on the new application

- Simplest way to deploy on Tomcat:
  - Create file structure according to conventions
  - Copy directory into Tomcat's "webapps" directory
  - Restart Tomcat server

- Other ways for deployment exist
  - E.g. administrative Web interface
File Structure for Deployment

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app ... "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <display-name>My little Date Application</display-name>
  <description>
    Small demo example, by Heinrich Hussmann, LMU.
  </description>
  <context-param>
    <param-name>webmaster</param-name>
    <param-value>husmann@ifi.lmu.de</param-value>
  </context-param>
  <servlet>
    <servlet-name>myDate</servlet-name>
    <description>...</description>
    <servlet-class>myDate</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>myDate</servlet-name>
    <url-pattern>/myDate</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>30</session-timeout>
    <!-- 30 minutes -->
  </session-config>
</web-app>
```

As a single archive: Web application archive (.war file)
# Administration Interface for Server

## Applications

<table>
<thead>
<tr>
<th>Path</th>
<th>Display Name</th>
<th>Running</th>
<th>Sessions</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>Welcome to Tomcat</td>
<td>true</td>
<td>0</td>
<td>Start  Stop  Reload  Undeploy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Expire sessions with idle ≥ 30 minutes</td>
</tr>
<tr>
<td>/docs</td>
<td>Tomcat Documentation</td>
<td>true</td>
<td>0</td>
<td>Start  Stop  Reload  Undeploy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Expire sessions with idle ≥ 30 minutes</td>
</tr>
<tr>
<td>/examples</td>
<td>Servlet and JSP Examples</td>
<td>true</td>
<td>0</td>
<td>Start  Stop  Reload  Undeploy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Expire sessions with idle ≥ 30 minutes</td>
</tr>
<tr>
<td>/host-manager</td>
<td>Tomcat Manager Application</td>
<td>true</td>
<td>0</td>
<td>Start  Stop  Reload  Undeploy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Expire sessions with idle ≥ 30 minutes</td>
</tr>
<tr>
<td>/manager</td>
<td>Tomcat Manager Application</td>
<td>true</td>
<td>1</td>
<td>Start  Stop  Reload  Undeploy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Expire sessions with idle ≥ 30 minutes</td>
</tr>
<tr>
<td>/myDateServlet</td>
<td>My little Date Application</td>
<td>true</td>
<td>0</td>
<td>Start  Stop  Reload  Undeploy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Expire sessions with idle ≥ 30 minutes</td>
</tr>
</tbody>
</table>

## Deploy

Deploy directory or WAR file located on server

- Context Path (required):
- XML Configuration file URL:
- WAR or Directory URL:

[Deploy]
Chapter 3: Interactive Web Applications

3.1 Web Server Interfaces

...

3.5 Web Programming with Java (Applets, Servlets, Java Server Pages)

Applets
Servlets
Java Server Pages

Literature:
http://java.sun.com/products/jsp
http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/
Introductory Example: Java Server Page (JSP)

HTML page with current date/time

```html
<html>
<%! String title = "Date JSP"; %>
<head><title> <%= title %> <title><head>
<body>
<h1> <%= title %> </h1>
<p>Current time is: 
<% java.util.Date now = new GregorianCalendar().getTime(); %>
<% = now %></p>
</body></html>
```

- Basic idea for Java Server Pages:
  - Scripts embedded in HTML ("Scriptlets")
  - Automatic translation into Java Servlet code

![Date JSP](http://localhost:8080/MylspExamples/DateJSP.jsp)

Current time is: Wed Nov 10 12:16:39 EET 2010
Java Server Pages und Servlets

Life of a JSP as sequence diagram:

Client  | JSP-Server | xyz.jsp
---------|------------|--------

xyz.jsp  | install    | xyz-Servlet
---------|------------|--------

compile

start

res1: HTML

res1

res2: HTML

res2

30
JSP Language Elements

Java Server Pages specification provides markup language to be compiled into Java Servlets

Current version is JSP 2.1

JSP can be used to generate arbitrary texts, not only HTML
   Interesting target language: XML

Language elements:
• Script elements
   Embedding of Java code
• Implicit objects
   Access to important parts of servlets
• Directives
   Global instructions for compilation
• Actions
   Standard elements for runtime behaviour
Embedding of Scriptlets in HTML

Two options for embedding:

• JSP-specific syntax: Tags with special symbols
  ```html
  <%, <%%!, <%%=, <%%@, %>, <%%--, --%>
  ```
  - Not elegant, but very practical

• XML-Syntax with *name spaces*
  » XML name space (xmlns) prefix, e.g. "jsp"
  » Prefix definition is bound to a URL
  » Tags take the form `<jsp: xyz>`
  » Used for JSP actions in particular
JSP Script Elements

• Declarations
  – Syntax: `<%! declarations %>`
  – `<jsp:declaration> declarations </jsp:declaration>`
  – Example: `<%! String title = "Date JSP"; %>`
  – Is translated into instance variable of generated class, i.e. visible in all methods of the class.

• Anweisungen (Scriptlets)
  – Syntax: `<%= commands %>`
  – `<jsp:scriptlet> commands </jsp:scriptlet>`
  – Example: `<%= java.util.Date now = new GregorianCalendar().getTime() %>`
  – Local variables are not visible in other methods.

• Expressions
  – Syntax: `<%= expression %>`
  – `<jsp:expression> expression </jsp:expression>`
  – Example: `<%= now %>`
  – Equivalent to `<% out.print(now); %>`
Implicit Objects in JSP Scripts

The most important implicit objects:

• **request** (javax.servlet.http.HttpServletRequest)
  – To read HTTP headers, parameters, cookies etc. from request

• **response** (javax.servlet.http.HttpServletResponse)
  – To write HTTP headers, cookies etc. into the response

• **session** (javax.servlet.http.HttpSession)
  – Tracking of associated interactions ("sessions")

• **out** (javax.servlet.jsp.JspWriter)
  – Output stream (result test)
  – Standard `print()` and `println()` commands

• Example:

```jsp
<% if (request.getParameter("CountButton")!=null) {
    counter.count();
}%>
```
Generated Servlet Code (Excerpt)

```html
<html>
  <%!
  String title = "Date JSP";
  %>
  <head>
    <title> <%=title% %> </title>
  </head>
  <body>
    <h1> <%=title% %> </h1>
    <p>
      Current time is:
      
      <%
      java.util.Date now = new GregorianCalendar().getTime();
      %>
      <%=now%>
    </p>
  </body>
</html>
```

```java
    out.write("\r\n");
    out.write("\t<body>\n");
    out.write("\t\t<h1> ");
    out.print(title);
    out.write(" </h1>\n");
    out.write("\t</p>Current time is:\n");
    out.write("\t\t\t\n");
    java.util.Date now = new GregorianCalendar().getTime();
    out.write("\n");
    out.write("\t\t\n");
    out.print(now);
    out.write("\n");
```
Cleaning Up the JSP Code

• Mixture between Java scriptlets and HTML markup
  – Is confusing
  – Is difficult to maintain

• Approaches to a better structure of the JSP:
  – Use JavaBeans
  – Use Tag Libraries and Expression Language
  – Use JSPX (XML syntax, easier to handle for editing tools)
What Is a JavaBean?

• JavaBeans is a *software component* model for Java

• Software components:
  – Units of software which can be stored, transmitted, deployed, configured, executed without knowing the internal implementation
  – Main usage: Tools for composing components

• Driver for JavaBeans technology: User Interfaces
  – AWT and Swing components are JavaBeans
  – GUI editing tools instantiate and configure JavaBeans

• Main properties of a JavaBean:
  – Has a simple constructor without parameters
  – Provides public getter and setter methods for its properties: `getProp, setProp`
  – Is serializable
  – Supports listener mechanism for property changes
JavaBeans in JSP: Action useBean

- Syntax of useBean Aktion:
  ```jsp
  <jsp:useBean id=localName class=className
  scope=scopeDefn />
  ```
  scope: "page" (current page), "request" (current request), "session" (current session), "application" (full application)

- Reading properties:
  ```jsp
  <jsp:getProperty name=localName property=propertyName/>
  ```

- Writing properties:
  ```jsp
  <jsp:setProperty name=localName property=propertyName/>
  ```
  value=valueAsString

<jsp:getProperty name=counter property=current/>
is equivalent to:
<%=counter.getCurrent();%>
Counter as JavaBean (1)

```java
package counter;
import java.beans.*;

public class Counter extends Object implements java.io.Serializable {

    private static final String PROP_CURRENT = "current";
    private static final String PROP_START_VALUE = "start value";
    private static final String PROP_INCR_VALUE = "incr value";
    private static final String PROP_ENABLED = "enabled";
    private int count;
    private int startValue;
    private int incrValue;
    private boolean enabled;
    private PropertyChangeSupport propertySupport;

    public Counter() {
        propertySupport = new PropertyChangeSupport ( this );
        startValue = 0;
        incrValue = 1;
        reset();
        enabled = true;
    }
}
```
Counter as JavaBean (2)

```java
public int getCurrent () {
    return count;
}

public int getStartValue () {
    return startValue;
}

public void setStartValue (int value) {
    int oldStartValue = startValue;
    startValue = value;
    propertySupport.firePropertyChange
        (PROP_START_VALUE, oldStartValue, startValue);
}

public int getIncrValue () {
    return incrValue;
}

public void setIncrValue (int value) {
    int oldIncrValue = incrValue;
    incrValue = value;
    propertySupport.firePropertyChange
        (PROP_INCR_VALUE, oldIncrValue, incrValue);
}
```
Counter as JavaBean (3)

```java
public boolean getEnabled () {...}

public void setEnabled (boolean value) {...}

public void reset () {
    count = startValue;
}

public void count () {
    if (enabled) {
        int oldCountValue = count;
        count += incrValue;
        propertySupport.firePropertyChange
            (PROP_CURRENT, oldCountValue, count);
    }
}

public void addPropertyChangeListener
    (PropertyChangeListener listener) {
    propertySupport.addPropertyChangeListener (listener);
}

public void removePropertyChangeListener (...) {...}
```
Counter JSP with JavaBeans: HTML Source

```html
<%@ page contentType="text/html" session="true" %>
<%@ page language="java" %>
<html>
    <head>
        <title>Counter Demo Page</title>
    </head>
    <body>
        <jsp:useBean id="counter" scope="session"
            class="counter.Counter"/>
        <% if (request.getParameter("CountButton")!=null) {
            counter.count();
            %>
        </form>
        <input name="CountButton" type="submit" value="Count">
        <input name="ResetButton" type="submit" value="Reset">
```
JSP Directives

• A JSP directive affects the overall structure of the generated servlet class.
• Syntax:
  ```
  <%@ directiveType attribute=value>  or
  <jsp:directive.directiveType attribute=value>
  ```
• Frequently used directive: `page`
  – Import of packages
  – Content type for response
  – Session support
  – Language (java)
  – ...


JSP Tag Libraries

• A tag library is a collection of *custom tags*
  – A custom tag invokes a *custom action*, described by *tag handler* code
• Typical examples of actions which may be invoked by custom tags:
  – Form processing, accessing databases, email, flow control
• Custom tags can be customized on the JSP page by attributes
  – Reduces actual scripting required on page
• Example: Custom *iteration* tag

```jsp
<%@ taglib uri="/tlt" prefix="tlt" %>
<html>...
<body>
  <jsp:useBean id="org" class="Organization"/>
  <table border=2 cellspacing=3 cellpadding=3>
    <tlt:iteration name="departmentName" type="String"
      group="<%= org.getDepartmentNames() %>">
      ...
      <tr>
        <td><a href="list.jsp?deptName=<%= departmentName %>">
          <%= departmentName %></a></td>
      </tr>
    </tlt:iteration>
  </table>
</body></html>
```
Example: Tag Handler for Iteration Custom Tag

```java
private Iterator iterator;

public void setGroup(Collection members) {
    if (members.size() > 0) {
        iterator = members.iterator();
    }
}

public int doStartTag() {
    if (iterator == null) {
        return SKIP_BODY;
    }
    if (iterator.hasNext()) {
        pageContext.setAttribute(name, iterator.next());
        return EVAL_BODY_TAG;
    } else {
        return SKIP_BODY;
    }
}

...
```

Sun JSP Tutorial
Java Server Pages Standard Tag Library (JSTL)

- Standardized collection of custom tags
  - To simplify construction of JSP pages
  - To avoid use of scripting (=Java)
  - To achieve homogeneity in style
- Areas covered by JSTL:
  - General purpose actions (display expressions, set and read attributes, ...)
  - Control flow actions (conditionals, iterators)
  - Tag library validation (enforce coding styles and specific libraries)
  - Frequently uses functions:
    » Accessing URL-based resources
    » Internationalization, text formatting
    » Relational database access (SQL)
    » XML processing
    » String manipulation
JSTL Examples

You have <c:out value="${sessionScope.user.itemCount}"/> items.

```xml
<table>
    <c:forEach var="customer" items="${customers}"
        <tr><td>${customer}</td></tr>
    </c:forEach>
</table>
```

```sql
<sql:query var="customers" dataSource="${dataSource}"
    SELECT * FROM customers
    WHERE country = 'China'
    ORDER BY lastname
</sql:query>

```xml
<acme:fullMoon v
    <c:choose>
        <c:when test=""
        ...
        </c:when>
        <c:otherwise>
        ...
        </c:otherwise>
    </c:choose>
</acme:fullMoon>

```xml
<table>
    <c:forEach var="row" items="${customers.rows}"
        <tr>
            <td><c:out value="${row.lastName}"/></td>
            <td><c:out value="${row.firstName}"/></td>
            <td><c:out value="${row.address}"/></td>
        </tr>
    </c:forEach>
</table>
```
JSP Expression Language (EL)

• With JSTL, a simpler way to access values was introduced
  – Since JSP 2.0, Expression Language is part of general JSP specification
  – Since JSP 2.1, "Unified" Expression Language covering also Java Server Faces (see later)

• Syntax:
  – ${ ... } (immediate evaluation)
  – #{ ... } (deferred evaluation)

• EL features:
  – Access to implicit objects
  – Type system
  – Classical operators
    (with textual keyword equivalents to special characters)
Java Server Faces (JSF)

- Java framework for building Web applications
  - Using a Model-View-Controller architecture

- FacesServlet
  - loads view template
  - builds component tree
  - processes events
  - renders response to client (mostly HTML)

- JSF can use several technologies for user interface display:
  - JSP (using a tag library)
  - Facelets (views described in XML)
  - XUL

- See: https://facelets.dev.java.net/

- Frameworks for integration of JSF and Ajax
  - e.g. IBM "Seam", OpenSource "Dynamic Faces"
Alternatives to JSP

- Microsoft Active Server Pages (ASP)
  - Uses a script language (VBScript, JScript, Perl)
  - Comparable to PHP, no longer maintained!
- Microsoft Server Pages .NET
  - www.asp.net
  - Creation of Web applications with all languages supported by .NET framework
    » E.g. C#, F#
  - Free technology
  - Comparable in functionality to JSP
    <%Response.Write(now())%>
  - Server controls (tags evaluated on server) ("runat="server")
  - Event handlers