Web Application Development 2

Jens Dietrich
The data structure manipulated / displayed is a collection. This is the model of the web application.

Whether the collection is sorted, ordered, contains duplicates etc is very flexible and depends only on the container class used (e.g., TreeSet used in the example is a sorted container that does not allow duplicates).

Actions are represented by name. Depending on the actions encountered, a certain block of code is executed. We could take this one step further and implement separate action objects.

The servlet could then invoke the actions, and delegate the actual view to another servlet or page that visualizes the model. This leads to a common design pattern for web applications called MVC2 (model view controller 2). A popular web app development framework based on MVC 2 is Apache Struts.
Connecting to a Database

• Servlets can connect to relational databases using JDBC.

• This involves the following steps:
  – Loading a driver.
  – Getting a connection.
  – Fire SQL statements.
  – Closing the connection.
Design and Performance Issues

- Database connections can become a bottleneck. This can be addressed by using connection pools.

- Compiled SQL statements can be re-used by using PreparedStatements instead of statements. These statements can be parameterized.

- Connections are usually not created directly by the servlet but either created using a factory (DataSource) or obtained from a naming service (JNDI).

- It is important to release connections (close them or return them to the connection pool). This should be done in a finally clause (so that it is done in any case). Note that close() also throws an exception!
Design and Performance Issues (ctd)

• Connecting to a database may lead to various errors. Therefore, a uniform exception handling strategy should be developed. This can include notifying the end user and writing error details to a server log using the ServletContext log facility. Log packages such as log4j can be used as well.

• Displaying an entire result set in one html table does generally not work in the real world. Instead, the result set can be attached to the session in order to support “next 10 result” pages.

• There are various problems when developing interfaces between OO and RDBMS. This is called the “paradigm mismatch”. The main problem is that databases use reference by value while OO languages use reference by address. So-called middleware/ object persistency framework software tries to ease this.
Design and Performance Issues (ctd)

- Server component models such as EJB also support object-relational mapping. This is done either by programming the mapping or by declaring it in configuration files (container managed persistency).

- Other approaches are:
  - Java Data Objects (like ADOs) – uniform interface for persistent objects
  - Embedded SQL (SQL is embedded in Java, precompiler translates this into standard java code).

- Yet another problem is type mapping. Note that JDBC gives the application program access to BLOBs using streams.
Including and Forwarding Requests

- Servlets have access to a servlet context object that represents the environment.

- Some of the useful features: accessing resources, log facilities, and obtaining request dispatchers.

- Request dispatchers can be used in order to forward the entire request or include another resource (include static text, include content generated by another script or page etc.).

- This can be used to design modular applications. Some issues that must be considered are:
  - Attributes are shared (request, session, context)
  - The output must be well formed (e.g., no nested <html> tags)
  - The content type can be set only once.
The Design of the Example Application

logout

ApplicationServlet

header (static)

login

footer

<forward> <include>

<forward> <include>
Modular Web Application Design

• This design is modular and more flexible than the “one servlet” approach.

• The design principle “separation of concerns” has been applied.

• Modelling these relationships with UML Case tools is not (yet) well-supported! Most URL principles (inheritance, association, aggregation, event based dependency) are not important for web application design!

• But: URL extension mechanisms such as profiles (set of stereotypes and constraints) or MOF models can be used.

• Leads to private / public classification: only a few visible public servlets. In general: create public dispatcher servlets or (alternatively) filters.
Classifying modules in a web application:

- **access:** private / public
- **role:** embedded, top level, dispatcher
- **implementation:** static/dynamic (page/script/filter)

Other modelling issues: attributes (properties) with different scopes (request, session, context)
Classifying relationships in a web application:

1. links (hyperlinks, form submit targets)
2. forward
3. include
4. filter forward
Visualizing web application design.
Exercise

• Redesign the example and address the following issues:
  – The application main site can be accessed directly when the URL is known.
  – The login page is responsible for two tasks: check login and print a form.
Generating Non-HTML Content

• Servlets can generate non-HTML content as well.

• Example include dynamic images (see example), pdf and xml.

• If the content is binary (such as images), an output stream must be used (instead of the writer).

• The appropriate content type must be set.
Example: Generating an Image Dynamically

```java
// set content type
response.setContentType("image/gif");
// get byte stream (and not a writer!)
ServletOutputStream out = response.getOutputStream();
// create an offline frame to draw the image - this is a hack
java.awt.Frame frame = new java.awt.Frame();
frame.addNotify();
Image img = new Image(500, 200);
Graphics g = img.getGraphics();
// draw something
g.setColor(Color.red);
g.draw3DRect(300, 100, 40, 20, true);
...
// encode the image to the stream (uses a class from a 3rd party library!)
GifEncoder encoder = new GifEncoder(img, out);
encoder.encode();
// clean up - very important in order to free resources on the server
g.dispose();
frame.removeNotify();
out.close();
```

Example 9
Case Study: XML Based Web Publishing

• Objectives:
  – Design a layered application that splits the presentation layer into two parts: device-independent and device dependent presentation.
  – Use XML for device independent presentation.
  – This XML is produced from SQL query results.
  – Use XSL Transformations to produce device dependent presentation.
  – Example: transform xml into WML (wireless markup language) or HTML
Case Study: XML Based Web Publishing (ctd)

- Technologies used:
  - Xerces for XML parsing.
  - Xalan for stylesheet transformations.
  - JDOM to create XML from the SQL query.
  - MySQL as database (incl. the JDBC driver).
  - A WAP emulator (such as deck-it) is needed to test the wireless output.
Case Study: XML Based Web Publishing (ctd)

- **Design:**

  - CatalogServlet
  - AbstractCatalogServlet
  - CatalogServlet4Server
  - CatalogServlet4WML
  - SQL2XML
  - DB

  **Flow:**
  - Checks user agent header
  - HTML
  - SQL result set
  - forward req.
  - subclasses
  - forward req.
  - subclasses
  - XML DOM
  - WML
  - XSL Stylesheets are applied here, content type is set.
Case Study: XML Based Web Publishing (ctd)

• Towards a real world application:

• Generate PDF using a post processor (after a stylesheet has been applied to convert the document into FOP).

• Use Apache Cocoon framework based on this architecture.

• Dynamic load balancing – detect browser and return XML with XSL reference directly to the browsers – the latest browser versions can apply the style sheets!

• Do not use DOM (but SAX) – do not keep entire result set in memory.

• Use XSP to build the device independent presentation layer.
Filters

- Servlet filters have been introduced in the servlet API 2.3. Filters are useful for pre – and post processing of requests. A filter has access to both the request and the response object, but does not generate content. Common uses of filters include logging requests or to compress the response (for examples and further information see: http://www.javaworld.com/javaworld/jw-06-2001/jw-0622-filters.html).

- A servlet filter implements the javax.servlet.Filter interface. This interface has two lifecycle methods init and destroy and the method doFilter that applies the filter.

- Filters are extremely useful to add transparent services to web applications.
Example: A Profiling Filter

```java
public class TimerFilter implements Filter {

    private FilterConfig config = null;
    public void init(FilterConfig config) throws ServletException {
        this.config = config;
    }
    public void destroy() {
        config = null;
    }
    public void doFilter(ServletRequest request, ServletResponse response,
                          FilterChain chain) throws IOException, ServletException {
        long before = System.currentTimeMillis();
        chain.doFilter(request, response);
        long after = System.currentTimeMillis();
        String name = "";
        if (request instanceof HttpServletRequest) {
            name = ((HttpServletRequest)request).getRequestURI();
        }
        config.getServletContext().log(name + "": " + (after - before) + "ms");
    }
}
```
Deploying Filters

- Servlet filters are configured in the web xml as follows:
- Filter name-class association:
  
  ```xml
  <filter>
    <filter-name>timerFilter</filter-name>
    <filter-class>TimerFilter</filter-class>
  </filter>
  ```

- URL patterns for which the filter should be applied (in this example: for all servlets):
  
  ```xml
  <filter-mapping>
    <filter-name>timerFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  ```
Customizing the Servlet Lifecycle

- Usually only one servlet instance per class exists and handles all requests.
- This implies in particular that servlets should be stateless (not use instance variables). If instance variables are used, access must be synchronized (e.g. using synchronized blocks).
- The destroy method can be overridden in order to customize with the destruction of the servlet (e.g., in order to release resources such as database connections).
- The init method can be overridden in order to customize the creation of the servlet. This method has a parameter representing the servlet configuration. In particular, this can be used to pass parameters from the web application configuration file to the servlet.
Web Application Development
Server 2 – Java Server Pages and Tag Libraries

Jens Dietrich
Java Server Pages and Tag Libraries

- Lifecycle of a Server Pages
- Deployment
- Directives
- Exception handling
- Expressions and Scriptlets
- Using Java Components in JSP Pages
- Tag Libraries
- Designing for Web Applications using MVC

Additional resources:
The Jakarta Taglibs Project: http://jakarta.apache.org/taglibs/
The Examples

• There are a number of examples for this lecture.
• To run these examples, you must download them as war file first. Deploy the war file (e.g., using the tomcat server, copy the war file into the webapps folder and restart the server) and point your browser to the respective url (e.g. localhost:8080/examples359).
• Click here to download the war file: *servlets\examples359.war*
• You can also try to run the example from the box in my office – I usually keep tomcat and mysql running. This will work only from within the massey network. Try this link:
• [http://it027580:8080/examples359](http://it027580:8080/examples359)
Embedding Code in Web Pages

• Support the separation of scripting and design

• Facilitates the use of design oriented web development tools (WYSIWYG editors such as Frontpage).

• These so-called server pages are not served directly to the client that makes the request, but first compiled into scripts by the web server.

• Different technologies: ASP (Microsoft), PHP (open source), JSP (Java)
Java Server Pages

• JSP pages typically consist of:
  – Static HTML/XML.
  – Special JSP tags
  – Optionally, snippets of (Java) code so-called "scriptlets."

• JSPs can be written and maintained using text/html editors.
JSP Deployment / Lifecycle

- The following steps are performed when a JSP is deployed:
  - A servlet class (java – sources code!!) is generated from the JSP page. Static line of HTML code (e.g. <body>) are replaced by lines of java code like `out.println("<java>")` and combined with java code found in the JSP page. This java class is stored in some temporary folder on the web server and can be found there. Usually, the class name contains the name of the JSP page.
  - The generated servlet class is compiled. This is the reason that tomcat and other application servers need the JDK and not only the JRE (which does not contain a compiler)
JSP Deployment / Lifecycle

• The servlet is loaded by the server (into the classpath of the respective web application).

• The server associates the JSP URL with the servlet class.
Example

JSP source

```html
<html>
<head><title>Server Date - JSP Version</title></head>
<body>
<h1>Server Date - JSP Version</h1>

The current server date is <%= new java.util.Date() %>
</body>
</html>
```

HTML output (as seen by the browser)

```
<h1>Server Date - JSP Version</h1>

The current server date is Mon Sep 29 10:23:03 NZST 2003
```

Page directive – this page is declared as a JSP page producing html.

Embedded java code.
Example (ctd) – The Generated Script

```java
public void _jspService(HttpServletRequest request, HttpServletResponse response) throws java.io.IOException, ServletException {
    ...
    response.setContentType("text/html");
    pageContext = _jspxFactory.getPageContext(this, request, response, 
        true, 8192, true);
    application = pageContext.getServletContext();
    config = pageContext.getServletConfig();
    session = pageContext.getSession();
    out = pageContext.getOut();

    // HTML // begin [file="/jsp/ServerDate.jsp";from=(0,33);to=(6,27)]
    out.write("\r\n");
    out.write("<html>");
    out.write("<head><title>Server Date - JSP Version</title></head>");
    out.write("<body>");
    out.write("<h1>Server Date - JSP Version</h1>");
    out.write("The current server date is ");
    // end

    // begin [file="/jsp/ServerDate.jsp";from=(6,30);to=(6,52)]
    out.print( new java.util.Date() );
    // end

    // HTML // begin [file="/jsp/ServerDate.jsp";from=(6,54);to=(9,0)]
    out.write("\r\n");
    out.write("</body>\r\n");
    out.write("</html>\r\n");
    out.write("");
    // end
}
• JSP directives are messages for the JSP engine.
• Directives do not produce any visible output.
• Syntax: `<%@ ... %>` tag.
• Three directives: page, include and taglib.
• The page directive declares the page, can be used to set the output type, set error handling pages and to import packages.
• Example:
• `<%@ page contentType="text/html import="java.util.*"%>`
The include directive can be used to manage server pages.

```jsp
<%@ include file="copyright.html" %>
```

Important: the include directive will be executed when the servlet is built (at deploy time) and not at runtime. It is not the same as the request dispatcher include (which can also be used in JSPs).
JSP Scripting

- JSP scripting is used to insert code into the JSP page. The default scripting language is Java.

- There are three different script types:
  - Scriptlets `<% code %>` , code is inserted into the servlet's service method
  - Expressions `<%= expression %>` that are evaluated and inserted into the output (e.g., the html)
  - Declarations `<%! code %>` that are inserted into the body of the servlet class, outside the service method (variable declarations, usually this feature is not used as variables in servlets should be avoided, sometimes useful in order to define constants).
Expressions

- Expressions and scriptlets can use some predefined object references, in particular:
  - request, the request;
  - response, the response;
  - session, the current session
  - out, the print writer used to generate output
  - application, a reference to the servlet context
Example (scriptlets and expressions)

```html
<%@page contentType="text/html" import="java.util.*,java.text.*"%>
<html>
<head><title>Server Date - JSP Version</title></head>
<body>
<h1>Server Date - JSP Version</h1>

<%  
    Date now = new Date(); 
    String time = DateFormat.getTimeInstance().format(now); 
    String date = DateFormat.getDateInstance().format(now); 
%>

The current server date is <strong><%= time %></strong>,
the time is <strong><%= time %></strong>.
The request has been made by <strong><%= request.getRemoteHost() %></strong>.

<p>
The user agent is: <strong><%= request.getHeader("user-agent") %></strong>
</p>
</body>
</html>
```

**Example 2**

Expressions. The predefined `request` object is used.
Example (Declarations and Expressions)

```html
<%@page contentType="text/html"%>
<html>
<head><title>A simple (transient) page counter</title></head>
<body>
<h1>A simple (transient) page counter</h1>
<p>Page counter value: <%= ++accessCount %></p>
</body>
</html>
```
JSP Exception Handling

• The page directive has two attributes related to exception handling.

• `errorPage="url"` Specifies the URL of a page (or script) that should process any exceptions or errors (instances of `Throwable`) thrown by the current page.

• `isErrorPage="boolean"` This indicates whether this current page can act as the error page for another JSP page. The default is false.
Example (Exception Handling)

The page that throws the exception.

```html
<%@page contentType="text/html" import="java.sql.*" errorPage=""
<html>
<head><title>Database Connection</title></head>
<body>
<%
    Connection con = DriverManager.getConnection("jdbc:nonsense
    out.println("Connection ok!");
%
</body>
</html>
```

The page that handles the exception.

```html
<%@page contentType="text/html" isErrorPage="true"%>
<html>
<head><title>Error</title></head>
<body>
<h3>Error</h3>
<% application.log("Exception",exception);%>
An error occurred when accessing <%= request.getRequestURI()%>. The error has been logged and the web master has been informed.<p>
<h3>Exception Details:</h3><p>
<tt><% exception.printStackTrace(new java.io.PrintWriter(out));%>
</tt>
</body>
</html>
```

The exception handler is defined here.

This non sense URI triggers an exception.

Here the page is declared as exception handler.

Exception handler pages have a predefined exception reference. Here we use the servlet context API to write to the server logs.
Session Handling

- JSP session handling is straight forward: JSP pages can reference a predefined session object.

- This is the HttpSession instance representing the session.

- Sessions can be turned off using the page directive session attribute: session="false". 
Working with Components

- JSPs can use some special tags to interact with java bean components.
- The server uses reflection to interpret the respective tags.
- Components have a name and a scope (usually the session, but application, page and request are also possible).
- The jsp:getProperty tag is used to read property values, while jsp:setProperty is used to set values.
- The jsp:setProperty tag can set values supplied directly (value) or as request attribute (parameter). This is to facilitate form processing.
- Using components is another step towards a design separating model (the bean) and view (the page). The tags provide model-view adapters (light weight controllers).
Example: Beans and Forms

```jsp
<%@page contentType="text/html"%>
<html>
<head>
<title>Student Editor</title>
</head>
<body>
<jsp:useBean id="student" scope="session"
           class="nz.ac.massey.webtech.jsp.Student" />

<!-- process form -->
<jsp:setProperty name="student" property="name" param="name"/>

<!-- header -->
<h3>Edit details of Mr./Mrs <jsp:getProperty name="student" property="name" /></h3>

<!-- print form -->
<form>
name:<br>
<input type="text" name="name" size="40"
value="<jsp:getProperty name="student" property="name" 
/>">
<p>
   . . .
<input type="submit" value="Apply">
</form>
</body>
</html>
```

Declare bean.

Set property. The new value is fetched from the request (the value of the name parameter)!

Get property.
Tag Libraries

• JSP v1.1 technology supports component based development using custom tags. A tag library is a collection of custom tags.
• The combination of scriptlets and beans tend to make pages complex, using tags can simplify pages.
• Tag libraries are jar file which have an additional descriptor XML file – the tag library descriptor (TLD).
• In particular, the TLD contains associations between tag names and tag handler classes.
Tag Libraries (ctd)

- Tag handler classes are similar to servlets (“back to the future”). Passing attributes and nesting of custom tags is supported.
- Tag libraries are declared using the taglib directive. This directive contains basically a prefix (name space) – tag lib URI association.
- Most professional IDEs (at least their enterprise editions) come with a comprehensive set of tag libraries. The jakarta tag library is a good vendor independent open source implementation.
- TagLibs facilitate strong separation of logic and presentation.
- Some problems remain: e.g. when tag libs produce “deep” html. (Hint: use CSS classes to make this HTML accessible for rendering)
Example: A Simple Custom Tag

The tag lib is associated with the prefix (namespace) sql.

At query (request) time, this tag is replaced with an html table built from the database result set.
Example (ctd) The associated TLD

```xml
<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>SQLTags</short-name>
  <uri>/SQLTags</uri>
  <display-name>SQLTags</display-name>
  <description>A blank tag library template.
  </description>
  <tag>
    <name>Select</name>
    <tag-class>nz.ac.massey.webtech.jsp.taglib.sql.SelectTag</tag-class>
    <body-content>JSP</body-content>
    <small-icon></small-icon>
    <large-icon></large-icon>
    <description></description>
    <example></example>
  </tag>
</taglib>
```

Here the symbolic tag name is associated with the tag handler class.
A proven design for more complex web applications is based on the MVC pattern.

The Apache Struts framework uses this design.

Model

View(s)
(JSP + tags)

controller
servlet

Action(s)

Action names are encoded in the URL or in request parameters. Actions are objects registered in an action map.

pre filter
chain
(optional, may include other servlets)

post filter
chain

request

forward

response
Design of a Shopping Mall using MVC

- Model
- View(s) (JSP + tags)
  - Visualize shopping cart as xml list
  - Convert xml into html or wml
- Controller servlet
  - pre filter chain (optional, may include other servlets)
  - check session
  - action(s)
  - forward

- Request
  - log and profile requests

- Response
  - log and profile requests
  - post filter chain

- Actions:
  - save, remove, add, check out, logout, find

- The shopping cart + wish list + bookmarks

Visualize shopping cart as xml list
Convert xml into html or wml