CS520 Web Programming
Servlet and JSP Review

Chengyu Sun
California State University, Los Angeles

What We Won't Talk About (But Expect You to Know)

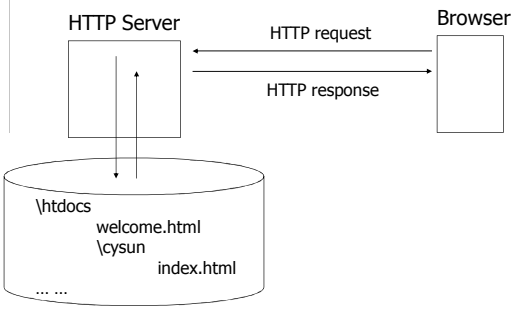
- Java
  - Use of collection classes like lists and maps
- HTML and CSS
  - Tables and forms
- Database access
  - Use of a DBMS
  - JDBC

URL

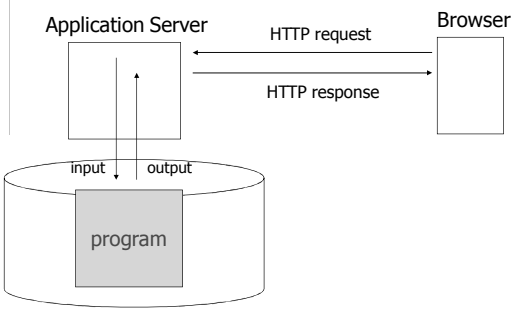
http://cs.calstatela.edu:8080/cysun/index.html

?? ?? ?? ??

Static Web Pages



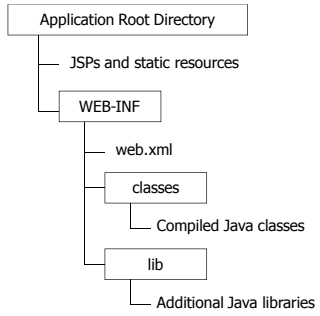
Deliver Dynamic Content



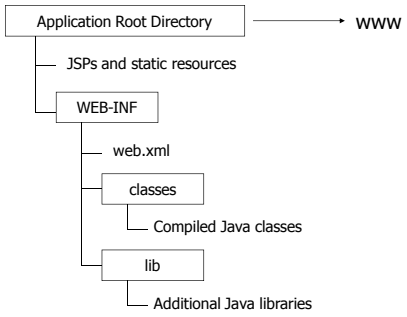
Web Application Development

- Server-side
  - CGI
    - C, Perl
  - Java EE
  - ASP.NET
    - VB, C#
  - PHP
  - Ruby
  - Python
- Client-side
  - HTML, CSS
  - JavaScript
  - Applet
  - Flash

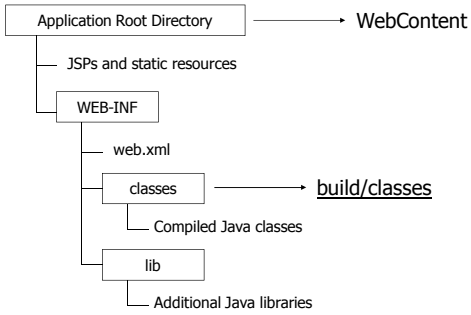
Directory Structure of a Java Web Application



Directory Structure on CS3



Directory Structure of an Eclipse Dynamic Web Project



Servlet HelloWorld

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {
   public void doGet( HttpServletRequest request,
                      HttpServletResponse response )
      throws ServletExceptoin, IOException
   {
      PrintWriter out = response.getWriter();
      out.println( "Hello World" );
   }
}
```

Some Simple Observations

- Inherits from HttpServlet
  - http://download.oracle.com/docs/cd/E17802_01/products/products/servlet/2.5/docs/servlet-2_5-mr2/javax/servlet/http/HttpServlet.html
- There's no `main()` method
- doGet() and doPost()
  - Input: HttpServletRequest
  - Output: HttpServletResponse → sent back to the client browser

About web.xml

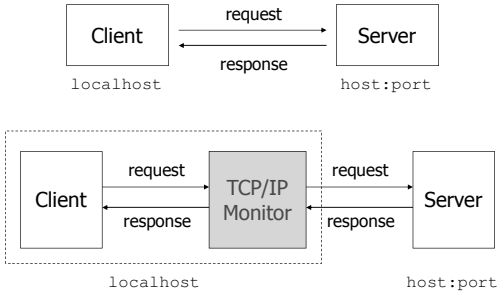
- Web application deployment descriptor
  - <welcome-file-list>
  - <servlet> and <servlet-mapping>
    - <load-on-startup>
- More about web.xml in Java Servlet Specification

Wildcard in Servlet Mapping

- A string beginning with a **/** and ending with a **/***
  - E.g. `/*`, `/content/*`
- A string beginning with a ***.**
  - E.g. `*.html`, `*.do`

See Servlet Specification 2.4, Section SRV.11.2

TCP/IP Monitor in Eclipse ...



... TCP/IP Monitor in Eclipse

- `Window` → `Preferences` → `Run/Debug` → `TCP/IP Monitor`
- Example: monitor the access of http:///sun.calstatela.edu/~cysun/public/form.html
  - *Local Monitoring Port?? Host?? Port??*
  - *Browser URL??*

HTTP Request Example

http://cs3.calstatela.edu:8080/whatever

GET /whatever HTTP/1.1
Host: cs3.calstatela.edu:4040
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.7.3) ...
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,...
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Cookie: nxt/gateway.dll/uid=4B4CF072; SITESERVER=ID=f1675...

HTTP Request

- Request line
  - Method
  - Request URI
  - Protocol
- Header
- [Message body]

Request Methods

- Actions to be performed regarding the resource identified by the *Request URI*
- Browser
  - GET
  - POST
- Editor
  - PUT
  - DELETE
- Diagnosis
  - HEAD
  - OPTIONS
  - TRACE

HttpServlet Methods

service()

- GET → doGet()
- POST → doPost()
- PUT → doPut()
- DELETE → doDelete()
- HEAD → doHead()
- OPTIONS → doOptions()
- TRACE → doTrace()

HttpServletRequest

- http://download.oracle.com/docs/cd/E17802_01/products/products/servlet/2.5/docs/servlet-2_5-mr2/javax/servlet/http/HttpServletRequest.html

Use Request Parameters as Input

- Query string
  - *?param1=value1&param2=value2&...*
- Form data
  - GET vs. POST

Use Request URI as Input

?param1=value1&param2=value2

⇩

/param1/value1/param2/value2

Servlet Examples

- Add
  - `Add?a=10&b=20`
- GuestBook
- File upload

Guest Book

Your name:

Your comments:

Submit

Joe says:	this is a nice site!
Jane says:	Hello
Tom says:	Are you a student?

File Upload – The Form

```
<form action="FileUploadHandler"
        method="post"
        enctype="multipart/form-data">

    First file: <input type="file" name="file1" /> <br />
    Second file: <input type="file" name="file2" /> <br />

    <input type="submit" name="upload" value="Upload" />

</form>
```

File Upload – The Request

```
POST / HTTP/1.1
Host: cs.calstatela.edu:4040
[...]
Cookie: SITESERVER=ID=289f7e73912343a2d7d1e6e44f931195
Content-Type: multipart/form-data; boundary=---------------------------146043902153
Content-Length: 509

-----------------------------146043902153
Content-Disposition: form-data; name="file1"; filename="test.txt"
Content-Type: text/plain

this is a test file.

-----------------------------146043902153
Content-Disposition: form-data; name="file2"; filename="test2.txt.gz"
Content-Type: application/x-gzip

???□??????UC
```

Apache commons-fileupload

- http://commons.apache.org/fileupload/using.html

```
FileItemFactory fileItemFactory = DiskFileItemFactory();
ServletFileUpload fileUpload = new ServletFileUpload( fileItemFactory );

List items = fileUpload.parseRequest( request );
for( Object o : items )
{
    FileItem item = (FileItem) items;
    if( ! item.isFormFiled() ) {...}
}
```

HTTP Response Example

```
HTTP/1.1 200 OK
Content-Type: text/html;charset=ISO-8859-1
Content-Length: 168
Date: Sun, 03 Oct 2004 18:26:57 GMT
Server: Apache-Coyote/1.1

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html><head><title>Servlet Life Cycle</title></head>
<body>
n is 299 and m is 440
</body>
</html>
```

HTTP Response

- Status line
  - Protocol
  - Status code
- Header
- [Message body]

Status Codes

- 100 – 199: Informational. Client should respond with further action
- 200 – 299: Request is successful
- 300 – 399: Files have moved
- 400 – 499: Error by the client
- 500 – 599: Error by the server

Common Status Codes

- 404 (Not Found)
- 403 (Forbidden)
- 401 (Unauthorized)
- 200 (OK)

Header Fields

- Request
  - Accept
  - Accept-Charset
  - Accept-Encoding
  - Accept-Language
  - Connection
  - Content-Length
  - Cookies
- Response
  - Content-Type
  - Content-Encoding
  - Content-Language
  - Connection
  - Content-Length
  - Set-Cookie

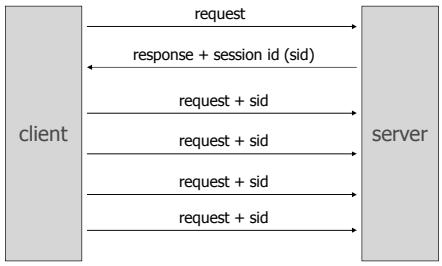
Example: File Download

- Download file using a servlet
  - Indicate file name
  - Indicate whether file should be displayed or saved

Session Tracking

- The Need
  - shopping cart, personalization, ...
- The Difficulty
  - HTTP is a "stateless" protocol
  - Even persistent connections only last seconds
- The Trick??

General Idea



Servlet Session Tracking API

- HttpServletRequest
  - HttpSession getSession()
- HttpSession
  - setAttribute(String, Object)
  - getAttribute(String)
  - setMaxInactiveInterval(int)
    - Tomcat default: 30 seconds
  - invalidate()

Example: Improved GuestBook

- A use only needs to specify a name when he or she adds the first comment

Sharing Data among Servlets

- HttpServlet
  - getServletContext()
- HttpServletContext
  - setAttribute(String name, Object value)
  - getAttribute(String name)

Example: GuestBook Using Two Servlets

- Separate GuestBook into two servlets
  - GuestBook
  - AddComment

Scopes and Data Sharing

- Application scope – data is valid throughout the life cycle of the web application
- Session scope – data is valid throughout the session
  - redirect, multiple separate requests
- Request scope – data is valid throughout the processing of the request
  - forward
- Page scope – data is valid within current page

Access Scoped Variables in Servlet

- Application scope
  - `getServletContext()`
- Session scope
  - `request.getSession()`
- Request scope
  - `request`
- Page scope (in JSP scriptlet)
  - `pageContext`

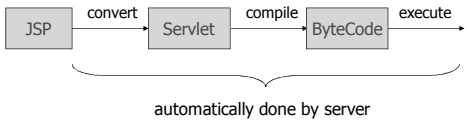
Java Server Page (JSP)

- Why?
  - It's tedious to generate HTML using println()
  - Separate presentation from processing
- How?
  - *Java code* embedded in *HTML documents*

HelloJSP.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<HTML>
<HEAD><TITLE>JSP Hello World</TITLE></HEAD>
<BODY>Hello World on <%= new java.util.Date() %>.
</BODY>
</HTML>
```

How Does JSP Work?



automatically done by server

- Look under $CATALINA_HOME/work/Catalina/localhost/*context_name*

JSP Components

- HTML template text
- Code elements of Java
  - Directives
  - Scripting elements
  - Beans
  - Expression language
  - Custom tag libraries

Directives

- Affect the overall structure of the JSP/servlet
- **<%@** *type attr="value" ...* **%>**
- Three type of directives
  - page
  - include
  - taglib

Directive Examples

```
<%@ page import="java.util.*, java.util.zip.*" %>
<%@ page contentType="text/html" %>
<%@ page pageEncoding="Shift_JIS" %>
<%@ page session="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ include file="path_to_file" %>
```

Scripting Elements

- JSP Expression: `<%= ... %>`
- JSP Scriptlet: `<% ... %>`
- JSP Declaration: `<%! ... %>`

RequestCounter Servlet

```
public class RequestCounter extends HttpServelt {

   int counter = 0;

   public void doGet( HttpRequest request, HttpResponse response )
   {
      ++counter;
       PrintWriter out = response.getWriter();
       out.println( "<html><body>" );
       out.println( "You are visitor #" + counter + "." );
       out.println( "</body></html>");
   }

}
```

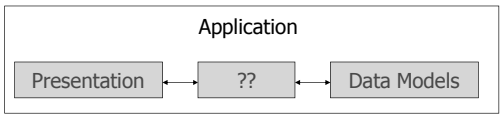
Request Counter JSP with Scripting Elements

```
<%! int counter=0; %>

<% ++counter; %>

<html><body>
You are visitor #<%= counter %>.
</body></html>
```

Problems with Scripting Elements

- Mixing presentation and processing
  - hard to debug
  - hard to maintain
- No clean and easy way to reuse code
- Solution – separate out Java code

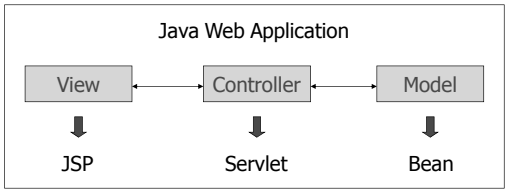
Separate Data and Presentation



- Presentation
  - Create UI
  - Input and output
  - Web, JFC/Swing ...
- Data Models
  - Independent of UI
  - POJO (Beans)
  - E.g. the `GuestBookEntry` class

Model-View-Controller (MVC) Architecture

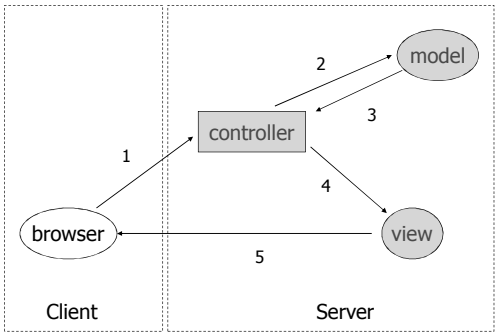
- A.K.A. Model 2 Architecture



About MVC

- Originate from the work on *Smalltalk*
- Widely used in GUI applications

MVC in a Web Application ...



... MVC in a Web Application

1. Process request
2. Create/update beans
3. Store beans in request, session, or application scope
4. Forward request to JSP page
5. Extract data from beans and display

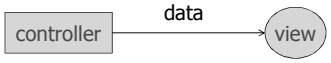
Example: GuestBook Using MVC

- **M**odel
  - GuestBookEntry.java
- **V**iew
  - AddComment.jsp, GuestBook.jsp
- **C**ontroller
  - AddComment.java, GuestBook.java

About Bean Properties

- Property naming conventions
  - 1st letter is always in lower case
  - 1st letter must be capitalized in *getter* (*accessor*) and/or *setter* (*mutator*)
- Property types
  - read-only property: only *getter*
  - write-only property: only *setter*
  - read/write property: both

From Controller to View ...




```
request.getRequestDispatcher( "path_to_jsp" )
        .forward( request, response );
```

... From Controller to View

- Objects in *application* and *session scope* are shared by all servlets and JSPs of the application
- Additional data can be passed from servlet to JSP in *request scope*

```
request.setAttribute( "objName", obj );
request.getRequestDispatcher( "path_to_jsp" )
        .forward( request, response );
```

Access Data in JSP

- Expression Language (EL)
- JSP Standard Tag Library (JSTL)

Expression Language

- Expression Language (EL)
  - A JSP 2.0 standard feature
  - A more concise way to write JSP expressions
    - vs. <%= expression %>
  - Java's answer to scripting languages
- EL Syntax

${ *expression* **}**

Expression

- Literals
- Operators
- Variables
- Functions
  - see Custom Tag Libraries

EL Literals

- `true`, `false`
- `23`, `0x10`, ...
- `7.5`, `1.1e13`, ...
- `"double-quoted"`, `'single-quoted'`
- `null`

- No char type

EL Operators

- Arithmetic
  - +, -, *, /, %
  - div, mod
- Logical
  - &&, ||, !
  - and, or, not
- Relational
  - ==, !=, <, >, <=, >=
  - eq, ne, lt, gt, le, ge
- Conditional
  - ? :
- empty
  - check whether a value is null or empty
- Other
  - [], ., ()

EL Evaluation and Auto Type Conversion

${2+4/2}	
${2+3/2}	
${"2"+3/2}	
${"2"+3 div 2}	
${"a" + 3 div 2}	
${null == 'test'}	
${null eq 'null'}	

${empty ""}	
${empty param.a}	
${empty null}	
${empty "null"}	
${"abc" lt 'b'}	
${"cs320" > "cs203"}	

EL Variables

- You cannot declare new variables using EL (after all, it's called *"expression"* language).
- However, you can access beans, implicit objects, and previously defined scoped variables.

Implicit Objects

- pageContext
  - servletContext
  - session
  - request
  - response
- param, paramValues
- header,headerValues
- cookie
- initParam
- pageScope
- requestScope
- sessionScope
- applicationScope

Limitations of EL

- Only expressions, no statements, especially *no control-flow statements*

JSTL

JSTL Example

```
<%@ page contentType="text/html" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<html><head><title>JSTL Hello</title></head>
<body>

<c:out value="Hello World in JSTL." />

</body>
</html>
```

taglib Directive

- URI
  - A unique identifier for the tag library
  - NOT a real URL
- Prefix
  - A short name for the tag library
  - Could be an arbitrary name

JSP Standard Tag Library (JSTL)

Library	URI	Prefix
Core	http://java.sun.com/jsp/jstl/core	c
XML Processing	http://java.sun.com/jsp/jstl/xml	x
I18N Formatting	http://java.sun.com/jsp/jstl/fmt	fmt
Database Access	http://java.sun.com/jsp/jstl/sql	sql
Functions	http://java.sun.com/jsp/jstl/functions	fn

http://java.sun.com/products/jsp/jstl/1.1/docs/tlddocs/index.html

JSTL Core

- Flow control
  - <c:if>
  - <c:choose>
    - <c:when>
    - <c:otherwise>
  - <c:forEach>
  - <c:forToken>
- Variable support
  - <c:set>
  - <c:remove>
- URL
  - <c:param>
  - <c:redirect>
  - <c:import>
  - <c:url>
- Output
  - <c:out>
- Exception handling
  - <c:catch>

Branch Tags

```
<c:if test="${!cart.notEmpty}"> The cart is empty.</c:if>

        <c:choose>
           <c:when test="${!cart.notEmpty}">
              The cart is emtpy.
           </c:when>
           <c:otherwise>
              <%-- do something --%>
           </c:otherwise>
        </c:choose>
```

Loop Tags

```
<%-- iterator style --%>
<c:forEach items="${cart.items}" var="i">
    ${i} <br>
</c:forEach>

<%-- for loop style --%>
<c:forEach begin="0" end="${cart.size}" step="1"  var="i">
    ${cart.items[i]}
</c:forEach>
```

<forToken ....> ➔ Exercise

Set and Remove Scope Variables

In Login.jsp

```
<c:set var="authorized" value="true" scope="session"/>
```

In CheckLogin.jsp

```
<c:if test="${empty sessionScope.authorized}">
   <c:redirect url="Login.jsp" />
</c:if>
```

URL Tags

```
<c:import url="/books.xml" var="something" />
<x:parse      doc="${something}"
              var="booklist"
              scope="application" />

<c:url var="url" value="/catalog" >
   <c:param name="Add" value="${bookId}" />
</c:url>
<a href="${url}">Get book</a>
```

Output

```
<c:out value="100" />          ⇨   ${100}
<c:out value="${price}" />         ${price}
```

- You want to use <c:out> if
  - *escapeXML*=true
  - *value* is a Java.io.Reader object

Character Conversion

- When *escapeXML*=true

<	&lt;
>	&gt;
&	&amp;
'	&#039;
"	&#034;

Exception Handling

- <c:catch>

Format Date and Time

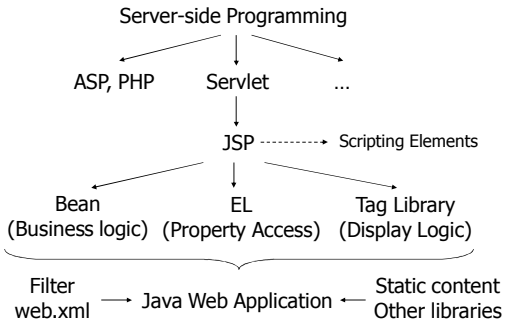
```
<fmt:formatDate value="${date}" type="date" />

<fmt:formatDate value="${date}" type="time" />

<fmt:formatDate value="${date}" type="both" />

<fmt:formatDate value="${date}"
        pattern="yyyy-MM-dd hh:mm:ss a" />
```

See http://java.sun.com/javase/6/docs/api/java/text/SimpleDateFormat.html for the date formatting patterns.

About MVC

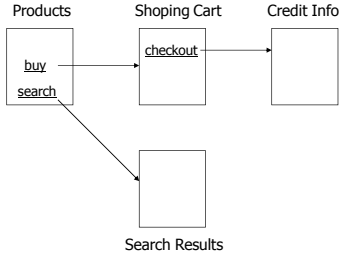
- Servlets do NOT generate HTML directly
  - No `out.println()`
  - Redirect and Forward
- JSPs are only used for display
- Use of scopes
  - Application and session scopes are shared by all servlets and JSPs of the application
  - Request scope is often used for passing data from a servlet to a JSP

Summary



Web App Development – Where Do We Start?

- Control flow driven approach



Web App Development – Where Do We Start?

- Data driven approach