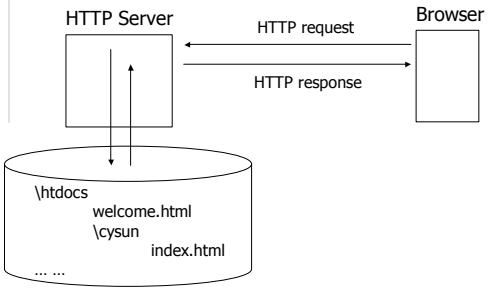
CS520 Web Programming
Servlet and JSP Review

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

Static Web Pages

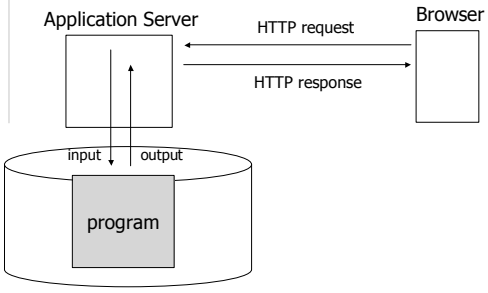


URL

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

?? ?? ?? ??

Deliver Dynamic Content



Servlet Hello World

```
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" );
   }
}
```

Program I/O

- Input: HTTP Request
- Output: HTTP Response

HTTP Request Example

http://cs3.calstatela.edu:4040/whatever

GET /whatever HTTP/1.1
Host: cs.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

- get*This*(), get*That*(), ...
- http://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/http/HttpServletRequest.html

Use Request Parameters as Input

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

Servlet Examples

- Add
- GuestBook

Use Request URI as Input

?param1=value1&param2=value2

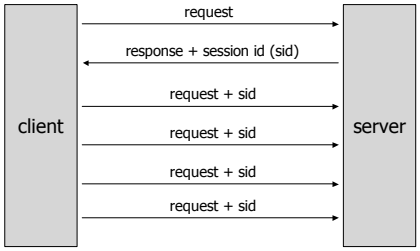
⇩

/param1/value1/param2/value2

Session Tracking

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

General Idea



Three Ways to Implement Session Tracking

- URL Re-writing
- Hidden form field
- Cookies

Cookies

- Issued by the server
  - HTTP Response: Set-Cookie
- Part of the next client request
  - HTTP Request: Cookie

Cookie Attributes

- Name, Value
- Host/Domain, Path
- Require secure connection
- Max age
- Comment (Version 1)

Servlet Cookie API

- Cookie
  - get*This*(), set*That*() ...
  - setMaxAge(int)
    - 1000??, -1??, 0??
- HttpServletResponse
  - addCookie(Cookie)
- HttpServletRequest
  - Cookie[] getCookies()

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 need to specify a name when he or she leaves the first message

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

More Response Header Fields

- Location
  - for redirect
- Refresh
  - "Push"
  - Incremental display
- Cache-Control, Expires, Pragma
  - for cache policies

Example: File Download

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

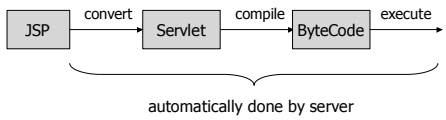
Java Server Page (JSP)

- 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?



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

Some Simple Observations about the JSP/Servlets

- In package org.apache.jsp
- _jspService() handles everything
- HTML text → out.write(...)
- A number of pre-defined variables
  - request, response, out
  - config, pageContext
  - page, session, application

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" %>

Comments

- **<%--** Hidden Comments **--%>**
- **<!--** Output (HTML) Comments **-->**

Scripting Elements

- JSP Expression
- JSP Scriptlet
- JSP Declarations

JSP Expression

- **<%=** Java expression **%>**
  - What's an expression??
- Converted to `out.print(…)` in `_jspService()`

JSP Scriptlet

- **<%** Java code **%>**
- All code goes *into* `_jspService()`

JSP Declaration

- **<%!** class variables or methods **%>**
- All code goes *outside* `_jspService()`

Scripting Elements Example

- Convert RequestCounter servlet to JSP using scripting elements

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

Java Beans

- Initially designed for GUI builders
- *Beans* support
  - Introspection
  - Customization
  - Events
  - Properties
  - Persistence

Java Beans for Dummies

- A zero-argument constructor (*)
- No public class variables (**)
- Properties
  - Properties are defined by *getter* and/or *setters*, e.g. `getFoo()` and `setFoo()`
  - Properties != Class variables

(*) Only needed if the bean is created in a JSP using <jsp:useBean>.
(**) You can have public class variables, but they can't be accessed directly in JSP.

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

Bean Tags and Attributes

- jsp:useBean
  - class
  - id
  - scope
    - page (default)
    - request
    - session
    - application
- jsp:getProperty
  - name
  - property
- jsp:setProperty
  - name
  - property
  - value
  - param

Scopes and Data Sharing

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

Simple Bean Example

- RequestCounter using a Counter bean

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
    - e.g. associative array
- 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

Simple EL Example

- RequestCounter that shows visitor's IP address

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

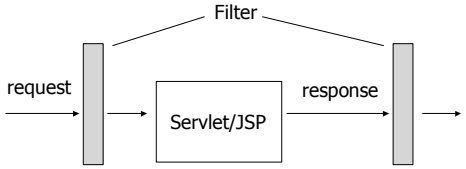
- <c:catch>

Bean + EL + JSTL Example

- GuestBook.jsp

Filter

- Intercept, examine, and/or modify request and response



Filter Example

- CheckParamFilter

Putting It All Together - Java Web Application

- Components
  - Servlets
  - Filters
  - JSPs
  - Classes
  - Static documents (HTML, images, sounds etc.)
  - Meta information
- *Everything in the same context is considered part of <u>one</u> application*

Directory Structure

- webapp root
  - WEB-INF/
  - WEB-INF/classes
  - WEB-INF/lib
  - WEB-INF/web.xml *(deployment descriptor)*

web.xml

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

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
        version="2.4" >

  <description>
        A Java web application example for CS320.
  </description>
  <display-name>CS320 Web App Example</display-name>

</web-app>
```

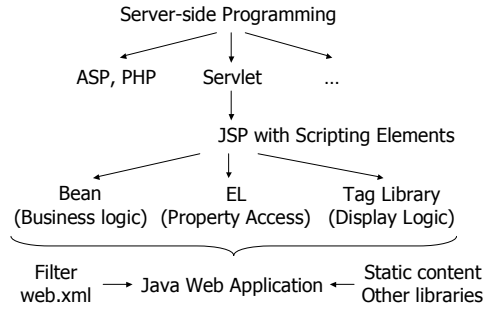
Some <web-app> Elements

- <welcome-file-list>
- <error-page>
- <servlet> and <servlet-mapping>
- <filter> and <filter-mapping>
- <session-config>
- <context-param>

More About web.xml

- Java Servlet 2.4 Specification
  - SRV.13.4

Summary


Server-side Programming
ASP, PHP
Servlet
…
JSP with Scripting Elements
Bean
(Business logic)
EL
(Property Access)
Tag Library
(Display Logic)
Filter
web.xml
Java Web Application
Static content
Other libraries