CMSC 433
Programming Language Paradigms and Technologies
William Pugh
Sept 13th, 2011
JSPs, Servlets and more
Project 1: Idea book

- Due Monday, Sept 19th, 6pm
- signed in users add ideas and can vote + or - on ideas
- each user gets one vote per idea, can change vote
- list ideas by net votes
- Details at http://goo.gl/ZLN3G
- Individual project: do not share or look at code by other students
Submitting project 1

- Deploy to Google Appengine
- Commit to CVS and submit project to submit server
Project 2

- Project 2 will be to complete several elements of WebGoat
Do something impressive

- You are just getting exposed to the bare edges of what can be done with web servers and appengine
  - mail, chat, sms, cron jobs
Recommended Servlet/JSP book

- I’m a fan of the head first series.
- Some people hate them
- Available free on line via Safari books
- when connecting from campus
Search engines are your friend

- Bing/google "core jstl" and you will get lots of pointers to the core taglibs in jstl
Scriptless JSPs

- No java source code in your jsp file
- Can go part way just using beans
- But probably want to go to using JavaServer Pages Standard Tag Library (JSTL)
JSP Header

<%@ page contentType="text/html;charset=UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<c:choose>
    <c:when test="${empty user}"/>
        <p>Welcome. <a href="${loginURL}">Log in</a> to sign ideas you post.</p>
    </c:when>
    <c:otherwise>
        <p>Hello <c:out value="${user.nickname}" /></p>
        (<a href="${logoutURL}">sign out</a>). </p>
    </c:otherwise>
</c:choose>
Two different things going on

• ${....} is a JSTL expression

• <c:choose>, <c:when> and <c:otherwise> are JSTL tags

• the c comes from the prefix for the taglib, but c is standard for the JSTL core
JSTL EL expressions

- constant
- attribute_name
- implicit_object_name
- exp.name
- exp[exp]
- unaryOp exp
- exp binaryOp exp
Servlet attributes

- An attribute can be associated with a session, request or page
- Each attribute has a name
- Easy to set attributes via filters
  - For example, user, loginURL and logoutURL are attributes
Setting attributes in a filter

UserService userService = UserServiceFactory.getUserService();

User user = userService.getCurrentUser();

request.setAttribute("user", user);

String requestURI = request.getRequestURI();

String logIn = userService.createLoginURL(requestURI);

request.setAttribute("loginURL", logIn);

String logOut = userService.createLogoutURL(requestURI);

request.setAttribute("logoutURL", logOut);
empty \( \exp \)

- True if \( \exp \) is null, an empty string, or an empty collection
exp.name

• Expects that exp evaluates to an object that has a getName or isName method

• for ${user.nickname}, user is an attribute of type User, which has a method
  public String getNickname() { … }

• Or, if exp is a map, exp.name is the same as exp.get(“name”)
If $exp_1$ is a map or list,
• gives $exp_1.get(exp_2)$

If $exp_1$ is an array,
• gives $exp_1[exp_2]$

Otherwise, same as $exp_1.exp_2$, except that $exp_2$ is evaluated, rather than being required to be a constant
implicit objects

• param
• header
• cookie
• pageContext
  • gives access to request and response
• plus 7 more, less common ones
Examples

- On test.jsp, have
  - `<p>x = ${param.x}</p>`

- request test.jsp?x=5
- Will generate `<p>x = 5</p>`
<c:choose>, <c:when>, ...

• <c:choose> can have any number of <c:when> clauses
• executes first with test that evaluates to true
• if none match, will execute <c:otherwise> clause
• Can also use <c:if test="${exp}$">, but no way to define an else clause
<c:forEach var="greeting" items="${greetings}">
    <c:choose>
        <c:when test="${empty greeting.author}"">
            <p>An anonymous person wrote:</p>
        </c:when>
        <c:otherwise>
            <p><c:out value="${greeting.author.nickname}" /> wrote:</p>
        </c:otherwise>
    </c:choose>
    <blockquote><c:out value="${greeting.content}" /></blockquote>
</c:forEach>
<c:forEach ...>

• Can iterator through a collection
  • <c:forEach var="greeting" items="${greetings}"/>

• Or a range of values
  • <c:forEach var="x" begin="0" end="10" step="2"/>

• Ended by </c:forEach>
<c:out value="..."/>

- Value is written to JSP output
- output is XML escaped
- can disable escaping with xmlEscape="false"
- Can supply default for when value is empty
- Can just embed ${...} in JSP body, but no escaping will occur
<c:set>

• Define Can use `<c:set var="name" value="${exp}"/>

• or

  • `<c:set var="name">
    <c:out value="${exp}"/>
  </c:set>`
• Encode a url
  • encodes parameters for the URL
    • parameter values are URL encoded
  • adds session identifier if needed
    • required to support sessions if cookies not enabled
c:url example

```html
<c:url var="previous" value="foreach.jsp">
  <c:param name="first"
    value="${param.first - noOfRows}" />
</c:url>

<a href="<c:out value="${previous}"/>">Previous Page</a>
```
Filters and forwarding

• You can define filters that validate/transform/lookup parameters, and put them into attributes
  • can also check authorization, redirect or forward

• A servlet or JSP page can also redirect or forward to another page or include another page
  • jsp's redirect using c:redirect, forward using jsp:forward
Forwarding in a servlet

// forward request
RequestDispatcher view = req.getRequestDispatcher("/view.jsp");
view.forward(req, resp);

// send redirect
resp.sendRedirect("/guestbook.jsp");
Forwarding in a JSP

- `<jsp:forward page="view.jsp"/>`
Forwarding and redirects

• You can't do either if you've already flushed any output

• Should be OK if you've written output but not flushed it

• Forward takes all the information in the request (including attributes) and says "use this URL to determine the output that should be generated"
forward vs. redirect

- a forward stays within the server
  - browser is ignorant of the redirected location
- a redirect goes back to the browser
  - redirected URL shows in the location field of the browser
  - reload loads the redirected URL
XSRF

- Cross site request forgery
- In order to teach you how to build systems, you need to learn how to crack systems. Be responsible. You can very quickly get hit with academic dishonesty or criminal charges
Web page for granting an extension on a project

- HTML to grant an extension

```html
<form class="form" name="form" action="/action/instructor/GrantExtension" method="POST">
  <input type="text" name="extension">
  <input type="hidden" name="studentRegistrationPK" value="36">
  <input type="hidden" name="projectPK" value="12">
  <br>
  <input type="submit" value="Grant extension">
</form>
```
First attempt

• You just save that HTML into a file, load it in your web browser, and hit the "Grant extension" button

• Won't work. Why not?
Need to be logged in as instructor for course

- The form isn't the only thing that gets sent to the server. The browser also sends cookies, which identify the session.

- The server checks the request against the session to see that the user stored in the session is authorized to be an instructor for the course.

- Perhaps you could get me to click the submit it?

- but then I'd see the response
But, maybe I won't click it

• Pre-populate all form fields
  • turn all text fields into hidden input fields with the values you want

• put the form inside a `<div style="display: none">` block

• Add a Javascript to submit the form after the page finished loading
  • server response to form submission is ignored
XSRF protection

• a request to a browser includes a referrer field
• for POST requests, check that the referrer field is present and is from the same web site
• Don't want to check for GET requests. This would prevent bookmarking and crosslinking
• Do not know of any ways to spoof referrer field unless browser or plugin is compromised
• adding hidden unguessable form field provides additional level of security
XSRF

• Both the submit server and the grades server are protected against XSRF

• Other campus services?
  • Try creating a web page that submits a form to do something you are already authorized to do
  • If it works, and doesn't have unguessable fields, you've found a XSRF vulnerability
HTTP sessions

• HTTP is stateless
• Each connection is distinct, not associated by http to previous connections from the browser/user
• but... must... buy... from... amazon
Sessions

- Server creates a session object, and generates a random session identifier
- Each request needs to send session identifier back to server
- Sending username or credit card number in each request would be a really bad idea
Sending the session identifier

- Cookies
- URL rewriting (add it to each URL)
- Hidden form fields
Cookies

- Cookies are easiest
- Server sends a cookie to browser that says:
  - every time you send a request to www.foo.com, send this cookie
  - and the cookie expires in 2 hours
- Some browsers/users disable cookies
  - some proxies disable cookies
- You close your windows, walk away from the computer
  - still has cookies, still logged into Amazon, Fidelity, whatever...
URL rewriting and hidden form fields

- Require effort everywhere you have a URL or form in generated web pages
- that you want to have connect to the same session
- Doesn't depend on browser/user/proxy support
- If you open a new web page to a site, you get a new session
- But even if all windows closed, browser history may provide access to old session
Automatic sessions

• Generally, servers will try to maintain a session using either cookies or URL rewriting, whichever is available.