Com S 417
Software Testing

Fall 2017 – Week 9, Lecture 16

Announcements

- Lab 4 is available and due Oct. 31.
- We *will* have 5 labs.
- Exam 2 will be delayed by one week (

Topics

- Issuing a request from a junit test.
- Template languages
- Hello World in JSP
- In-container vs. Out-of-Container tests.
- Spring and alternative deployments.
- Combinatorial Testing
- Exam Schedule.

HttpClient

- Download a binary distribution from https://hc.apache.org/downloads.cgi
  Put it in your lib directory (in an application project, not the webapp project) and add it to the path (as an external jar.)
- You need four jars (minimal install):
  - httpClient
  - httpCore
  - commons-logging
  - commons-codec
- Plus httpclient-win for windows?

Basic HttpClient Usage

```
@Test
public void testGet() throws IOException {
    CloseableHttpClient httpclient = WinHttpClients.createDefault();
    String queryStr = "?name=Robert";
    HttpGet httpGet = new HttpGet("http://localhost:8080/Hello/hi"+queryStr);
    CloseableHttpResponse response1 = httpclient.execute(httpGet);

    try {
        System.out.println(response1.getStatusLine());
        HttpEntity entity1 = response1.getEntity();
        BufferedReader r = new BufferedReader(
                new InputStreamReader(entity1.getContent()));
        String line = null;
        while ((line = r.readLine()) !=null){
            System.out.println(line);
        }
        EntityUtils.consume(entity1);
    } finally {
        response1.close();
    }
}
```

POST vs. GET

- POST designed to be used with form submit.
  - designed to update server.
  - form values (coded as name value pairs) are embedded in request body (not just query string).
  - post can handle more complex data (e.g., arrays) more gracefully.
  - post can request a different page in the response.
    - i.e., form is on page1, but after submitting, the browser is looking at page2.
- GET data limited by maximum URL length of client and server.

‘Fluent’ HttpClient GET

- Requires fluent-hc jar

```
@Test
public void testFluentGet() throws ClientProtocolException, IOException{

    Content content = Request.Get("http://localhost:8080/Hello/hi")
        .execute().returnContent();
    System.out.println(content.asString());

}
```

‘Fluent’ HttpClient POST

```
@Test
public void testFluentPost() throws ClientProtocolException, IOException {
    Content content = Request.Post("http://localhost:8080/Hello/hi")
    .bodyForm(Form.form().add("name",  "FluentGuy").build())
    .execute().returnContent();
    System.out.println(content.asString());
}
```

Bottom line:

- Be aware the “gen 2” HttpClient is there and that most of its objects are accessible from fluent structures …
- Be prepared to “drop down” to gen 2 if you need some unusual/sophisticated control of HTTP tx …
- But learn and use Fluent as your primary Java HTTP tool.

POST vs. GET

8) GET sends data as part of URI while POST method sends data as HTTP content. GET requests are sent as a query string on the URL:

```
GET index.html?name1=value&name2=value HTTP/1.1
Host: java67.blogspot.com
```

POST requests are sent in the body of the HTTP request:

```
POST /index.html HTTP/1.1
Host: java67.blogspot.com
name1=value&name2=value
```

Blank Line Missing!!

from: http://www.java67.com/2014/08/difference-between-post-and-get-request.html

POST (more accurate)

```
POST /path/script.cgi HTTP/1.0
From: frog@jmarshall.com
User-Agent: HTTPTool/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 32

home=Cosby&favorite+flavor=flies
```

from: https://stackoverflow.com/questions/14551194/how-are-parameters-sent-in-an-http-post-request

JSP (Java Server Pages)

- The following slides (with white background) were prepared by S. Mitra.

JSP

Java Server Pages

Reference: http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/Servlet-Tutorial-JSP.html

A "Hello World" servlet

(from the Tomcat installation documentation)

```
public class HelloServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String docType =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">\n";
    out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>Hello</TITLE></HEAD>\n" +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1>Hello World</H1>\n" +
                "</BODY></HTML>");
  }
}
```

This is mostly Java with a little HTML mixed in

Servlets vs. JSP

- The purpose of a servlet is to create a Web page in response to a client request
- Servlets are written in Java, with a little HTML mixed in
  - The HTML is enclosed in out.println( ) statements
- JSP (Java Server Pages) is an alternate way of creating servlets
  - JSP is written as ordinary HTML, with a little Java mixed in
  - The Java is enclosed in special tags, such as <% … %>
  - The HTML is known as the template text
- JSP files must have the extension .jsp
  - JSP is *translated* into a Java servlet, which is then *compiled*
  - Servlets are run in the usual way
  - The browser or other client sees only the resultant HTML, as usual
- Tomcat knows how to handle servlets and JSP pages

JSP scripting elements

- There is more than one type of JSP "escape," depending on what you want done with the Java
- <%= ***expression*** %>
  - The ***expression*** is evaluated and the result is inserted into the HTML page
- <% ***code*** %>
  - The ***code*** is inserted into the servlet's service method
  - This construction is called a scriptlet
- <%! ***declarations*** %>
  - The ***declarations*** are inserted into the servlet *class,* not into a method

Example JSP

- <HTML>
  <BODY>
  Hello! The time is now <%= new java.util.Date() %>
  </BODY>
  </HTML>

- Notes:
  - The <%= … %> tag is used, because we are computing a *value* and inserting it into the HTML
  - The fully qualified name (java.util.Date) is used, instead of the short name (Date), because we haven't yet talked about how to do import declarations

Variables

- You can declare your own variables, as usual
- JSP provides several predefined variables
  - request : The HttpServletRequest parameter
  - response : The HttpServletResponse parameter
  - session : The HttpSession associated with the request, or null if there is none
  - out : A JspWriter (like a PrintWriter) used to send output to the client
- Example:
  - Your hostname: <%= request.getRemoteHost() %>

Scriptlets

- Scriptlets are enclosed in <% … %> tags
  - Scriptlets *do not* produce a value that is inserted directly into the HTML (as is done with <%= … %>)
  - Scriptlets are Java code that *may* write into the HTML
  - Example:

```
<% String queryData = request.getQueryString();
   out.println("Attached GET data: " + queryData); %>
```

- Scriptlets are inserted into the servlet *exactly as written,* and are not compiled until the entire servlet is compiled
  - Example:

```
<% if (Math.random() < 0.5) { %>
        Have a <B>nice</B> day!
<% } else { %>
        Have a <B>lousy</B> day!
<% } %>
```

Declarations

- Use <%! … %> for declarations to be added to your servlet class, not to any particular method
  - Caution: Servlets are multithreaded, so nonlocal variables must be handled with extreme care
  - If declared with <% … %>, variables are local and OK
  - Data can also safely be put in the request or session objects
- Example:

```
<%! private int accessCount = 0; %>
    Accesses to page since server reboot:
<%= ++accessCount %>
```

- You can use <%! … %> to declare *methods* as easily as to declare *variables*

Directives

- Directives affect the servlet class itself
- A directive has the form:

  <%@ *directive attribute*="***value***" %>

  or

  <%@ *directive attribute1*="***value1***"
  attribute2="***value2***"
  ...
  attributeN="***valueN***" %>

- The most useful directive is page, which lets you import packages
  - Example: <%@ page import="java.util.*" %>

The include directive

- The include directive inserts another file into the file being parsed
  - The included file is treated as just more JSP, hence it can include static HTML, scripting elements, actions, and directives
- Syntax: <%@ include file="***URL*** " %>
  - The ***URL*** is treated as relative to the JSP page
  - If the ***URL*** begins with a slash, it is treated as relative to the home directory of the Web server
- The include directive is especially useful for inserting things like navigation bars

```
<%@ page contentType="text html; charset = ISO-8859-1" %>


<HTML>  <HEAD>   <META HTTP-EQUIV="Content-Type"
CONTENT="text/html; charset=ISO-8859-1">
<TITLE>CS417 Hello JSP World</TITLE>
</HEAD>
<p><font color=red>Here we print what we got from original form +
from servlet1   + from servlet2</font></p>
<%  out.println(request.getParameter("original"));
     out.println(request.getAttribute("sv1_message"));
     out.println(request.getAttribute("sv2_message"));
%>
</HTML>
```

Best Practices

- Know what character set your editor uses so that you can code the charset in

```
<%@ page contentType="text html; charset = ISO-8859-1" %>
```

  correctly. It should describe the charset used to create/edit the jsp file.
- Include a valid strict-mode doc-type header
- <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

  See https://www.quirksmode.org/css/quirksmode.html and https://www.w3.org/QA/2002/04/valid-dtd-list.html
- Use UTF-8 in the generated HTML: <meta charset="UTF-8">

JSP – further exploration

- JSP details
  - https://www.tutorialspoint.com/jsp/index.htm
- JSTL (standard tag library)
  - https://www.tutorialspoint.com/jsp/jsp_standard_tag_library.htm
- EL (expression language)
  more convenient access to certain pre-defined container objects. See Implicit Objects in
  - https://www.tutorialspoint.com/jsp/jsp_expression_language.htm