E-Applications

Introduction to JSP

Michail Lampis
michail.lampis@dauphine.fr

Outline

- Server-side programming
- JSP – Servlets
- Using Tomcat
- Examples

Server-Side Programming

- So far, we have been using javascript to create dynamic pages
  - We were programming the **client** (browser)
  - There are limits to what can be done this way
    - (Persistency, Consistency, …)
- For more complicated applications, we need to also program the web server

Reminder: Client-Server Web

- Recall how the web works
  - All the data for a web site is stored on a **web server**
  - Clients (browsers) communicate with this server using the http protocol, requesting pages
  - The server returns the requested information (HTML)
- Client-side programming
  - The server returns a page that contains a **program** with instructions for the browser.

Server-Side programming

- Depending on the client's request the server may
  - Return a static HTML page (perhaps with js inside)
  - Perform some computations, return a page that is their result
    - Typical example: look up information in a local (server-side) database
- Server-side programming is about the 2nd of these cases

Client queries

- Typically, server responds to queries (questions) by the clients
  - In static cases the client just requests a URL
  - In our case the client gives some extra parameters
  - The HTTP protocol allows two main methods for doing this: GET and POST requests

GET and POST

- In both cases, the client requests a URL and additionally sends some more info
  - In GET requests this extra info is encoded in the URL as follows:

    https://www.google.fr/?q=good+pizza+in+paris
  - Rules: parameters follow a ?
  - They come in <name>=<value> pairs
  - If many pairs are given, they are separated by &

GET and POST

- In both cases, the client requests a URL and additionally sends some more info
  - In GET requests this extra info is encoded in the URL
  - In POST requests the extra info is sent separately
  - Advantage of GET: simpler, easier to debug
  - Advantage of POST: safer, allows to send non-text, more complicated data

A small example

```
<html>
<body>
<form method="get">
Name: <input type="text" name="username" /> <br />
Password: <input type="password" name="pw" /> <br />
<input type="submit" value="Go" />
</form>
</body>
</html>
```

Explanation

- The form allows the user to prepare (with the help of the browser) a GET or POST request
  - It also takes an action attribute, indicating the URL to be requested. If this is empty, the current page is used.
- When the user clicks the submit button, a GET request is sent, with the values of the inputs that have names.
  - Example:

    file:///home/mlampis/Downloads/test.html?username=a&pw=b

Responding to requests

- So far we have explained how to send simple requests to the server
- How can the server respond intelligently to such requests?
  - It will generally run a program that reads the parameters and does something with them
- What kind of program?
  - Here, there are many choices...

CGI

- Common Gateway Interface
- The classical approach to this problem
  - Write a program (in any language you like!) that takes text input representing the parameters
  - The program outputs the HTML that is the proper response
  - The web server is simply the intermediary, calling this program

Why not CGI?

- CGI is the most straightforward (and oldest) solution for client-side programming
- Its main problem is performance
  - Each time a client request arrives, the server must run a new instance of the program
  - In most operating systems this has a considerable cost (context switch)
  - → CGI does not scale well...

Server-based solutions

- Another approach is to have the server do all the work itself
- Instead of a new process, we start a new thread inside the server (faster!)
- But, the server must be programmable
  - This means that we can only use one of a “limited” number of languages.

Server-Based Solutions

- PHP
  - Basis of the LAMP architecture (Linux-Apache-MySQL-PHP)
- ASP
  - Active Server Pages (Microsoft)
- Node.js
  - Allows server-side programming in javascript
- JSP
  - Java Server Pages

JSP

- In this class we will mostly focus on a JSP solution
- General idea:
  - We write a program in Java that can handle GET/POST requests and produce HTML output
  - Such programs are called **servlets**
  - The program is stored inside the web server (container)
  - Platform we will use: Tomcat

My first JSP program

```
<html>
<body>
Hello! <br />
I can do math: 2 + 2 = <%= 2+2 %>
</body>
</html>
```

Explanation

- The new part of this program is the <%= … %> “tag”.
- This delimits a Java expression, that will be computed. Its value will be placed in the HTML instead of the expression.
  - So, upon loading the page the user will see:

    2 + 2 = 4

How does this work?

- We store this file with a jsp extension and put it in our server (details later)
- The first time a client requests the file, the server **compiles** it and produces a servlet
- The output of the servlet is then given to the client, and for any future clients the servlet is re-executed (without re-compiling, unless the code is changed).

Not javascript!

- Caution:
  - The HTML page that the user will receive does not contain any code! The Java expression has been replaced by its result.
- This is very different from programming in javascript (though superficially similar)
  - We are writing a program, not HTML
  - Think of all HTML as being inside an out.println( ) call...

JSP syntax

- Scriptlets: pieces of code delimited by <%... %>
- Example:

```
<body>
Hello World!<br/>
<%
out.println("Your IP address is " +
request.getRemoteAddr());
%>
</body>
```

JSP syntax

- Inside scriptlets you can write regular Java, also using some predefined objects (e.g. out)
- **Unlike javascript** you can mix and match scriptlets with HTML!

```
<body>
<% if (day == 1 | day == 7) { %>
    <p> Today is weekend</p>
<% } else { %>
    <p> Today is not weekend</p>
<% } %>
</body>
```

- (we assume that day is a previously declared variable...)

JSP expressions

- Instead of scriptlets, we can simply evaluate an expression with <%= … %>
  - This is essentially the same as

    <% out.println(...) %>

JSP syntax

- Another mix-match example:

```
<%for (int fontSize = 1; fontSize <= 3; fontSize++){ %>
   <font color="green" size="<%= fontSize %>">
    JSP Tutorial
   </font><br />
<%}%>
```

JSP declarations

- When the JSP is compiled into a servlet, it will turn into a class.
- All scriptlet code, as well as all HTML will become part of one of the methods of this class
  - Scriptlet variables are local...
- We can define other properties of this class, methods, or even other helper classes.

JSP declarations

- Declarations are done with <%! … %>
- Example: <%! int counter = 0; %>
- These are like **global** variables for our jsp program
  - They are also **shared** between all threads!
  - This can help us communicate between threads (clients)
  - It can also lead to problems/not scale too well...

JSP directives

- JSP directives give general commands that affect the whole program
  - Most common: importing other Java classes
- Syntax: <%@ … %>
  - Example: <%@ page import="java.util.*" %>

Standard objects

- JSP gives us some pre-defined objects to use in our program. These include:
  - The request object: contains info about request parameters
  - The session object
  - The out object: allows us to write on the HTML page directly

The request object

- Contains the parameters of the GET/POST request
  - Main method:

    String getParameter(String name)
  - Returns the value of the given (named) parameter, or null if the parameter is not set in the request.
- General class: HttpServletRequest

The session object

- Keeps track of information related to a user's visit to the site
  - Remains consistent upon repeated visits from the same client (unless time-out occurs)
- To disable sessions:

```
<%@ page session="false" %>
```

Using sessions

- Main methods:

  public Object getAttribute(String name);

  public void removeAttribute(String name);

  public void setAttribute(String name, Object value);
- Allows to read/remove/write attributes on a session object.
- How to end a session?
  - Either remove the attributes you care about
  - Or session.invalidate( ); (completely destroys session)

Getting started

- We now know enough to be able to write a small application
- But, how are we supposed to compile/run it?
- We will be using Tomcat
  - The reference implementation for JSP
- General idea:
  - You will run a local instance of Tomcat
  - You will place your code there
  - You will connect to it with a browser to test your code

Running Tomcat

- Step 1: download Tomcat 7 (or another version)

  https://tomcat.apache.org/download-70.cgi
- Step 2: unpack it in your home directory:
  - tar xvzf apache-tomcat-7.0.77.tar.gz
  - mv apache-tomcat-7.0.77 Tomcat/
- Step 3: start it
  - Tomcat/bin/startup.sh
- Check if everything works by opening a browser at http://localhost:8080

The Tomcat directory structure

- Inside your Tomcat directory you find:
  - bin: contains start/stop scripts
  - conf: configuration files (defaults should be OK)
  - webapps: places your jsp files here, with a different folder for each application
  - work: compiled servlets are placed here...

Getting started

- Recall the following program:

```
<html> <body>

Hello! <br />

I can do math: 2 + 2 = <%= 2+2 %>

</body> </html>
```

- Store it as test.jsp. Save it in the directory webapps/test/
- Open with your browser: http://localhost:8080/test/test.jsp

Getting started

- If all goes well, the file will be compiled by Tomcat automatically the first time you request the page.
  - The next time the jsp is already compiled, so everything goes faster
- You can find your IP address with /sbin/ifconfig
  - Ask a friend to check if your server works by going to http://192.168.*.*:8080/test/test.jsp (use your address instead of *)