Introduction to Web Engineering SENG2050/6050

Lecture 2 – Servlets

Server-side Web Technologies

- Serving **static** Web pages is simple using a Web Server
  1. Server receives request
  2. Maps URI to a local document
  3. Transmits local document
- The document is usually the same
  - Only changes when manually updated on the server
  - It can be “cached” by intermediate devices

Server-side Web Technologies

- It’s difficult to get dynamic behavior using static content
  - Can embed scripting languages in the document – JavaScript, VBScript, etc
  - Can embed Java Applets in the document
- Disadvantage
  - Browser is required to do the processing – what about low-power devices like mobile phones?
  - Browser must handle all the information – even if not needed or sensitive!

Server-side Web Technologies

- Serving **dynamic** pages is more involved
- Might include
  - Parsing the document to identify “special instructions” for the server
  - Running a local executable to generate the document in real-time
  - Contacting other servers to provide information used to generate the document
- The document can “change” each time
  - Intermediate “caches” must be careful

Server-side Web Technologies

Dynamic Web pages are:

- Based on data sent by the client
- Derived from data that changes frequently
- Generated with information from database and other sources

Server Side Includes

- They provide a handler which will process files before they are sent to the client.
- They allow conditional text, the inclusion of other files or programs, and the setting and printing of environment variables.
- Points for
  - Relatively simple to use
  - Only moderate performance impact
  - Run in a standard web server
- Points against
  - Severely restricted capabilities, or
  - Suffer from major security issues

Server Side Includes

“Special tags” embedded in a HTML document to

- Conditionally use fragments of the document
  - If agent = = Firefox then … else …
- Include an external HTML fragment in the document
  - Use a common header and footer across a entire Web site
- Insert the value of certain system variables
  - Insert the current date and time into a page
- Run a local executable and paste its output into the document

- http://httpd.apache.org/docs/2.2/howto/ssi.html

Common Gateway Interface – CGI

- A standard for passing information between a Web server and a local executable
- Often acts as gateways to applications or database systems

Common Gateway Interface – CGI

- Poor performance
  - Process started for each CGI request
  - If CGI program acts as a gateway to a DBMS, then a connection to the DBMS has to be opened and closed for each request
- For this reason, various improvements were proposed
  - Dedicated, persistent processes for handling requests to specific applications

Modern Server-side Web Technologies

- The more modern approach uses a ‘web application server’
  - The application server is a ‘server’ that runs applications
  - Application will run persistently in background
  - Application server will forward requests to application
  - Web application server is an application server that is also a web server
- Seen with many languages
  - C/C++, **Java**, C#/.Net, PHP, Python, Ruby
- Typically provides services to applications
  - E.g. authentication & security, request validation, database management

Servlet Containers

- A type of Java application server
- Provides a runtime environment for Java servlets, plus other handy features
  - Passes request data to a mapped servlet
  - Servlets are request handlers
  - Sends response data generated from the mapped servlet to client
  - Can also serve static web content
    - html
    - JavaScript files
    - CSS files
    - text files
    - etc.
- Popular examples include **Apache Tomcat** and Eclipse Jetty

Tomcat

- We will be running Java servlets with the Tomcat servlet container
- Please use version latest version 9 release with Java 11
  - https://tomcat.apache.org/download-90.cgi
  - **Version 10 is not compatible with course materials!**
- Implements version
  - 4.0 of the servlet spec
  - 2.3 of the JSP spec
- Runs as server, hosting many applications
  - i.e. you can use the same copy of Tomcat for *all* applications developed in this course!
- Handles HTTP requests and responses for you
- Providing them as objects for your servlets to work with

JakartaEE Application Servers

- A Servlet container is a ‘subset’ of a Java application server
- JakartaEE defines a standard for Java application servers
  - Lists a set of features a compliant server ***must*** implement
- Provides an enterprise-grade environment for Java applications
  - Notable features include:
    - Web services (RESTful and XML-based)
    - Request validation
    - Relational DBMS access
    - Messaging
    - Batch Processing
- Note: also referred to as J2EE and JavaEE

JakartaEE Application Servers

- Examples include:
  - Glassfish
  - Payara
  - Wildfly
  - OpenLiberty
  - TomEE (Tomcat, but JakartaEE Compliant)
- Most offer advanced server management capabilities
- For this course, we only need Tomcat
- Use of JakartaEE application servers covered in later courses

Java Servlet

- Java class that runs within a compatible:
  - Servlet container
  - Web application Server
- Reads the HTTP request data sent by a browser
- Generates a response
  - Generally, a web page to display
  - Can be other documents (in the context of an AJAX request maybe)
  - Can forward or redirect requests
- Sends the response back over HTTP

Java Servlet

Java Servlets provide

- Dynamic generation of documents
- Processing of other documents on the server
- Processing of other documents on other servers (through another servlet!)
- Handling insecure requests from client-side scripts
- Services independent of the Web server!

Java Servlet

- Precompiled code “inside” the servlet container
  - Shared Library on disc, or actually in memory
  - Some servers keep common CGI scripts in memory
    - http://perl.apache.org/start/index.html
  - Java Servlets are servlets written in Java, stored in the memory of the web server
- Points for
  - Much better performance
  - Better security
- Points against
  - Harder to configure (servlet container vs. web server)

Java Servlet general process

1. **Container** reads the HTTP request data sent by a client.
   - Will identify the URL, and dispatch to a matching servlet
2. **Servlet** processes the request and produces a response
   - Generally, a HTML web page to display
   - Can be other non-HTML documents
   - E.g. an XML or JSON document, even static documents!
   - Can ‘forward’ or ‘redirect’ requests
3. **Container** sends the response back over HTTP

Java Servlets

- Regardless of the application, servlets usually carry out the following routine:
  1. Read any data sent by the user
     - Capture data submitted by an HTML form.
  2. Look up any HTTP information
     - Determine the browser version, host name of client, cookies, etc.
  3. Generate the Results
     - Connect to databases, connect to legacy applications, etc.
  4. Format the Results
     - Generate HTML on-the-fly
  5. Set the Appropriate HTTP headers
     - Tell the browser the type of document being returned or set any cookies.
  6. Send the document back to the client


import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
@WebServlet(urlPatterns = {"/MyServlet"})
public class MyServlet extends HttpServlet {
@Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
//your code to deal with a POST request
}
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
//your code to deal with a GET request
}
}
Standard imports
Annotation to map to URL
Must extend 'HttpServlet'
Request handlers
(one for each verb)
Handlers have two params:
1. Request
2. Response

Java Servlet

- Import the Servlet API:
  - **import javax.servlet.*;**
  - **import javax.servlet.http.*;**
- All your servlets must extend HTTPServlet.
- HTTPServlet represents the base class for creating Servlets within the Servlet API.
- The Full Servlet API is available at:
  - http://java.sun.com/products/servlet/2.3/javadoc/index.html
- Once you have extended HTTPServlet, you must override one or both:
  - doGet(): to capture HTTP Get Requests
  - doPost(): to capture HTTP Post Requests
  - (or less commonly doDelete, doHead, doOptions, doPut, doTrace)

Java Servlet

- The doGet() and doPost() methods each take two parameters:
  - HTTPServletRequest: encapsulates all information regarding the browser request.
    - Form data, client host name, HTTP request headers.
  - HTTPServletResponse: encapsulates all information regarding the servlet response.
    - HTTP Return status, outgoing cookies, HTML response.
- If you want the same servlet to handle both GET and POST, you can have doGet call doPost or vice versa.

Java Servlet

- The HTTPResponse object has a getWriter() method.
- This method returns a java.io.PrintWriter object for writing data out to a web page for the Web Browser.

```
PrintWriter out = response.getWriter();
```

Java Servlet

- Once you have an OutputStream object, you just call the println() method to output to the browser.
- To generate HTML, you need to add two steps:
  - Tell the browser that you are sending back HTML.
  - Modify the println() statements to return valid HTML.

Example

```java
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

@WebServlet(urlPatterns = {"/MyServlet"})
public class MyServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head><title>Hello</title></head>");
        out.println("<body>");
        out.println("\t<h1>Hello World!</h1>");
        out.println("</body");
        out.println("</html>");
        out.close();
    }
}
```

Java Servlet – Architecture

A Java Servlet is a class which implements the `Servlet` interface

- `init(ServletConfig config)` – called only when the servlet is first loaded
- `destroy()` – called when the servlet is unloaded
- `service(ServletRequest req, ServletResponse res)` – called every time the servlet is accessed

Java Servlet – Architecture

`ServletConfig`

- Passes configuration from the server to the servlet
- Allows the servlet to access helper methods in the server – `config.getServletContext.`…
- `getServerInfo()` – server version info
- `log(`*`message`*`)` – write a message to the server's log file

Java Servlet – Architecture

`ServletRequest` - Represents the current request

- **`getRemoteAddr()`**, **`getRemoteHost()`** – the "client" making the request
- **`getServerName()`**, **`getServerPort()`** – the "server" processing the request
- **`getParameter(`*`name`*`)`** – the (single) value of the named parameter (e.g., a <form> input)
- **`getParameterNames()`** – list all parameters
- **`getParameterValues(`*`name`*`)`** – list all values of the named parameter (an array of **`String`**)
- **`getReader()`** – for reading uploaded files

Java Servlet – Architecture

`SerlvetResponse` - Represents the current response

- **`setContentType(`*`type`*`)`** – the MIME type of the response
- **`setContentLength(`*`length`*`)`** – the length of the response
- **`getWriter()`** – a **`PrintWriter`** for generating the response document

Java Servlet – Architecture

`HttpServletRequest`

- **`getQueryString()`** – query part of URI
- **`getRemoteUser()`** – *who* made the request
- **`getCookies()`** – get the cookies associated with this request
- **`getSession()`** – get the session associated with this request
- **`getHeaderNames()`**, **`getHeader(`*`name`*`)`** – access all HTTP headers

Java Servlet – Architecture

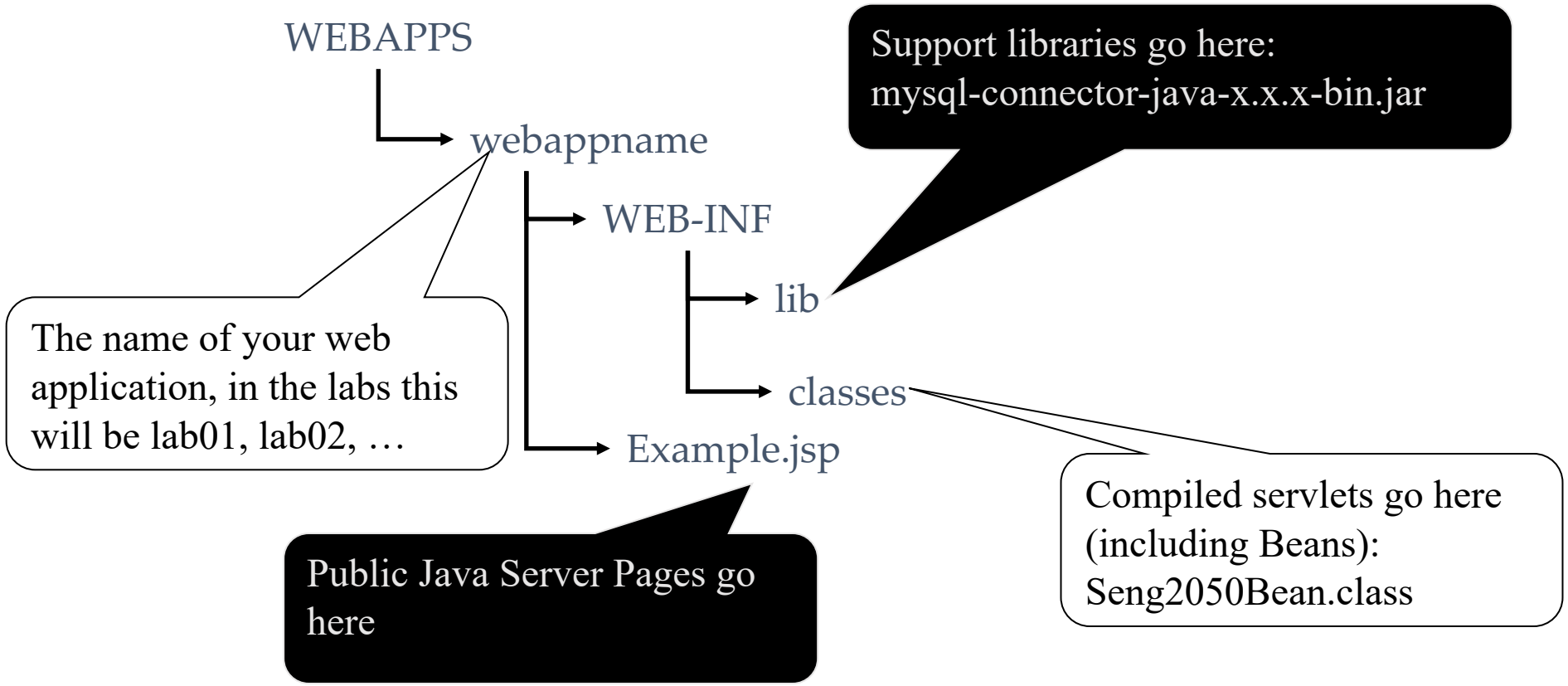
`HttpServletResponse`

- `setStatus()` – the HTTP response code
- `sendError(`*`code`*`)` – send an error status
- `sendRedirect(`*`uri`*`)` – redirect the client
- `addCookies()` – add a cookie associated with this request
- `setHeader(`*`name`*`, `*`value`*`)` – set any HTTP header

Java Packages

- Package: A group of related classes.
  - For example:
    - package coreservlets;
- In real web sites, multiple programmers may be creating multiple servlets.
- By dividing your code base into packages, it helps to modularize the code.
- A very common practice in the real world.

Java Servlets and Tomcat



Assignment 1

- Lab materials are available online, you are welcome to work on it in your own time and come to the labs for questions next week.
  - Note: configuration could be a very sticky task.
- For Assignment 1, I would suggest to work on the following this week.
  - Write static html for Page A and B.
  - Write CSS.
  - Write Javascript for client-side validation.
  - A data-entity object handling all the messages.

THE END

QUESTIONS??

THANKS!!