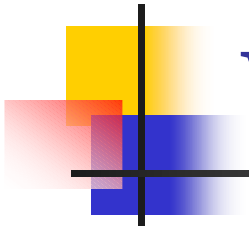
CSC309 Tutorial – Servlets

TA: Lei Jiang

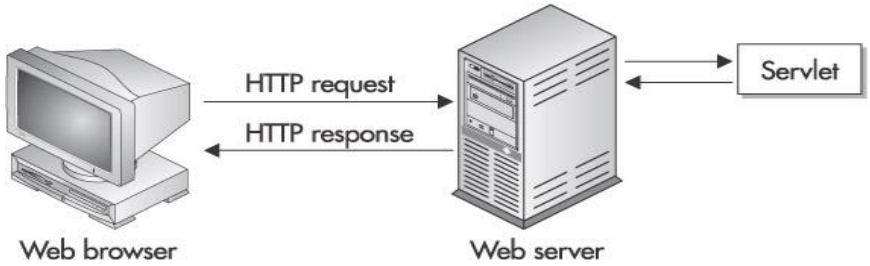
March 17, 2003



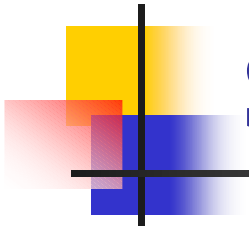
What Are Servlets?

- A servlet can be thought of as a server-side applet.
  - Is a Java class which is loaded and executed by a web server.
  - Like applets are loaded and executed by a web browser.
- A servlet accepts requests from a client (via the web server), performs some task, and returns the results.

What Are Servlets?

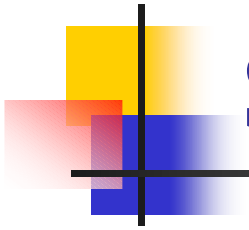


1. The client (most likely a web browser) makes a request via HTTP.
2. The web server receives the request and forwards it to the servlet.
   - If the servlet has not yet been loaded, the web server loads it into the Java Virtual Machine (JVM) and executes it.
3. The servlet receives the HTTP request and performs some type of process.
4. The servlet returns a response to the web server.
5. The web server forwards the response to the client.



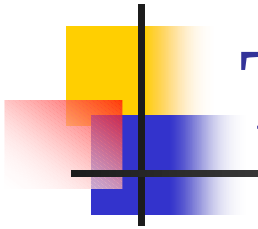
Servlets v.s. CGI

- Performance
  - CGI:
    - Need to create new processes
    - Web servers spend as much time on operating system overhead as they did serving pages.
    - very inefficient
  - Servlets:
    - Run in the same process and address space as its host Web server.
    - Little inter-process overhead
    - Much more efficient



Servlets v.s. CGI

- Persistency
  - CGI:
    - Transient:
      - each time a request is made to a CGI script, it must be loaded and executed by the web server.
      - All program initialization (such as connecting to a database) must be repeated each time a CGI script is used.
  - Servlets:
    - Persistent:
      - loaded only once by the web server and can maintain services (such as a database connection) between requests.

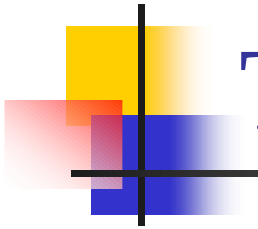


Tomcat architecture

1. A web browser makes a request to the server.

   e.g. http://some.server.com/servlet/my_servlet?param=value

2. Tomcat receives this request,
   - through its built-in web server on port 8080 (standalone), or
   - from a *web server connector*, a piece of native code attached to the web server forwarding requests to the servlet engine (embedded)
3. If a static resource, such as a file, is requested, the **DefaultServlet** will open the file, read it, and send it back to the client.
4. If a servlet is requested, the **InvokerServlet** will execute the servlet.



Tomcat architecture

- When Tomcat starts, it reads a web application descriptor file, web.xml, which sets the default values for all web applications

A sample descriptor file:

```
<!-- The mapping for the default servlet -->
<servlet-mapping>
  <servlet-name>default</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>

<!-- The mapping for the invoker servlet -->
<servlet-mapping>
  <servlet-name>invoker</servlet-name>
  <url-pattern>/servlet/*</url-pattern>
</servlet-mapping>
```

Tomcat architecture

```
<!-- The mapping for the default servlet -->
<servlet-mapping>
  <servlet-name>default</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>

<!-- The mapping for the invoker servlet -->
<servlet-mapping>
  <servlet-name>invoker</servlet-name>
  <url-pattern>/servlet/*</url-pattern>
</servlet-mapping>
```

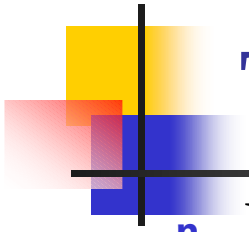
- The servlet-name elements refer to the class name of the servlet
- Tomcat attempts to match the URL:

  "http://some.server.com/servlet/my_servlet?param=value"

  with the list of mappings (longest mapping first):

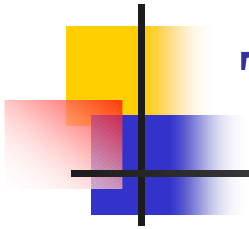
  /servlet/*

  /
- The /servlet/* pattern matches, and the servlet **InvokerServlet** is invoked
  - It ensures that the requested servlet **my_servlet** is loaded and then pass control off.



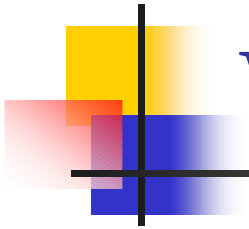
Tomcat architecture

- When **InvokerServlet** gets control, what happens next? The basic servlet lifecycle is as follows:
  1. The servlet is loaded.
     - The servlet is loaded only once;
     - multiple threads of the same servlet will handle multiple client requests.
  2. The servlet is initialized.
     - The **init()** method on the servlet is called to allow for the servlet to perform some type of initialization
  3. The servlet **service()** method is invoked.
     - The servlet performs some type of processing and returns a response.
  4. If the servlet engine is terminated, or the servlet engine decides to unload the servlet for any reason, the **destroy()** method is invoked.



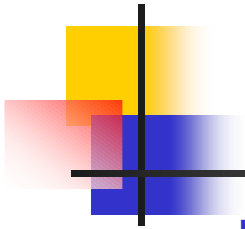
Tomcat architecture

- The Tomcat architecture has made it very easy:
  - To focus on writing the servlet pieces needed to perform work
- you don’t need to be concerned about:
  - loading and unloading a servlet,
  - handling the HTTP protocol,
  - performing chaining, invoking filters, and so on.



What Makes a Servlet a Servlet?

- To create a servlet, write a program that implements the **javax.servlet.Servlet** interface, which includes following methods
  - **init()** Called by the servlet engine (servlet container) after the servlet is loaded.
    - The init() method is passed a configuration object, which the servlet can use to read initialization parameters.
  - **service()** Allows the servlet to process a request.
  - **destroy()** Called when the servlet is being removed from the servlet engine.
  - **getServletConfig()** Returns the javax.servlet.ServletConfig object,
    - which holds initialization and startup information for the servlet.
  - **getServletInfo()** Returns information about the servlet,
    - such as the name, version, and author.
- Every servlet must implement these methods.
  - You could implement each of these methods every time you write a servlet, or,
  - You can extend two abstract base classes instead, saving your time:
    - **javax.servlet.GenericServlet:** extend the class when writing a protocol-independent servlet.
    - **javax.servlet.http.HttpServlet** extend this class when creating a servlet responding to HTTP requests.



Servlet Example – Basic Steps

- Writing a servlet for HTTP request generally requires only two basic steps:
  1. Create a new servlet class extending **javax.servlet.http.HttpServlet**
  2. Override one or both of the **doGet()** and **doPost()** methods.
     - **HttpServlet** calls these methods from the service() method depending upon the type of HTTP request.
- A servlet can also optionally override the **init()** and **destroy()** methods.
  - initialize a database connection in the init() method and close the connection in the destroy() method
- An example: **Properties** servlet
  - simply return an HTML page to the client
  - The response includes
    - information about the client,
    - any parameters that were passed,
    - and all the system properties of the server.

The **Properties** servlet

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.io.PrintWriter;

/**
 * First sample servlet. Returns any parameters and
 * lists server properties.
 */
public class Properties
       extends HttpServlet
{
   public void init(ServletConfig config)
        throws ServletException
   {
      // Initialize the servlet here.
       super.init(config);
   }


   public void doGet(HttpServletRequest req,
                     HttpServletResponse resp)
      throws ServletException, IOException
   {
      // Get an output stream
      PrintWriter out = resp.getWriter();
      // Print the HTML header to the output stream
      out.println("<html>");
```

```
      out.println("<head>");
      out.println("<title>My First Servlet</title>");
      out.println("</head>");
      out.println("<h2><center>");
      out.println("Information About You</center></h2>");
      out.println("<br>");

     // Create a table with information about the client
      out.println("<center><table border>");
      out.println("<tr>");
      out.println("<td>Method</td>");
      out.println("<td>" + req.getMethod() + "</td>");
      out.println("</tr>");

      out.println("<tr>");
      out.println("<td>User</td>");
      out.println("<td>" + req.getRemoteUser() + "</td>");
      out.println("</tr>");

      out.println("<tr>");
      out.println("<td>Client</td>");
      out.println("<td>" + req.getRemoteHost() + "</td>");
      out.println("</tr>");

      out.println("<tr>");
      out.println("<td>Protocol</td>");
      out.println("<td>" + req.getProtocol() + "</td>");
      out.println("</tr>");
```

The **Properties** servlet

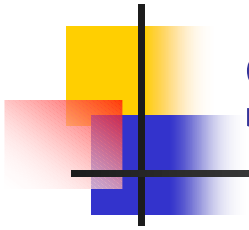
```
// Get and dump all of the request parameters
java.util.Enumeration enum =
      req.getParameterNames();
while (enum.hasMoreElements()) {
  String name = (String) enum.nextElement();
  out.println("<tr>");
  out.println("<td>Parameter '" + name +
              "'</td>");
  out.println("<td>" + req.getParameter(name) +
              "</td>");
  out.println("</tr>");
}

out.println("</table></center><br><hr><br>");


// Create a table with information about the server
out.println("<h2><center>");
out.println("Server Properties</center></h2>");
out.println("<br>");
out.println("<center><table border width=80%>");
java.util.Properties props = System.getProperties();
enum = props.propertyNames();
```

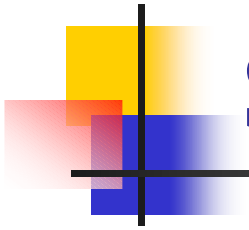
```
while (enum.hasMoreElements()) {
        String name = (String) enum.nextElement();
        out.println("<tr>");
        out.println("<td>" + name + "</td>");
        out.println("<td>" + props.getProperty(name) +
                    "</td>");
        out.println("</tr>");
      }
      out.println("</table></center>");


      // Wrap up
      out.println("</html>");
      out.flush();
   }
   public void destroy()
   {
      // Clean up here
   }
}
```



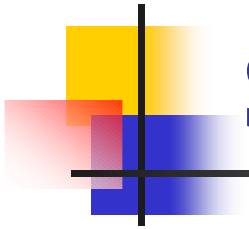
Servlet Example – Packages

- When writing servlets, almost always need to import:
  - **javax.servlet.***
    - Contains the *generic* servlet interfaces and classes,
    - Contains special servlet exceptions.
  - **javax.servlet.http.***
    - contains the *special* interfaces and classes used when writing servlets responding to HTTP requests.



Servlet Example – init()

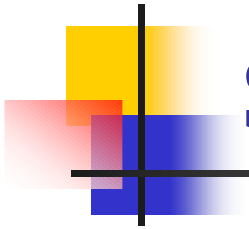
- The first method in the program is an override of the **init()** method
  - It override its superclass's init() method by calling **super.init(config)**
  - Can use the **ServletConfig** object to get initialization parameters
- required in every HTTP servlet.



Servlet Example – doXXX

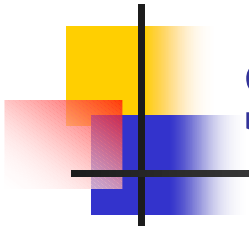
- doGet is invoked by the Web server each time a client browser requests via an HTTP GET request.
- There are other doXXX methods corresponding to other HTTP requests
  - doDelete, doOptions, doPost and doPut
- doXXX methods take two argments:
  - **ServletRequest** and **ServletResponse**
- doXXX methods must throw two exceptions:
  - **ServletException** and **IOException**
- **A typical doGet method signature:**

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
```



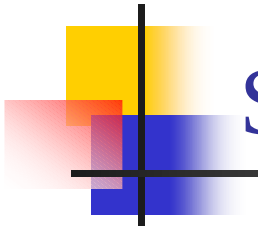
Servlet Example – ServletRequest

- **ServletRequest** class contains all the information the servlet needs regarding the current request.
- Its methods include:
  - getContentLength()
    - Returns the length, in bytes, of the request body
    - For HTTP servlets, this is the same as the value of the CGI variable CONTENT_LENGTH.
  - getContentType()
    - returns the MIME type of the body of the request
    - for HTTP servlets, this is the same as the value of the CGI variable CONTENT_TYPE.
  - getParameter(String)
    - returns the value of a request parameter
    - use it when the parameter has only one value



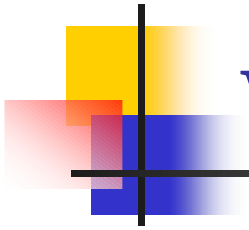
Servlet Example – ServletRequest

- getParameterNames()
  - returns an Enumeration the names of the parameters
- getParameterValues(String)
  - returns an array of the values for the given parameter
- getRemoteHost()
  - returns the fully qualified name of the client that sent the request
- getServerPort()
  - Returns the port number on which this request was received
- getServerName()
  - Returns the hostname of the server that received the request
- getMethod()
  - Returns the name of the HTTP method



Servlet Example – ServletResponse

- **ServletResponse** class allows the servlet to return its results to the Web server.
- Its methods include:
  - setContentType(String)
    - Sets the content type of the response being sent.
    - Allowable types include text/html, text/plain, and other MIME types.
  - getWriter()
    - Returns a **PrintWriter** object
    - Suitable for writing HTML pages or plain text in the response
  - **getOutputStream()**
    - **Returns a ServletOutputStream object**
    - **Suitable for writing binary data (e.g. images) in the response**
    - **Can be used to return a part of a page at a time**



Web Applications – Overview

- According to the Servlet specification:
  - A web application is a collection of servlets, HTML pages, classes, and other resources
    - That make up a complete application on a web server.
  - The web application can be bundled and run on multiple containers from multiple vendors
- The notion of a web application addresses issues like:
  - Where to place servlet classes
  - How to supply initialization parameters for servlets
  - Where to place other resources, such as HTML files and images
  - How to configure the servlet container
  - How to install supporting libraries (JAR files)

Web Applications – Directory Structure

- A web application contains a predefined directory structure
  - that defines where to put classes, libraries, and general resources.

/	• The web application root directory. • General resources, such as HTML files, can be placed here. • You are encouraged to create subdirectories for better organization.
/WEB-INF	• Contains the web application descriptor file named **web.xml**. • This directory is not visible outside of the web application. • In other words, a client cannot request to view the contents of the /WEB-INF directory (or any of its subdirectories).
/WEB-INF/classes	• The location of any classes used by the web application. • This is the root directory of any class packages.
/WEB-INF/lib	• The location of any supporting libraries (JAR files). • All JAR files found in this location will automatically be placed on the classpath. • Servlets can be packaged in JAR files and placed here as well.

Web Applications – Directory Structure

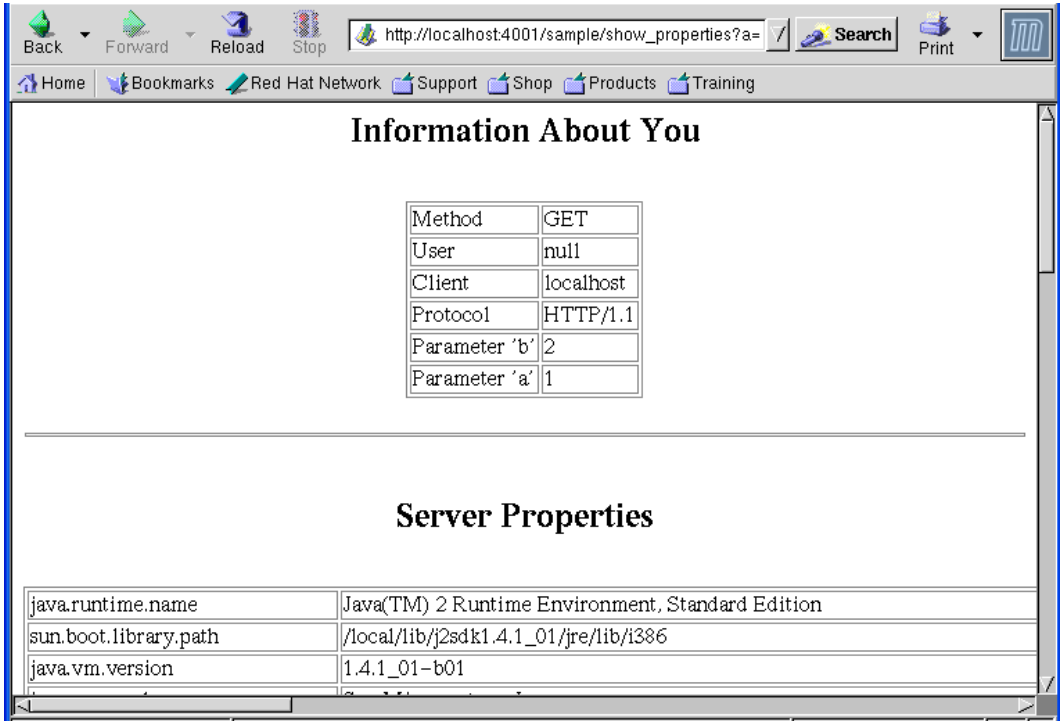
A sample directory structure

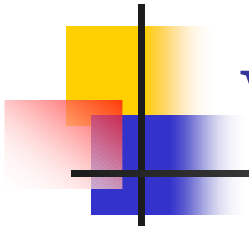
/webapps/sample/index.html
/webapps/sample/images/banner.gif
/webapps/sample/images/logo.gif
/webapps/sample/WEB-INF/web.xml
/webapps/sample/WEB-INF/classes/Properties.class
/webapps/sample/WEB-INF/lib/jaxp.jar
/webapps/sample/WEB-INF/lib/xalan.jar

```
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
  <!-- this is where we specify our servlet's java class -->
  <servlet>
    <servlet-name>Prop309</servlet-name>
    <servlet-class>Properties</servlet-class>
  </servlet>
  <!-- this is where we specify the path to access our servlet -->
  <servlet-mapping>
    <servlet-name>Prop309</servlet-name>
    <url-pattern>/prop</url-pattern>
  </servlet-mapping>
</web-app>
```

Web Applications – Directory Structure

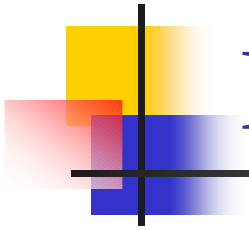
- Given the above directory structure (under the Tomcat home), we have in a web application named “sample”
  - It receives requests that begin with **/sample**
- http://localhost:4001/sample/show_properties?a=1&b=2





Web Applications – WAR Files

- **Web Application Archive**
  - Bundle all the files together in a portable collection.
  - Is simply a JAR file that has .war extension.
  - Use the jar tool to create a WAR file and place it into a running servlet container.
    - jar –cf sample.war *
    - invoke the jar command from the root directory of the web application.
  - The servlet container recognizes the WAR file and automatically create a new mapping to handle incoming requests.



Most Common Error

- When a servlet is loaded, only one instance will be created.
  - Each thread will handle a single request
  - This means multiple threads may be calling a method of the same instance at the same time
- Most common error: declare a class field and use it within service(), doGet() or doPost() methods.
  - Without synchronizing access, it will cause problems
- In general, always avoid declaring class fields within servlets
  - unless they are to be used as read-only constants.