Java Servlets

CS 4640
Programming Languages for Web Applications

[Based in part on SWE432 and SWE632 materials by Jeff Offutt]
[Robert W. Sebesta, “Programming the World Wide Web”]
Web Applications

• A web application uses enabling technologies to
  • Make web site contents dynamic
  • Allow users of the system to implement business logic on the server

• Web applications allow users to affect state on the server

An enabling technology makes web pages interactive and responsive to user inputs
Server Side Processing

UI implemented in a browser

Client

Web server

Container engine

Program components

Server

HTTP Request

data

HTML

HTTP Response

Apache
Tomcat

review
What are Servlets?

• Servlets are small Java classes that
  • Process an HTTP request
  • Return an HTTP response

• Servlet container or engine
  • Connects to network
  • Catches requests
  • Produces responses
  • Creates object instances of servlet classes
  • Hands requests to the appropriate object

• Programmers use a servlet API to write servlet classes
Servlets vs. Java Applications

- Servlets do not have a `main()`
  - The `main()` is in the server (for example, Tomcat)
  - Entry point to servlet is via call to a method (`doGet()` or `doPost()`)
- Servlet interaction with end user is indirect via request / response object APIs
  - Actual HTTP request / response processing is handled by the server
- Servlet output is usually HTML
Servlet Container (or Engine)

- Servlet container is a **plug-in** for handling Java servlets
- A servlet container has **five** jobs:
  1. Creates servlet **instance**
  2. Calls **init()**
  3. Calls **service()** whenever a request is made
     - **service()** calls a method written by a programmer to handle the request
     - **doGet()** to handle GET requests, **doPost()** to handle POST requests
     - More on this later …
  4. Calls **destroy()** before killing servlet
  5. **Destroys** instance

- A mini operating system
Servlet Container (2)

When a request comes to a servlet, the servlet container does one of two things:

1. If there is an active object for the servlet, the container creates a Java thread to handle the request

2. If there is no active object for the servlet, the container instantiates a new object of that class, and the object handles the request
Servlet Container (3)

A servlet instance runs until the container decides to destroy it:

• When is not specified by the servlet rules
• Most servlet containers destroy the object \( N \) minutes after the last request
• \( N \) defaults to 15 or 30 – can be set by the system administrator
• Container can also be configured to never destroy a servlet object
  • This is sometimes called production mode
Servlet Container (4)

• What if the same servlet gets multiple requests?

• More than one execution thread may be running at the same time, using the same memory

![Diagram of Servlet Container](image-url)
Servlet Container (5)

- By default, there is only one instance of a servlet class per servlet definition in the servlet container.

- **Distributable**: If the application is distributable, there is one instance of a servlet class per virtual machine.
  - Sometimes each VM is on a different computer in the cluster.
  - Sometimes multiple VMs are on one computer.
Servlet Object Thread Lifecycle

UML State Diagram

Does not exist

Unavailable

Destroyed

Instantiated

Initialized and/or ready for requests

Service

- initialization
- HTTP requests from clients
- end of service thread
- timeout or a container shutdown
- temporary or permanent failure
- back to service if temporarily unavailable (optional)
- initialization failed

- instantiation based on a request or at container startup
- release reference
Servlet API

- `javax.servlet` – primarily containers
- `javax.servlet.http` – methods to service requests

<table>
<thead>
<tr>
<th>GenericServlet</th>
<th>HttpServlet</th>
<th>MyServlet</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Abstract class *(not all methods are implemented)*

- *All methods are implemented*
- *Servlet written by a programmer*
Generic Servlet and HTTP Servlet

Servlets can have five methods:

1. `init()` – called when servlet starts
2. `service()` – called to process requests
3. `destroy()` – called before servlet process ends
4. `getServletConfig()` – servlet can access information about servlet container
5. `getServletInfo()` – servlet container can access info about servlet
Generic Servlet and HTTP Servlet (2)

• These methods are defined by the library classes GenericServlet and HttpServlet

• We write servlets by extending (inheriting from) them

• GenericServlet does not implement service() (it is abstract)

• HttpServlet extends GenericServlet with:

  service (HttpServletRequest req, HttpServletResponse res)

  throws ServletException, IOException
1. $\text{init}()$

- Read \textit{configuration} data
- Read \textit{initialization} parameters (\texttt{javax.servlet.ServletConfig})
- \textbf{Initialize} services:
  - Database driver
  - Connection pool
  - Logging service
- Seldom used in \textit{simple} applications
2. service( )

- The **entry point** for the servlet – this is the method that is called from the servlet container

- Called **after** the initialization (**init()**)  

- Primary purpose is to **decide** what type of request is coming in and then **call** the appropriate method
  - **doGet()**
  - **doPost()**
Types of HTTP Requests

- GET
- POST
- HEAD
- OPTIONS
- DELETE
- PUT
- TRACE

\[
\begin{align*}
\text{doGet}() & \\
\text{doPost}() & \\
\text{doHead}() & \\
\text{doOptions}() & \\
\text{doDelete}() & \\
\text{doPut}() & \\
\text{doTrace}() & \\
\end{align*}
\]

same signatures as `service()`
Types of HTTP Requests (2)

• **HttpServlet** implements these methods as “stubs” that print error messages

```java
    doGet() {
        print("Error HTTP 405, doGet() not implemented");
    }
```

• Programmers implement services by **overriding** these methods
  • usually `doGet()` and `dopost()`
3. Destroy() 

• Called by container before the servlet instance is killed

• The threads from the service() method are given time to terminate before destroy() is called

• Can be used to clean up the state of the servlet:
  • Un-registering a database driver
  • Closing a connection pool
  • Informing another application the servlet is stopping
  • Saving state from the servlet
4. `getServletConfig()`

- Returns a `ServletConfig` object, which stores information about the servlet’s configuration.

- The `ServletConfig` object was passed into `init()` by the servlet container.
5. `getServletInfo()`

- Returns a `String` object that stores information about the servlet:
  - Author
  - Creation date
  - Description
  - Usage
  - ...

- This string should be formatted for **human readability**
Simple Servlet Example

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

public class Hello extends HttpServlet {
    public void doGet (HttpServletRequest req,
                        HttpServletResponse res)
                        throws servletException, IOException {
        res.setContentType ("text/html; charset=UTF-8");
        PrintWriter out = res.getWriter ();
        out.println ("<HTML>");
    }
```
Simple Servlet Example (2)

```java
out.println ("<HEAD>");
out.println ("<TITLE>Servlet example</TITLE>");
out.println ("</HEAD>");
out.println ("<BODY>");
out.println ("<P>My first servlet.</P>");
out.println ("</BODY>");
out.println ("</HTML>");
out.close ();
}
// end doGet()

} // end Hello
Servlet Parameters – requests

Parameters are conveniently stored in objects

- **String** `req.getParameter(String KEY)`
  - Returns *value* of field with the name = KEY
  - Names are defined in HTML, and values supplied by the users

- **String[]** `req.getParameterValues(String KEY)`
  - Returns *all* values of KEY
  - For example checkboxes

- **Enumeration** `req.getParameterNames()`
  - Returns an *Enumeration* object with a list of all parameter names

- **String** `req.getQueryString()`
  - Returns the *entire query* string
Servlet Parameters – Transmission

• Parameter data is the Web analog of arguments in a method call:
  
  ```java
  System.out.println("aString");
  ```
  
  http://www.example.com/servlet/PrintThis?arg=aString

• Query string syntax and semantics
  
  • Multiple parameters are separated by ‘&’
    
    http://www.example.com/servlet/PrintThis?color=red&arg=aString

  • Order of parameters does not matter
    
    http://www.example.com/servlet/PrintThis?arg=aString&color=red

  • All parameter values are strings
    
    http://www.example.com/servlet/PrintThis?arg=&age=39

Empty string
Servlet Parameters – Creation

• HTML forms generate *query strings* when submitted

• Parameter names are specified as the value of the *name* attributes in the form controls
  
  <input type="text" name="username" size="35" />

• Parameter *values* depend on control type

  <input type="checkbox" name="daysFree" value="Mon" />

  Value sent to the server:
  daysFree=Mon
## Servlet Parameters – Creation (2)

<table>
<thead>
<tr>
<th>Controls</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>input/text</td>
<td>Text that the user has entered into the control field when the form is submitted</td>
</tr>
<tr>
<td>input/password</td>
<td></td>
</tr>
<tr>
<td>textarea</td>
<td></td>
</tr>
<tr>
<td>input/checkbox</td>
<td>String assigned to the value attribute in the HTML tag</td>
</tr>
<tr>
<td>input/radio</td>
<td>The control must be selected or clicked for the parameter to be sent</td>
</tr>
<tr>
<td>input/submit</td>
<td></td>
</tr>
<tr>
<td>input/image</td>
<td></td>
</tr>
<tr>
<td>button/submit</td>
<td></td>
</tr>
<tr>
<td>input/hidden</td>
<td>String assigned to the value attribute</td>
</tr>
<tr>
<td></td>
<td>Not rendered on the client</td>
</tr>
<tr>
<td>select</td>
<td>String assigned to the value attribute of the selected options, or content of any selected option for which the value is not defined</td>
</tr>
</tbody>
</table>
Servlet Parameters – Creation (3)

Enter your name: <input type="text" name="username" size="35" />
<p>Check all the days that you are free:
<label>
<input type="checkbox" name="daysFree" value="Mon" />
Mon
<input type="checkbox" name="daysFree" value="Tue" />
Tue
<input type="checkbox" name="daysFree" value="Wed" />
Wed
<input type="checkbox" name="daysFree" value="Thur" />
Thur
<input type="checkbox" name="daysFree" value="Fri" />
Fri
</label>
</p>

username=George&daysFree=Mon&daysFree=Wed&daysFree=Thur
Servlet Parameters – Re-Transmission

• Most browsers give a warning before submitting POST data for the second time
  • Avoid duplicate submissions and updates
  • Avoid duplicate purchases
• Users should be very careful before overriding this hesitation
• However ... how many users understand this message?
Servlet Output – responses

Standard output is sent directly back to the client browser

• res.setContentType (String type)
  • “text/html; charset=‘UTF-8’” is an HTML page with robust encoding

• PrintWriter res.getWriter()
  • Use print() and println() to write HTML to browser
Servlet Performance

- Some servlets will run a lot

- Servlets run as *lightweight threads*, so are fast

- The *network speeds* usually dominate, but two things can add speed:
  - avoid *concatenation* ("+")
  - *out.flush()* – Sends current output to user’s screen while servlet continues processing
GET and POST Requests

• An HTTP GET request is generated when the URL is entered directly
  
  `doGet()` is called from `service()`

• An HTML form can generate either a GET or a POST request
  
  “... Method=POST” or “... Method=GET”

• GET requests put form data on the URL as parameters
  
  `http://www ... /RunForm?NAME=George&TITLE=prof`

• The length of GET parameters is limited by some browsers (usually 1024 bytes)

• POST requests put form data in body of request

• POST requests can be arbitrarily long
GET and POST Requests (2)

• Books say:
  • Use GET to retrieve data
  • Use POST to change state on server (update file or DB)
  • Use POST when there are a lot of data items

• This is a little ambiguous and incomplete …

• Prof’s suggestion:
  • Use POST when sending data to server
  • Use GET when no data is sent

• GET is also useful when the entire request needs to be bookmarked
  • Google maps
GET and POST Requests (3)

• If a servlet is primarily based on processing data and it uses POST, **good engineering** says to implement a simple `doGet()` method as a filler:

```html
... <body>  
  <center>A Title ...</center> 
  <hr /> 
  
  <p> 
    You should run this from 
    <a href="http://... .html"> http://... .html</a> 
  </p> 
</body>
```
Writing to Files from Servlets

• File must be in a (publicly) writeable directory:
  /apache-tomcat/cs4640/WEB-INF/data/

• Open a file, write to it, and close it:
  ```java
  FileWriter outfile = new FileWriter("/apache-tomcat/cs4640/WEB-INF/data/info-file.txt");
  outfile.write ( ... the data to save ...);
  outfile.close ();
  ```

• Open a file in append mode:
  ```java
  FileWriter outfile = new FileWriter("/apache-tomcat/cs4640/WEB-INF/data/info-file.txt", true);
  ```

• Remember Unix / Windows path differences !!
  • “info-file” does NOT equal “INFO-FILE” !!

• When you use a shared server/a shared directory, include your user name/account as part of the file name !!!!
  • Imagine, what happens if everyone names his/her file as “info-file.txt”
Redirecting to Another URL from Servlets

• Servlets usually generate an HTML file as a response, but sometimes you may want to send the client to a different servlet

  • `res.sendRedirect(“http://another-web-component/...”);`

  • Do not need to set content type (`setContent_type()`)  

• The client will be “sent” to the specified URL  
  • Server tells the client to generate another request to the new URL  
  • Browser then repeats request to the new URL  
  • Invisible to users … but both requests logged in history

This is a control mechanism that does not exist in traditional software
Redirect and Forward

```java
public class login extends HttpServlet {
    ...
    public void doGet(...) {
        ...
        response.sendRedirect(URL1);
    }
}
```

```java
public class login extends HttpServlet {
    ...
    public void doGet(...) {
        ...
        getServletContext(URL1).getResponseDispatcher().forward(request, response);
    }
}
```

Redirect control connection

Forward control connection
Sending Mail Messages from Servlets

- Import mail utilities:

```java
import java.util.Properties;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.AddressException;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import javax.mail.PasswordAuthentication;
```
Sending Mail Messages (2)

• Setup mail utilities and authenticate an email account

```java
Properties props = new Properties();
props.put("mail.smtp.auth", "true");
props.put("mail.smtp.starttls.enable", "true");
props.put("mail.smtp.host", "smtp.gmail.com");
props.put("mail.smtp.port", "587");

Session session = Session.getInstance(props,
    new javax.mail.Authenticator() {
        protected PasswordAuthentication getPasswordAuthentication() {
            return new PasswordAuthentication(username, password);
        }
    });
```

Assuming an email will be sent from someone@gmail.com, username=someone and password=somepassword
Sending Mail Messages (3)

• Setup mail header and message:

```java
Message message = new MimeMessage(session);
message.setFrom(new InternetAddress(from_email));
message.setRecipients(Message.RecipientType.TO, InternetAddress.parse(from_email));
// message.addRecipient(Message.RecipientType.TO, new InternetAddress(to_email));
// message.addRecipient(Message.RecipientType.TO, new InternetAddress(to_email2));
message.addRecipient(Message.RecipientType.TO, new InternetAddress(email));
message.setSubject("CS4501 (" + semester + ") : servlet example ");
message.setText("Message to be sent");
```

• Send message:

```java
Transport.send(message);
```

where from_email=someone@gmail.com and to_email=someoneelse@virginia.edu
Deployment Testing

• Development and deployment computers often differ

• Web apps must be tested on final deployment platform
  • Must test just as real users use it

• Issues to check for:
  • Different platforms (DOS / Unix / Linux / Mac …)
    • File names and path names (local/nonlocal, DOS/Unix)
    • Upper case dependencies
  • Incomplete deployment
  • Compiler and runtime system version
  • Permissions (data and DB)
Servlet Summary

• Servlets are very powerful programming tools for developing robust, large, and reliable web applications

• The container engine and the servlet library insulate programmers from a lot of detailed communication issues

• The separation of the client’s UI from the back-end software is very powerful, but makes debugging hard

• Lots of development tools and advanced development frameworks for servlets