

Java Servlets

Building Servlet Web Applications

Instructor: Rick Palmer, SCWCD

rick@online-ettraining.com

Topics Covered

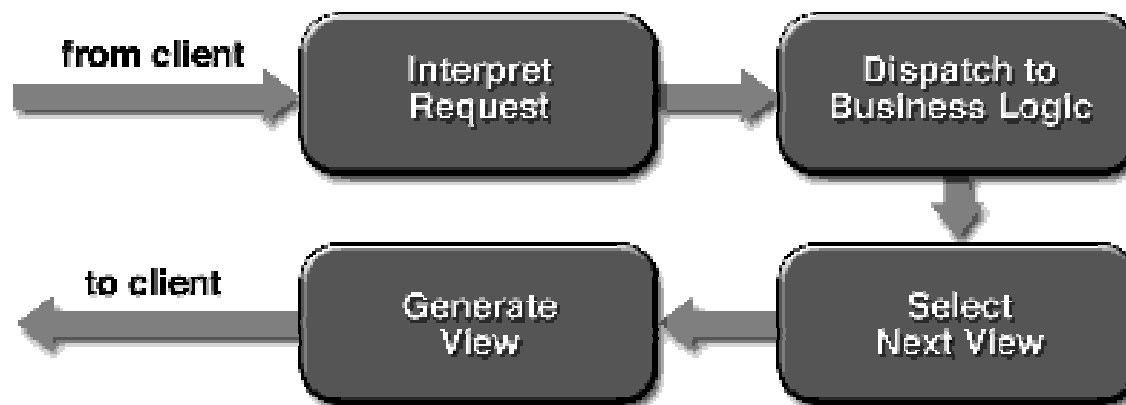
- ❑ Java Servlet introduction
- ❑ Handling requests from a web browser
- ❑ Sending a dynamic response to a web browser
- ❑ Building, packaging, and deploying a servlet-based web application

Servlet Introduction

- ❑ Java programs that extend a web server
 - ⇒ Dynamic content (HTML embedded in code)
 - ⇒ Access to the full Java API
- ❑ Portable - will run on any J2EE-compliant server
- ❑ Java Servlet 2.4 API is the most current
 - ⇒ Supported by Tomcat and all major web containers

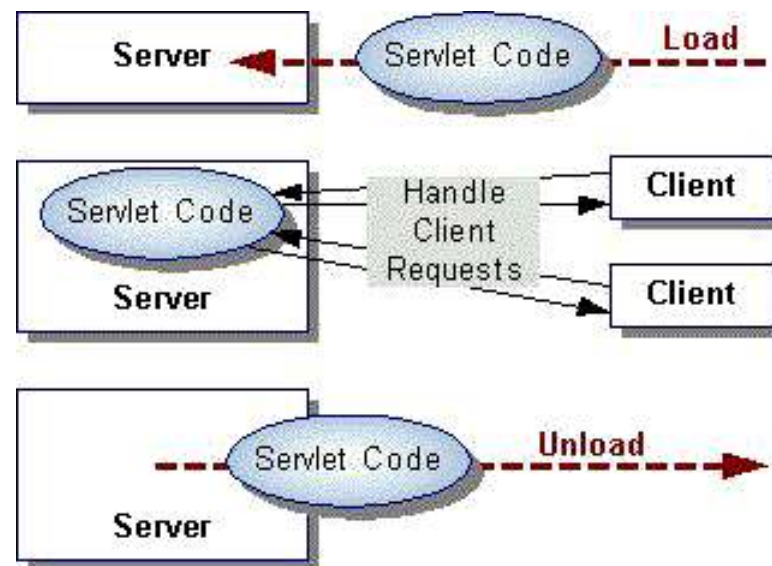
Servlet Responsibilities

- ❑ Handles web server requests (HTTP)
 - ⇒ GET (URL entered in the browser)
 - ⇒ POST (HTML login page submittal)
- ❑ Performs or delegates server-side processing
 - ⇒ Business logic or calculations
 - ⇒ Database queries/updates using JDBC or an EJB
- ❑ Sends web server responses (HTML, XML, etc) or forwards request to a JSP page to generate the view.



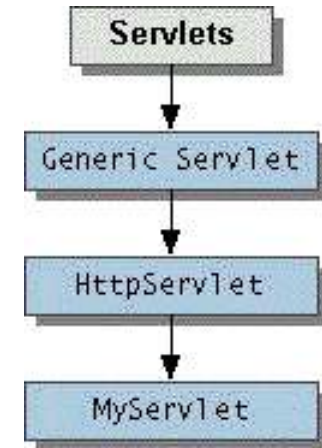
Servlet Life Cycle

- ❑ Web server loads and initializes the servlet only once.
- ❑ Loaded servlet handles each client request on a separate thread.
- ❑ Web server removes the servlet (garbage collection) during server shutdown, or after a specified period of inactivity.



Servlet Class Hierarchy

- ❑ Servlet interface
- ❑ GenericServlet base class
 - ⇒ service() method receives all requests
- ❑ HttpServlet extends GenericServlet
 - ⇒ for HTTP-specific requests/responses
 - ⇒ Determines request type and sends to doGet(), doPost()
- ❑ MyServlet (which you create) extends HttpServlet
 - ⇒ Provides the specific implementation of your servlet.



HttpServletRequest

- ❑ Represents the HTTP request from the client browser or stand-alone application.
- ❑ Exposes request parameters in name-value pair format
 - ⇒ `getParameter(String)`; – returns the String value of the named request parameter.
- ❑ Consider the following URL and servlet code:

```
http://localhost/MyApp/servlet/MyServlet?user=Pablo  
String strUser = request.getParameter("user");
```

⇒ The variable `strUser` now contains the value "Pablo".

HttpServletResponse

- Represents the server's HTTP response

- Main response methods:

- ⇒ `setContentType (String) ;` – “text/plain” by default. Change to “text/html” for HTML.

- ⇒ `getWriter () ;` – for character-based response (HTML, XML, plain text, etc)

```
PrintWriter out = response.getWriter();  
out.println("<P>Welcome to my page.</P>");  
out.flush();
```

- ⇒ `getOutputStream () ;` – for binary response

ServletContext

- ❑ Acts as a global object repository for each web application.
 - ⇒ All servlets and JSP pages within a web application have access to the same ServletContext.
- ❑ Provides information about the web server.
- ❑ Main ServletContext methods:
 - ⇒ `ServletContext ctx = getServletContext();`
 - ⇒ `ctx.setAttribute(String, Object);` – save an object
 - ⇒ `ctx.getAttribute(String);` – reference a saved object
 - ⇒ `ctx.getRequestDispatcher();` – used for redirecting requests to other servlets or JSP pages.

Cookies

- ❑ Key-value pairs stored on client browser
- ❑ Transferred in the HTTP request and response headers
- ❑ Main methods
 - ⇒ `request.getCookies()` ; – returns an array of Cookies
 - ⇒ `response.addCookie(Cookie)` ;
 - ⇒ `Cookie.getValue()` ;
 - ⇒ `Cookie.setValue(String)` ;
- ❑ Always set cookie values first, and *then* send the servlet's response output.
 - ⇒ Servlets are *not* buffered
 - ⇒ Cookies live in the response header, which must be created first.

HTTPSession

- Represents a single continuous interaction with a client (browser or stand-alone application).

 - ⇒ Unique ID is passed between client and server.

 - ⇒ ID is stored in Cookie, or passed with the URL.

 - ⇒ User's state (data) is associated with the ID.

- Main methods

 - ⇒ `HttpSession session = request.getSession();`

 - ⇒ `session.getAttribute(String);`

 - ⇒ `session.setAttribute(String, Object);`

 - ⇒ `session.isNew();`

 - ⇒ `session.invalidate();`

Servlet Example

```
import java.io.*;                //libraries for input/output
import javax.servlet.*;          //GenericServlet
import javax.servlet.http.*;     //HttpServlet

public class MyServlet extends HttpServlet {

    public void doGet(HttpServletRequest request,
                       HttpServletResponse response)
        throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML><BODY>");
        out.println("<H2>Welcome to my servlet page.</H2>");
        out.println("<P>Time: "+new java.util.Date()+"</P>");
        out.println("</BODY></HTML>");

    }
}
```

Compiling a Servlet

- ❑ Not necessary if using Eclipse or other IDE
- ❑ Configure the system
 - ⇒ Add `C:\<tomcat>\common\lib\servlet-api.jar` to the **CLASSPATH**.
 - ⇒ Add `<java home>\bin` to the **Path**, so the system can see the `javac.exe` **compiler**.
- ❑ Compile the servlet
 - ⇒ Run `javac MyServlet.java` from the command line.
 - ⇒ `MyServlet.java` **compiles to** `MyServlet.class`.
 - ⇒ **Class file should be located in the web application's classes folder:**
 - `C:\<tomcat>\webapps\MyApp\WEB-INF\classes`

Assigning Names to Servlets

- ❑ Consider a Servlet residing in folder
`webapps/MyApp/WEB-INF/classes/LoginServlet.class`
- ❑ Assign a name to the servlet in `web.xml`

```
<servlet>  
    <servlet-name>Login</servlet-name>  
    <servlet-class>LoginServlet</servlet-class>  
</servlet>
```
- ❑ Run the Servlet using the following URL:
`//localhost/MyApp/servlet/Login`
- ❑ Tomcat internally maps all URLs starting with `/servlet/*` to the `/WEB-INF/classes` folder.

Mapping Custom URLs to Servlets

- Map a servlet to a custom URL

```
<servlet-mapping>
```

```
    <servlet-name>Login</servlet-name>
```

```
    <url-pattern>/AccountLogin</url-pattern>
```

```
</servlet-mapping>
```

- Run the Servlet using the following URL:

```
//localhost/MyApp/AccountLogin
```

- Mapped URLs also look better and aid in security.

- Note: `<servlet>` element must come before

```
<servlet-mapping> element
```

Packaging and Deploying a Servlet

- ❑ Package the web app using

```
jar cvf MyAppName.war *
```

- ❑ Deploy (copy) war file to Tomcat's webapps folder.

- ❑ Start Tomcat, or restart it if already running.

⇒ Tomcat will automatically expand the war file into a subfolder with the same name as the war file.

⇒ If redeploying, make sure to delete the previously expanded subfolder before restarting Tomcat.

- ❑ Access servlet from the browser.

```
http://localhost/MyApp/servlet/MyServlet
```

Thread-safe Servlets

- ❑ A class or method is only thread-safe if its variables *cannot* be modified by other threads.
- ❑ Web servers creates a new thread for each request (each request gets its own “copy” of the servlet’s `service` method).
- ❑ Variables declared outside the `service` method will be accessible to all threads (much like static variables, though not declared as `static`).
- ❑ Variables declared local to the `service` method (or `doGet/doPost`), will be thread-safe.

Servlet Pros/Cons

□ Pros

- ⇒ Full access to all Java APIs
- ⇒ Full control of output (text, binary, etc)
- ⇒ Great as a web application "controller".

□ Cons

- ⇒ Programmatic HTML output is hard to maintain:
`out.println("<html>...");`
- ⇒ Requires manual compilation (unless using Eclipse)
- ⇒ Low-level; harder to work with (similar to ISAPI DLL)
- ⇒ Easier to use JSP for building HTML display output.