

Lecture 9: Java Servlet and JSP

Wendy Liu
CSC309F – Fall 2007

1

Outline

- HTTP Servlet Basics
- Servlet Lifecycle
- Request and Response
- Session Management
- JavaServer Pages (JSP)

2

HTTP Servlet Basics

Current API 2.5:
<https://glassfish.dev.java.net/nonav/javace5/api/index.html>

3

Servlet API

- javax.servlet
 - Support generic, protocol-independent servlets
 - Servlet (interface)
 - GenericServlet (class)
 - service()
 - ServletRequest and ServletResponse
 - Provide access to generic server requests and responses
 - javax.servlet.http
 - Extended to add HTTP-specific functionality
 - HttpServlet (extends GenericServlet)
 - doGet()
 - doPost()
 - HttpServletRequest and HttpServletResponse
 - Provide access to HTTP requests and responses

4

User-defined Servlets

- Inherit from HttpServlet
- Override doGet() and doPost()
 - To handle GET and POST requests
- Have no main() method

5

doGet() and doPost()

```
protected void doGet(  
    HttpServletRequest req,  
    HttpServletResponse resp)  
  
protected void doPost(  
    HttpServletRequest req,  
    HttpServletResponse resp)
```

6

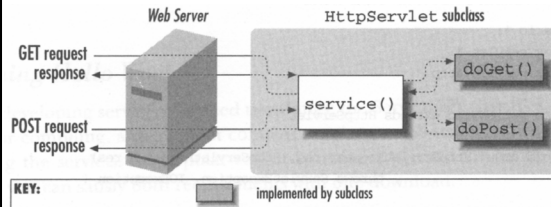
Hello World

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

public class HelloWorld extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<html>");
        out.println("<head><title>hello world</title></head>");
        out.println("<body>");
        out.println("<big>hello world</big>");
        out.println("</body></html>");
    }
    public void doPost(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        doGet(req, res);
    }
}
```

7

Handling GET and POST Request



Adapted from "Java Servlet Programming", Jason Hunter, 2001

8

End-to-end Process

- Client
 - Makes a request to a servlet
- Web Server
 - Receives the request
 - Identifies the request as a servlet request
 - Passes the request to the servlet container
- Servlet Container
 - Locates the servlet
 - Passes the request to the servlet
- Servlet
 - Executes in the current thread
 - The servlet can store/retrieve objects from the container
 - Output is sent back to the requesting browser via the web server
 - Servlet continues to be available in the servlet container

9

Servlet Lifecycle

10

Servlet Container

- Provide web server with servlet support
 - Execute and manage servlets
 - E.g., Tomcat
- Must conform to the following lifecycle contract
 - Create and initialize the servlet
 - Handle zero or more service calls from clients
 - Destroy the servlet and then garbage collect it
- Three types
 - Standalone container
 - Add-on container
 - Embeddable container
- Usually execute all servlets in a single JVM

11

Loading Servlet

- Server calls servlet's `init()` method
 - After the server constructs the servlet instance and before the servlet handles any requests
- `init()`
 - Servlet initialization is defined
 - May be called ...
 - When the server starts
 - When the servlet is first requested, just before the `service()` method is invoked
 - At the request of the server administrator

12

Removing Servlet

- Server calls the `destroy()` method
 - After the servlet has been taken out of service and all pending requests to the servlet have completed or timed out
- `destroy()`
 - Resources acquired should be freed up
 - A chance to write out its unsaved cached info
 - Last step before being garbage collected

13

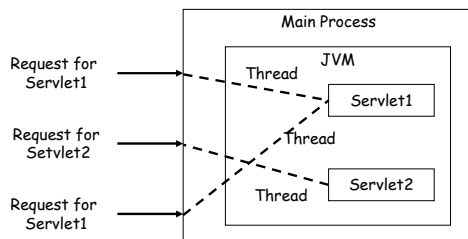
Servlet Instance Persistence

- Servlets persist between requests as object instances
- When served it loaded, the server creates a single instance
- The single instance handles every request made of the servlet
- Improves performance in three ways
 - Keeps memory footprint small
 - Eliminates the object creation overhead
 - Enables persistence
 - May have already loaded required resources

14

Servlet Thread Model

Servlet-Based Web Server



15

A Simple Counter

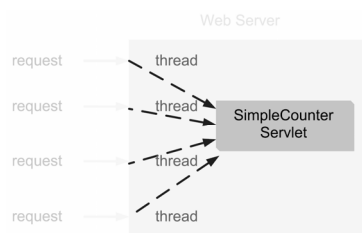
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SimpleCounter extends HttpServlet {
    int count = 0;

    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        count++;
        out.println("Since loading, this servlet has been accessed " +
            count + " times.");
    }
}
```

16

Multi-threading



- Each client request is another thread that calls the servlet via the `service()`, `doGet()`, and `doPost()` methods

17

Thread Safety

- Non-local variables are not thread-safe
 - Variables declared within a class but outside any specific method
 - May cause data corruption and inconsistencies
 - In the example
 - `count ++` // thread 1
 - `count ++` // thread 2
 - `out.println` // thread 1
 - `out.println` // thread 2
- Local variables are thread-safe
 - Variables declared within a method

18

Synchronization

- Place code within a synchronization block
 - Guaranteed not to be executed concurrently by another thread
- Before any thread begins to execute synchronized code
 - A **monitor** (lock) must be obtained on a specified object instance
 - If one thread already has the monitor, all other threads must wait
- Use only when necessary
 - Reduced parallelism
 - Increased overhead to obtain the monitor

19

Simple Counter Has Four Options

- Option 1

```
public synchronized void doGet(HttpServletRequest req, HttpServletResponse res)
```
- Option 2

```
PrintWriter out = res.getWriter();
synchronized(this) {
    count++;
    out.println("Since loading, this servlet has been accessed " + count + " times.");
}
```
- Option 3 (the best)

```
PrintWriter out = res.getWriter();
int local_count;
synchronized(this) {
    local_count = ++count;
}
out.println("Since loading, this servlet has been accessed " + count + " times.");
```
- Option 4
 - Don't care; I can live with the inconsistency

20

Request and Response

<https://glassfish.dev.java.net/nonav/jvace5/api/index.html>

21

HttpServletRequest

- Encapsulate all information from the client request
 - HTTP request header and request body
- Methods to retrieve data
 - Inherited from ServletRequest
 - `getParameter()`
 - `getParameterNames()`
 - `getParameterValues()`
 - `getInputStream()`
 - `getReader()`

22

HttpServletResponse

- Encapsulate all data to be returned to client
 - HTTP response header and response body (optional)
- Set HTTP response header
 - Primitive manipulation
 - `setStatus()`, `setHeader()`, `addHeader()`
 - Convenience methods
 - `setContentType()`, `sendRedirect()`, `sendError()`
- Set HTTP response Body
 - Obtain a `PrintWriter` or `ServletOutputStream` to return data to the client
 - `getWriter()`, `getOutputStream()`

23

GET vs. POST

- GET
 - All form parameters are embedded in the URL
 - If you reload, or bookmark and return, the query will get executed a second time with the same parameters
 - Bad if page is a credit card order confirmation – 2 charges!
 - Use GET to obtain info
- POST
 - Form parameters are included in the request body
 - On reload
 - Browser will ask if it should re-post
 - On bookmark and return
 - Query will proceed with no parameters
 - Use POST to change state

24

Session Management

25

Session Management

- Provide state between HTTP requests
 - Client-side
 - Cookies
 - Hidden variables
 - URL rewriting
 - User authentication
 - `getRemoteUser()`
 - Provided by `HttpServletRequest`
 - Server-side
 - Servlet's built-in session tracking

26

Persistent Cookies

- Stored at the browser
 - As name=value pairs
 - Browsers limit the size of cookies
 - Users can refuse to accept them
- Attached to subsequent requests to the same server
- Servlets can create cookies
 - Included in the HTTP response header

27

Adding a Cookie

- Cookie
 - `Cookie(String name, String value)`
 - `setMaxAge(int expiry)`
 - `setDomain(String pattern)`
 - `setPath(String uri)`
 - `setSecure(boolean flag)`
- `HttpServletRequest`
 - `addCookie(Cookie cookie)`
- `MaxAge`
 - Given in seconds
 - If negative, cookie persists until browser exists
- `Domain`
 - Return cookie to servers matching the specified domain pattern
 - By default, only return to the server that sent it
- `Path`
 - Where the client should return the cookie
 - Visible to all the pages in the specified directory and subdirectory
- `Secure flag`
 - Send cookie only on secure channels

28

Retrieving Cookies

- `HttpServletRequest`
 - `public Cookie[] getCookies()`
- `Cookie`
 - `getName()`
 - `getValue()`
 - `getDomain()`
 - `getPath()`
 - `getSecure()`

29

Servlet's Built-in Session Tracking

- Session tracking API
 - Devoted to servlet session tracking
- Most servers support session tracking
 - Through the use of persistent cookies
 - Able to revert to URL rewriting when cookies fail
 - Servlet uses the token to fetch session state
- Session objects are maintained in memory
 - Some servers allow them to be written to file system or database as memory fills up or when server shuts down
 - Items in the session need to be serializable

30

Session Tracking Basics

- Every user of a site is associated with a `javax.servlet.http.HttpSession` object
 - To store and retrieve information about the user
- Retrieve the current `HttpSession` object
 - `public HttpSession HttpServletRequest.getSession()`
 - Creates one if no valid session found

31

Session Attributes

- To add data to a session object
 - `public void HttpSession.setAttribute(String name, Object value)`
- To retrieve an object from a session
 - `public Object HttpSession.getAttribute(String name)`
- To get the names of all objects in a session
 - `public Enumeration HttpSession.getAttributeNames()`
- To remove an object from a session
 - `public void HttpSession.removeAttribute(String name)`
- Deprecated methods since API 2.2 (**do not use**)
 - `putValue()`, `getValue()`; `getValueNames()`; `removeValue()`

32

Session Lifecycle

- Sessions do not last forever
 - Expire automatically due to inactivity
 - Default 30 min
 - Explicitly invalidated by servlet
- When session expires (or is invalidated)
 - The `HttpSession` object and its data values are removed

33

HttpSession Lifecycle Methods

- `boolean isNew()`
 - Returns true if the client does not yet know about the session or if the client chooses not to join the session
- `void invalidate()`
 - Invalidates this session then unbinds any objects bound to it
- `long getCreationTime()`
 - Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT
- `long getLastAccessedTime()`
 - Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT, and marked by the time the container received the request

34

Setting Session Timeout

- Setting default session timeout
 - Deployment descriptor, `web.xml`
 - Applies to all sessions created in the given web application

```
<session-config>
  <session-timeout>
    30 <!-- minutes -->
  </session-timeout>
</session-config>
```
 - Configured individually
 - `public void HttpSession.setMaxInactiveInterval(int secs)`
 - `public int HttpSession.getMaxInactiveInterval()`
- Terminating a session
 - `session.invalidate()`
 - Deletes session related attributes from server
 - Removes cookie from client

35

ServletContext

- Reference to the web application in which servlets execute
 - One `ServletContext` for each app on the server
 - Allow servlets to access server information
 - `ServletContext GenericServlet.getContext()`
- Using `ServletContext`, servlets can
 - Log events
 - Obtain URL references to resources
 - Access to initialization parameters (in `web.xml`)
 - Share attributes with other servlets in the context
 - Not thread safe

36

JavaServer Pages (JSP)

Currently JSP 2.1

<http://java.sun.com/products/jsp/syntax/2.0/syntaxref20.html>

37

Simple JSP Example

```
<html>
<head>
<title>JSP Example</title>
</head>

<body>
<h1>JSP Example</h1>
<hr/>
<p>
<% out.println("Hello " + request.getParameter("name")); %>
<%= "Hello again, " + request.getParameter("name") + "!"%>
</p>
</body>
</html>
```

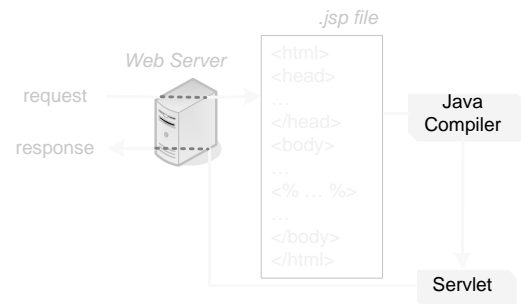
38

JavaServer Pages

- Embed Java servlet code in HTML pages
- Purposes
 - “Enable the separation of dynamic and static content”
 - “Enable the authoring of Web pages that create dynamic content easily but with maximum power and flexibility”
- Server automatically creates, compiles, loads, and runs a special servlet for the page

39

Behind the Scene



40

Elements in JSP (1)

- XHTML
- Scriptlet
 - `<% servlet code for service() %>`
 - `<%-- comment --%>`
- Expressions and declarations
 - `<%= expression %>`
 - Eliminates the clutter of `out.println()`
 - `<%! code for outside of service() %>`
 - Declare static or instance variables, and define new methods
- Directives
 - Control different aspects of the workhorse servlet
 - `<%@ directiveName attribName="attribValue" %>`
 - `<%@ page contentType="text/plain" %>`

41

Elements in JSP (2)

- Include directive (or raw include)
 - Included directly as if it were typed; occurs at translation time
 - `<%@ include file="/header.html" %>`
- Action tags
 - `<jsp:include>`
 - Occurs at request time; includes the output of the dynamic resource
 - `<jsp:include page="pathToDynamicResource" flush="true"/>`
 - Add parameters to request object passed to `<jsp:include>`
 - `<jsp:param name="User" value="John" />`
 - Must be enclosed by a pair of `<jsp:include>` tags
 - Forward a request
 - Occurs at request time
 - `<jsp:forward page="pathToDynamicResource" />`

42

Elements in JSP (3)

- Support for custom tag libraries
 - Let a JSP page contain XML tags that cause specific Java code to be executed
 - `<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>`
 - JSP Standard Tag Library (JSTL)
 - `<c:out value="{param.address}"/>`
 - `<%@ taglib prefix="struts" uri="/WEB-INF/struts.tld" %>`
 - Apache Struts Library
 - Installation is required for custom library use
 - `<struts:property name="cookie" property="name"/>`

43

Active Server Pages (ASP)

- Microsoft-specific technology
- Similar to JSP
 - Use VisualBasic instead of Java
 - Can access a vast array of COM and DCOM objects

44