Web Application: Java Servlets

BY: DR AJUNEWANIS ISMAIL
FACULTY OF COMPUTING
UNIVERSITI TEKNOLOGI MALAYSIA

ESSENTIAL MATERIALS

Prepared by ajune@utm.my
Outline

- Overview of Web application, Servlet technology
- Writing your first servlet
- Running and debugging Servlets
- Handling the client request
  - Form data, retrieve parameters
  - Understand HTTP, HTTP request headers
- Generating the server response
  - HTTP status codes
  - HTTP response headers
- Advanced Servlet Concepts
  - HTTP Redirects, Handling cookies, Session tracking
- Review Servlets

Prepared by ajune@utm.my
Web Application Topics

- Web Application Architecture
  - 1-, 2-, 3-Tier Architectures

- J2EE framework
  - Java Servlets
  - JavaServer Pages
  - Enterprise JavaBeans
  - JDBC
  - JavaMail
  - Java Transaction Service (JTS), …
Web Application model

Client Tier

application

browser

Middle Tier

Web Container

Servlet
Servlet
JSP
...

Enterprise Information System (EIS) Tier

Database

JDBC
A Servlet’s Job

- **Read** explicit data sent by client (form data)
- **Read** implicit data sent by client (request headers)
- **Generate** the results
- **Send** the explicit data back to client (HTML)
- **Send** the implicit data to client (status codes and response headers)
The Advantages of Servlets Over “Traditional” CGI

- **Efficient**
  - Threads instead of OS processes, one servlet copy, persistence
- **Convenient**
  - Lots of high-level utilities
- **Powerful**
  - Sharing data, pooling, persistence
- **Portable**
  - Run on virtually all operating systems and servers
- **Secure**
  - No shell escapes, no buffer overflows
- **Inexpensive**

Prepared by ajune@utm.my
Why Build Pages Dynamically?

- The Web page is based on data submitted by the user
  - E.g., results page from search engines and order-confirmation pages at on-line stores
- The Web page is derived from data that changes frequently (E.g., a weather report)
- The Web page uses information from databases or other server-side sources
  - E.g., an e-commerce site could use a servlet to build a Web page that lists the current price and availability of each item that is for sale
Free Servlet and JSP Engines (Servlet/JSP Containers)

- **Apache Tomcat**
  - Version 4.1.12 - support Servlet 2.3 and JSP 1.2
  - Version 5 – support Servlet 2.4 and JSP 2.0
- **Allaire/Macromedia JRun**
- **New Atlanta ServletExec**
  - [http://www.servletexec.com/](http://www.servletexec.com/)
- **Gefion Software LiteWebServer**
Servlet Engine (container)

IE
Netscape
Opera

Many other Servlets
Compiling and Invoking Servlets

- **Set your CLASSPATH**
  - Servlet JAR file (e.g., `install_dir/lib/servlet.jar`).
  - Top of your package hierarchy

- **Put your servlet classes in proper location**
  - Locations vary from server to server. E.g.,
    - `tomcat_install_dir/webapps/ROOT/WEB-INF/classes`

- **Invoke your servlets (HTTP request)**
  - `http://localhost/servlet/ServletName`
  - Custom URL-to-servlet mapping (via web.xml)
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.println("Hello World");
    }
}
• **Package: javax.servlet**
  - Provides many interfaces and abstract classes for protocol independent generic servlets

• **Package: javax.servlet.http**
  - Provides interfaces and abstract classes for HTTP servlets
  - Extends from the interfaces and classes used in the generic servlets

• Located in `<tomcat_home>/lib/servlet.jar`

• Or, you can get the servlet class file from [http://java.sun.com](http://java.sun.com) for your development.
The skeleton of a common servlet.

```java
public class MyServlet extends HttpServlet {
    public void init() {
        // Initialization here
    }
    public void service() {
        // Your work happens here
    }
    public void destroy() {
        // release resources here
    }
}
```
Generating HTML

- Set the Content-Type header
  - Use `response.setContentType`

- Output HTML
  - Be sure to include the DOCTYPE
  - `PrintWriter.println()`

- Use an HTML validation service
  - [http://validator.w3.org/](http://validator.w3.org/)
public class HelloWWW extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response) 
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String docType = "<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 " + 
            "Transitional//EN"">\n";
        out.println(docType + "<HTML>
            "<HEAD><TITLE>Hello WWW</TITLE></HEAD>
            "<BODY>
            "<H1>Hello WWW</H1>
            "</BODY></HTML>"); 
    }
}
The Servlet Life Cycle

- **init**
  - Executed once when the servlet is first loaded, or at server start. Not called for each request

- **service**
  - Called in a new thread by server for each request. Dispatches to doGet, doPost, etc.
    - Don’t override this method!

- **doGet, doPost, doXxx**
  - Handles GET, POST, etc. requests
  - Override these methods to provide desired behavior

- **destroy**
  - Called when server deletes servlet instance.
    - Not called after each request
The Servlet Life Cycle

- Load:
  - Server
  - Servlet Code

- Handle Client Requests:
  - Server
  - Client

- Unload:
  - Server
  - Servlet Code
• Servlets, JSP pages, HTML files, utility classes, beans, tag libraries, etc. are bundled together in a single directory hierarchy or file.

• Access to content in the Web app is always through a URL that has a common prefix:
  - http://host/webAppPrefix/Servlet/MyServlet

• Many aspects of Web application behavior controlled through deployment descriptor (web.xml):
  - The deployment descriptor is covered in the tutorial.
Purposes of Web Applications
(A single WAR file)

• **Organization**
  - Related files grouped together in a single file or directory hierarchy.
    - HTML files, JSP pages, servlets, beans, images, etc.

• **Portability**
  - All compliant servers support Web apps.
  - Can redeploy on new server by moving a single file.

• **Separation**
  - Each Web app has its own:
    - ServletContext, Class loader
    - Sessions, URL prefix, Directory structure
Structure of a Web Application

- **JSP and regular Web content** (HTML, style sheets, images, etc.):
  - Main directory or a subdirectory thereof.
- **Servlets**:
  - WEB-INF/classes (if servlet is unpackaged – i.e. in default package)
  - A subdirectory thereof that matches the package name.
- **JAR files**:
  - WEB-INF/lib.
- **web.xml**:
  - WEB-INF
- **Tag Library Descriptor files**:
  - WEB-INF or subdirectory thereof
- **Files in WEB-INF not directly accessible to outside clients**
Why You Should Not Override service

- You can add support for other types of requests by adding doPut, doTrace, etc.
- You can add support for modification dates
  - Add a getLastModified method
- The service method gives you automatic support for:
  - HEAD, OPTIONS, and TRACE requests
- Alternative: have doPost call doGet

```java
default void doPost(HttpServletRequest request, HttpServletResponse response) {
    doGet(request, response);
}
```
Common in real-life servlets
- E.g., initializing database connection pools.

Use ServletConfig.getInitParameter to read initialization parameters
- Call getServletConfig to obtain the ServletConfig object

Set init parameters in web.xml (ver 2.2/2.3)
- …/WEB-INF/web.xml
- Many servers have custom interfaces to create web.xml

It is common to use init even when you don’t read init parameters
- E.g., to set up data structures that don’t change during the life of the servlet, to load information from disk, etc.
public class ShowMessage extends HttpServlet {
    private String message;
    private String defaultMessage = "No message."
    private int repeats = 1;
    public void init() throws ServletException {
        ServletConfig config = getServletConfig();
        message = config.getInitParameter("message");
        if (message == null) { message = defaultMessage; }
        try {
            String repeatString = config.getInitParameter("repeats");
            repeats = Integer.parseInt(repeatString);
        } catch (NumberFormatException nfe) {} 
    }
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "The ShowMessage Servlet";
        out.println(ServletUtilities.headWithTitle(title)+
                "<BODY BGCOLOR="#FDF5E6">
                 <H1 ALIGN=CENTER>") + title + "</H1>");
        for(int i=0; i<repeats; i++) {
            out.println(message + "<BR>");
        }
        out.println("</BODY></HTML>");
    }
}
Debugging Servlets

- You don’t execute them directly, but you trigger errors by means of an HTTP request.
- Look at the HTML source
- Return error pages to the client
  - Plan ahead for missing/malformed data
- Use the log file
  - log("message") or log("message", Throwable)
- Look at the request data separately
- Look at the response data separately
- Stop and restart the server
Handling the Client Request: Form Data

- Example URL at online travel agent
  - http://host/path?user=Marty+Hall&origin=iad&dest=nrt
  - Names (user) come from HTML author; values (Marty+Hall) usually come from end user

- Parsing form (query) data in traditional CGI
  - Read the data one way for GET requests, another way for POST requests
  - Chop pairs at &, then separate parameter names (left of the "=") from parameter values (right of the "=")
  - URL decode values (e.g., "%7E" becomes "~")
  - Need special cases for omitted values (param1=val1&param2=&param3=val3) and repeated params (param1=val1&param2=val2&param1=val3)
Reading Form Data (Query Data)

- `getParameter("name")`
  - Returns value as user entered it. I.e., URL-decoded value of first occurrence of name in query string.
  - Works identically for GET and POST requests.
  - Returns null if no such parameter is in query.

- `getParameterValues("name")`
  - Returns an array of the URL-decoded values of all occurrences of name in query string.
  - Returns a one-element array if param not repeated.
  - Returns null if no such parameter is in query.

- `getParameterNames()`
  - Returns Enumeration of request params.
An HTML Form With Three Parameters

```html
<form action="/servlet/cwp.ThreeParams">
    First Parameter: <input type="text" name="param1"><br>
    Second Parameter: <input type="text" name="param2"><br>
    Third Parameter: <input type="text" name="param3"><br>
</form>
```

![Image of a form collecting three parameters](http://localhost/cwp/ThreeParamsForm.html)
public class ThreeParams extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "Reading Three Request Parameters";
        out.println(ServletUtilities.headWithTitle(title) +
            "<BODY BGCOLOR="#FDF5E6"><H1 ALIGN=CENTER>" + title +
            "</H1><ul>
            <li><b>param1</b>: " + request.getParameter("param1") + "
            <li><b>param2</b>: " + request.getParameter("param2") + "
            <li><b>param3</b>: " + request.getParameter("param3") + "
        </ul></BODY></HTML>");
    }
}
Result of ShowParameters Servlet
Server receives the data from user

A Sample FORM using POST

<table>
<thead>
<tr>
<th>Item Number</th>
<th>127A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantity</td>
<td>10</td>
</tr>
<tr>
<td>Price Each</td>
<td>$4.95</td>
</tr>
<tr>
<td>First Name</td>
<td>Marty</td>
</tr>
<tr>
<td>Last Name</td>
<td>Hall</td>
</tr>
<tr>
<td>Middle Name</td>
<td></td>
</tr>
</tbody>
</table>

Shipping Address: Johns Hopkins Applied Physics Lab 11100 Johns Hopkins Rd. Laurel, MD 20723

Credit Card:
- Visa
- Master Card
- American Express
- Discover
- Java SmartCard

Credit Card Number: *********
Repeat Credit Card Number: *********

Submit Order

Reading All Request Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Value(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>address</td>
<td>Johns Hopkins Applied Physics Lab 11100 Johns Hopkins Rd. Laurel, MD 20723</td>
</tr>
<tr>
<td>initial</td>
<td>No Value</td>
</tr>
<tr>
<td>price</td>
<td>$4.95</td>
</tr>
<tr>
<td>cardNum</td>
<td>3.14159 3.14159</td>
</tr>
<tr>
<td>firstName</td>
<td>Marty</td>
</tr>
<tr>
<td>itemNum</td>
<td>127A</td>
</tr>
<tr>
<td>cardType</td>
<td>Java SmartCard</td>
</tr>
<tr>
<td>quantity</td>
<td>12</td>
</tr>
<tr>
<td>lastName</td>
<td>Hall</td>
</tr>
</tbody>
</table>

Note that order of parameters in Enumeration does not match order they appeared in Web page

Prepared by ajune@utm.my
Filtering Strings for HTML-Specific Characters

- You cannot safely insert arbitrary strings into servlet output
  - `< and >` can cause problems anywhere
  - `&` and " cause problems inside of HTML attributes
- You sometimes cannot manually translate
  - String is derived from a program excerpt or another source where it is already in standard format
  - String is derived from HTML form data
- Failing to filter special characters makes you vulnerable to cross-site scripting attack
public static String filter(String input) {
    StringBuffer filtered = new StringBuffer(input.length());
    char c;
    for(int i=0; i<input.length(); i++) {
        c = input.charAt(i);
        if (c == '<') { filtered.append("&lt;"); }
        else if (c == '>') { filtered.append("&gt;" ); }
        else if (c == '"') { filtered.append("&quot; ); }
        else if (c == '&') { filtered.append("&amp;" ); }
        else { filtered.append(c); }
    }
    return(filtered.toString());
}
The Java 'if' Statement

if (a
Note that you must use curly braces when the 'if' or 'else' clauses contain more than one expression.

The Java 'if' Statement

if (a<b) {
    doThis();
} else {
    doThat();
}

Note that you must use curly braces when the 'if' or 'else' clauses contain more than one expression.
Understand HyperText Transfer Protocol (HTTP)

1) Specify Get / Post
2) Request header
3) Form Data (Parameters)

1) Status code
2) Response header
3) Content-Type
4) HTML pages / Other files

Web Client

Web Server

Prepared by ajune@utm.my

SCSJ3323 – SERVLET
Reminder: What do Servlets really do?

- **Handle the incoming requests**
  - Handle Get / Post / or others
  - Read and parse request headers
  - Process form parameters

- **Generate the server response pages**
  - HTTP Status codes for the client
  - Generate HTTP response headers
  - Specify the content type
  - Send the web pages and other files
Break Time – 15 minutes
Handling the Client Request: HTTP Request Headers

- Request headers are distinct from the form data.
- They are indirectly set by the browser.
- Example HTTP 1.1 Request
  - GET /search?keywords=servlets+jsp HTTP/1.1
  - Accept: image/gif, image/jpg, */*
  - Accept-Encoding: gzip
  - Connection: Keep-Alive
  - Cookie: userID=id456578
  - Host: www.somebookstore.com
  - Referer: http://www.somebookstore.com/findbooks.html
  - User-Agent: Mozilla/4.7 [en] (Win98; U)
- The servlet needs to explicitly read these HTTP request headers to make use of this information.
Java Web Client

Web Client

Host: localhost
Port: 80
Request Line: GET /servlet/HelloWorld HTTP/1.0
Request Headers:

Submit Request

Results

HTTP/1.1 200 OK
Content-Length: 23
Date: Sun, 03 Nov 2002 03:16:08 GMT
Server: Apache Coyote/1.0
Connection: close
Hello INE2720 Student

Web Client

Host: localhost
Port: 80
Request Line: GET /ThreeParamsForm.html HTTP/1.0
Request Headers:

Submit Request

Results

<BODY bgcolor="#FDF5E6">
<H1 ALIGN="CENTER">Collecting Three Parameters</H1>

<form action="/servlet/ThreeParams">
First Parameter: <input type="text" name="param1" />
Second Parameter: <input type="text" name="param2" />
Third Parameter: <input type="text" name="param3" />
<input type="submit" />
</form>

Interrupt Download
• **General-purpose way**
  - `getHeader`, `getHeaders`, `getHeaderNames`

• **Specialized – commonly used headers**
  - `getCookies`
  - `getAuthType` and `getRemoteUser`
  - `getContentLength`, `getContentType`
  - `getDateHeader`, `getIntHeader`

• **Related info – main request line**
  - `getMethod`, `getRequestURI`, `getProtocol`
public class ShowRequestHeaders extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "Servlet Example: Showing Request Headers";
        out.println(ServletUtilities.headWithTitle(title) +
            "<BODY BGCOLOR=""#FDF5E6"">" +
            "<H1 ALIGN=CENTER>" + title + "</H1>" +
            "<B>Request Method: </B>" + request.getMethod() + "<BR>" +
            "<B>Request URI: </B>" + request.getRequestURI() + "<BR>" +
            "<B>Request Protocol: </B>" + request.getProtocol() + "<BR>" +
            "<TABLE BORDER=1 ALIGN=CENTER>
            <TR BGCOLOR="#FFAD00">
            <TH>Header Name</TH><TH>Header Value</TH>
            <TR>
            Enumeration headerNames = request.getHeaderNames();
            while(headerNames.hasMoreElements()) {
                String headerName = (String)headerNames.nextElement();
                out.println("<TR><TD>" + headerName);
                out.println("<TD>" + request.getHeader(headerName));
            out.println("</TR></TABLE>");
        out.println("</BODY></HTML>"; }
    public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response);
    }
}
Common HTTP 1.1 Request Headers

- **Accept**
  - Indicates MIME types browser can handle
  - Can send different content to different clients

- **Accept-Encoding**
  - Indicates encodings (e.g., gzip) browser can handle
  - See following example

- **Authorization**
  - User identification for password-protected pages.
  - Instead of HTTP authorization, use HTML forms to send username/password. Store in session object.
• **Connection**
  - In HTTP 1.0, keep-alive means browser can handle persistent connection. In HTTP 1.1, persistent connection is default. **Persistent connections mean that the server can reuse the same socket over again for requests very close together from the same client.**
  - Servlets can't do this unilaterally; the best they can do is to give the server enough info to permit persistent connections. So, they should set Content-Length with `setContentLength`.

• **Cookie**
  - Gives cookies previously sent to client. (getCookies)
• **Host**
  - Indicates host given in original URL
  - This is a required header in HTTP 1.1. This fact is important to know if you write a custom HTTP client (e.g., WebClient used in book) or telnet to a server and use the HTTP/1.1 version

• **If-Modified-Since**
  - Indicates client wants page only if it has been changed after specified date
  - Don’t handle this situation directly.
Common HTTP 1.1 Request Headers (Continued)

- **Referrer**
  - URL of referring Web page
  - Useful for tracking traffic; logged by many servers
  - Can be easily spoofed

- **User-Agent**
  - String identifying the browser making the request
  - Use sparingly
  - Again, can be easily spoofed
public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    response.setContentType("text/html");
    String encodings = request.getHeader("Accept-Encoding");
    String encodeFlag = request.getParameter("encoding");
    PrintWriter out; String title;
    if ((encodings != null) && (encodings.indexOf("gzip") != -1) && !"none".equals(encodeFlag)) {
        title = "Page Encoded with GZip";
        OutputStream out1 = response.getOutputStream();
        out = new PrintWriter(new GZIPOutputStream(out1), false);
        response.setHeader("Content-Encoding", "gzip");
    } else {
        title = "Unencoded Page";
        out = response.getWriter();
    }
    out.println(ServletUtilities.headWithTitle(title) + "<BODY BGCOLOR="#FDF5E6">\n" + "<H1 ALIGN=CENTER>" + title + "</H1>\n"); 
    String line = "Blah, blah, blah, blah, blah. " + "Yadda, yadda, yadda, yadda.";
    for(int i=0; i<10000; i++) {
        out.println(line);
    }
    out.println("</BODY></HTML>");
    out.close();
}
Sending Compressed Pages: Results

- **Uncompressed** (28.8K modem), Netscape 4.7 and Internet Explorer 5.0: > 50 seconds
- **Compressed** (28.8K modem), Netscape 4.7 and Internet Explorer 5.0: < 5 seconds
Generating the Server Response:

1) HTTP Status Codes

- **Example HTTP 1.1 Response**

  HTTP/1.1 200 OK
  Content-Type: text/html (or text/plain or ...)

  ```
  <!DOCTYPE ...>
  <HTML>
  ...
  </HTML>
  ```

- **Changing the status code lets you perform a number of tasks not otherwise possible**
  - Forward client to another page
  - Indicate a missing resource
  - Instruct browser to use cached copy

- **Set status** before sending document
public void setStatus(int statusCode)
  - Use a constant for the code, not an explicit int. Constants are in HttpServletResponse
  - Names derived from standard message. E.g., SC_OK, SC_NOT_FOUND, etc.
  - SC stands for “Status Code”.

public void sendError(int code, String message)
  - Wraps message inside small HTML document
  - Usually 404 with a short message.

public void sendRedirect(String url)
  - Generates a 302 response with a location header
  - Relative URLs permitted in Servlets 2.2/2.3
  - Also sets Location header
SC general categories

- **100-199**
  - Indicate the client should respond with some other actions.

- **200-299**
  - Indicate the request was successful.

- **300-399**
  - Usually include a location header

- **400-499**
  - Indicate an error by the client.

- **500-599**
  - Indicate an error by the server.
Common HTTP 1.1 Status Codes

- **200 (OK)**
  - Everything is fine; document follows
  - Default for servlets

- **204 (No Content)**
  - Browser should keep displaying previous document, no new document is available.

- **301 (Moved Permanently)**
  - Requested document permanently moved elsewhere (indicated in Location header)
  - Browsers go to new location automatically
• **302 (Found)**
  - Requested document temporarily moved elsewhere (indicated in Location header)
  - Browsers go to new location automatically
  - Servlets should use sendRedirect, not setStatus, when setting this header. See example

• **401 (Unauthorized)**
  - Browser tried to access password protected page without proper Authorization header.

• **404 (Not Found)**
  - SC_NOT_FOUND
  - No such page.
  - Servlets should use sendError to set this header
public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    String searchString = request.getParameter("searchString");
    if ((searchString == null) || (searchString.length() == 0)) {
        reportProblem(response, "Missing search string.");
        return;
    }
    searchString = URLEncoder.encode(searchString);
    String numResults = request.getParameter("numResults");
    ...
    String searchEngine = request.getParameter("searchEngine");
    SearchSpec[] commonSpecs = SearchSpec.getCommonSpecs();
    for(int i=0; i<commonSpecs.length; i++) {
        SearchSpec searchSpec = commonSpecs[i];
        if (searchSpec.getName().equals(searchEngine)) {
            String url = searchSpec.makeURL(searchString, numResults);
            response.sendRedirect(url);
            return;
        }
    }
    reportProblem(response, "Unrecognized search engine.");
}
private void reportProblem(HttpServletResponse response, String message) throws IOException {
    response.sendError(response.SC_NOT_FOUND, "<H2>" + message + "</H2>");
}
Front End to Search Engines: Result of Legal Request

Searching the Web

Search String: servlets JSP book
Results to Show Per Page: 10

Google: 2,580 results for servlets JSP book

Core Servlets and JavaServer Pages (JSP): Book Table of Contents
- Contents of Core Servlets and JavaServer Pages (JSP) in HTML...

Core Servlets and JavaServer Pages
- Integrating servlets and JSP

Prepared by ajune@utm.my
Generating the Server Response: 2) HTTP Response Headers

- Purposes
  - Give forwarding location
  - Specify cookies
  - Supply the page modification date
  - Instruct the browser to reload the page after a designated interval
  - Give the document size so that persistent HTTP connections can be used
  - Designate the type of document being generated
  - Etc.
• **public void setHeader**(String headerName, String headerValue)
  ○ Sets an arbitrary header

• **public void setDateHeader**(String name, long millisecs)
  ○ Converts millis since 1970 to date in GMT format

• **public void setIntHeader**(String name, int headerValue)
  ○ Prevents need to convert int to String

• **addHeader**, **addDateHeader**, **addIntHeader**
  ○ Adds header instead of replacing
Methods for specifying common headers

- **setContentType**
  - Sets the Content-Type header (MIME Types). Servlets almost always use this header.

- **setContentLength**
  - Sets the Content-Length header. Used for persistent HTTP connections.

- **addCookie**
  - Adds a value to the Set-Cookie header.

- **sendRedirect**
  - Sets Location header (plus changes status code)
- **Cache-Control (1.1) and Pragma (1.0)**
  - A no-cache value prevents browsers from caching page. Send both headers or check HTTP version.
- **Content-Encoding**
  - The way document is encoded. Browser reverses this encoding before handling document (compression example).
- **Content-Length**
  - The number of bytes in the response
  - Use ByteArrayOutputStream to buffer document so you can determine size.
Common HTTP 1.1 Response Headers (Continued)

- **Content-Type**
  - The MIME type of the document being returned.
  - Use `setContentType` to set this header

- **Expires**
  - The time at which document should be considered out-of-date and thus should no longer be cached
  - Use `setDateHeader` to set this header

- **Last-Modified**
  - The time document was last changed.
  - Don’t set this header explicitly; provide a `getLastModified` method instead.
• **Location**
  - The URL to which browser should reconnect.
  - Use sendRedirect instead of setting this directly.

• **Refresh**
  - The number of seconds until browser should reload page. Can also include URL to connect to.

• **Set-Cookie**
  - The cookies that browser should remember. Don’t set this header directly; use addCookie instead.

• **Via, WWW-Authenticate, ...**
Idea: generate list of large (e.g., 150-digit) prime numbers
  - Show partial results until completed
  - Let new clients make use of results from others

Demonstrates use of the Refresh header

Shows how easy it is for servlets to maintain state between requests
  - Very difficult in traditional CGI

Also illustrates that servlets can handle multiple simultaneous connections
  - Each request is in a separate thread
  - Synchronization required for shared data
public void doGet(HttpServletRequest request, HttpServletResponse response) 
	throws ServletException, IOException {
    int numPrimes = ServletUtilities.getIntParameter(request, "numPrimes", 50);
    int numDigits = ServletUtilities.getIntParameter(request, "numDigits", 120);
    // findPrimeList is synchronized
    PrimeList primeList = findPrimeList(primeListVector, numPrimes, numDigits);
    if (primeList == null) {
        primeList = new PrimeList(numPrimes, numDigits, true);
        synchronized(primeListVector) {
            if (primeListVector.size() >= maxPrimeLists)
                primeListVector.removeElementAt(0);
            primeListVector.addElement(primeList);
        }
    }
    Vector currentPrimes = primeList.getPrimes();
    int numCurrentPrimes = currentPrimes.size();
    int numPrimesRemaining = (numPrimes - numCurrentPrimes);
    boolean isLastResult = (numPrimesRemaining == 0);
    if (!isLastResult) {
        response.setHeader("Refresh", "5");
    }
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    // Show List of Primes found ...
Some 150-Digit Prime Numbers

Primes found with 150 or more digits: 9.
Still looking for 16 more...

1. 686340185130616973272610018202748073761107415752014763897883103299218
2. 686340185130616973272610018202748073761107415752014763897883103299218
3. 686340185130616973272610018202748073761107415752014763897883103299218
4. 686340185130616973272610018202748073761107415752014763897883103299218
5. 686340185130616973272610018202748073761107415752014763897883103299218
6. 686340185130616973272610018202748073761107415752014763897883103299218
7. 686340185130616973272610018202748073761107415752014763897883103299218
8. 686340185130616973272610018202748073761107415752014763897883103299218
9. 686340185130616973272610018202748073761107415752014763897883103299218

Prepared by ajune@utm.my
Break Time – 15 minutes
The Potential of Cookies

• **Idea**
  - Servlet sends a simple name and value to client
  - Client returns same name and value when it connects to same site (or same domain, depending on cookie settings)

• **Typical Uses of Cookies**
  - Identifying a user during an e-commerce session
    - Servlets have a higher-level API for this task
  - Avoiding username and password
  - Customizing a site
  - Focusing advertising
Cookies and Focused Advertising

Prepared by ajune@utm.my

SCSJ3323 – SERVLET
The problem is privacy, not security

- Servers can remember your previous actions
- If you give out personal information, servers can link that information to your previous actions
- Servers can share cookie information through use of a cooperating third party like doubleclick.net
- Poorly designed sites store sensitive information like credit card numbers directly in cookie

Morals for servlet authors

- If cookies are not critical to your task, avoid servlets that totally fail when cookies are disabled.
- Don't put sensitive info in cookies
Sending Cookies to Browser

- **Standard approach:**
  
  ```java
  Cookie c = new Cookie("name", "value");
  c.setMaxAge(...); // Means cookie persists on disk
  // Set other attributes.
  response.addCookie(c);
  ```

- **Simplified approach:**
  
  - Use `LongLivedCookie` class:
    ```java
    public class LongLivedCookie extends Cookie {
        public static final int SECONDS_PER_YEAR =
            60*60*24*365;

        public LongLivedCookie(String name, String value) {
            super(name, value);
           setMaxAge(SECONDS_PER_YEAR);
        }
    }
    ```
• **Standard approach:**

```java
Cookie[] cookies = request.getCookies();
if (cookies != null) {
    for(int i=0; i<cookies.length; i++) {
        Cookie c = cookies[i];
        if (c.getName().equals("someName")) {
            doSomethingWith(c);
            break;
        }
    }
}
```

• **Simplified approach:**
  
  - Extract cookie or cookie value from cookie array by using `ServletUtilities.getCookieValue` or `ServletUtilities.getCookie`
public static String getCookieValue(Cookie[] cookies, String cookieName, String defaultVal) {
    if (cookies != null) {
        for (int i = 0; i < cookies.length; i++) {
            Cookie cookie = cookies[i];
            if (cookieName.equals(cookie.getName()))
                return (cookie.getValue());
        }
    }
    return (defaultVal);
}

- The getCookie method is similar
  - Returns the Cookie object instead of the value
public class SetCookies extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        for (int i = 0; i < 3; i++) {
            Cookie cookie = new Cookie("Session-Cookie-" + i, "Cookie-Value-S" + i);
            response.addCookie(cookie);
            cookie = new Cookie("Persistent-Cookie-" + i, "Cookie-Value-P" + i);
            cookie.setMaxAge(3600);
            response.addCookie(cookie);
        }
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println(...);
    }
}
public class ShowCookies extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "Active Cookies";
        out.println(ServletUtilities.headWithTitle(title) +
            "<BODY BGCOLOR="#FDF5E6">
            "<H1 ALIGN="CENTER"">" + title + "</H1>
            "<TABLE BORDER=1 ALIGN="CENTER">
            "<TR BGCOLOR="#FFAD00">
            "<TH>Cookie Name</TH>
            "<TH>Cookie Value</TH>
        Cookie[] cookies = request.getCookies();
        if (cookies != null) {
            Cookie cookie;
            for(int i=0; i<cookies.length; i++) {
                cookie = cookies[i];
                out.println("<TR>
                    "<TD>" + cookie.getName() + "\n" + "<TD>" + cookie.getValue());
            }
        }
        out.println("</TABLE></BODY></HTML>");
    }
}
Result of Cookie-Viewer (Before & After Restarting Browser)

![Active Cookies - Microsoft Internet Explorer](image)

<table>
<thead>
<tr>
<th>Cookie Name</th>
<th>Cookie Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session-Cookie-0</td>
<td>Cookie-Value-S0</td>
</tr>
<tr>
<td>Persistent-Cookie-0</td>
<td>Cookie-Value-P0</td>
</tr>
<tr>
<td>Session-Cookie-1</td>
<td>Cookie-Value-S1</td>
</tr>
<tr>
<td>Persistent-Cookie-1</td>
<td>Cookie-Value-P1</td>
</tr>
<tr>
<td>Session-Cookie-2</td>
<td>Cookie-Value-S2</td>
</tr>
<tr>
<td>Persistent-Cookie-2</td>
<td>Cookie-Value-P2</td>
</tr>
</tbody>
</table>

![Active Cookies - Microsoft Internet Explorer](image)

<table>
<thead>
<tr>
<th>Cookie Name</th>
<th>Cookie Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Persistent-Cookie-0</td>
<td>Cookie-Value-P0</td>
</tr>
<tr>
<td>Persistent-Cookie-1</td>
<td>Cookie-Value-P1</td>
</tr>
<tr>
<td>Persistent-Cookie-2</td>
<td>Cookie-Value-P2</td>
</tr>
</tbody>
</table>
• **getDomain/setDomain**
  - Lets you specify domain to which cookie applies. Current host must be part of domain specified

• **getMaxAge/setMaxAge**
  - Gets/sets the cookie expiration time (in seconds). If you fail to set this, cookie applies to current browsing session only. See LongLivedCookie helper class given earlier

• **getName/setName**
  - Gets/sets the cookie name. For new cookies, you supply name to constructor, not to setName. For incoming cookie array, you use getName to find the cookie of interest
- **getPath/setPath**
  - Gets/sets the path to which cookie applies. If unspecified, cookie applies to URLs that are within or below directory containing current page
- **getSecure/setSecure**
  - Gets/sets flag indicating whether cookie should apply only to SSL connections or to all connections
- **getValue/setValue**
  - Gets/sets value associated with cookie. For new cookies, you supply value to constructor, not to setValue. For incoming cookie array, you use getName to find the cookie of interest, then call getValue on the result
The specified option will be used as the initial choices next time when you browse the same page.
Session Tracking

Why?
- When clients at an on-line store add an item to their shopping cart, how does the server know what’s already in the cart?
- When clients decide to proceed to checkout, how can the server determine which previously created shopping cart is theirs?
- HTTP is a “Stateless” protocol.

How?
- Cookies
- URL-rewriting
- Hidden form fields

Servlets provide a technical solution: HttpSession API
- Higher-level API needed
The Session Tracking API

- Session objects live on the server
- Automatically associated with client via cookies or URL-rewriting
  - Use `request.getSession(true)` to get either existing or new session
    - Behind the scenes, the system looks at cookie or URL extra info and sees if it matches the key to some previously stored session object. If so, it returns that object. If not, it creates a new one, assigns a cookie or URL info as its key, and returns that new session object.
- Hashtable-like mechanism lets you store arbitrary objects inside session
  - `setAttribute` stores values
  - `getAttribute` retrieves values
```java
HttpSession session = request.getSession(true);
ShoppingCart cart =
    (ShoppingCart)session.getAttribute("shoppingCart");
if (cart == null) {
    // No cart already in session
    cart = new ShoppingCart();
    session.setAttribute("shoppingCart", cart);
}
doSomethingWith(cart);
```
HttpSession Methods

- **getAttribute, getValue** [2.1]
  - Extracts a previously stored value from a session object. Returns null if no value is associated with given name.

- **setAttribute, putValue** [2.1]
  - Associates a value with a name. Monitor changes: values implement HttpSessionBindingListener.

- **removeAttribute, removeValue** [2.1]
  - Removes values associated with name.

- **getAttributeNames, getValueNames** [2.1]
  - Returns names of all attributes in the session.

- **getId**
  - Returns the unique identifier for each session.
HttpSession Methods (Continued)

- **isNew**
  - Determines if session is new to client (not to page)

- **getCreationTime**
  - Returns time at which session was first created

- **getLastAccessedTime**
  - Returns time session was last sent from client

- **getMaxInactiveInterval, setMaxInactiveInterval**
  - Gets or sets the amount of time session should go without access before being invalidated

- **invalidate**
  - Invalidates the session and unbinds all objects associated with it
public void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Session Tracking Example";
    HttpSession session = request.getSession(true);
    String heading;
    Integer accessCount = (Integer)session.getAttribute("accessCount");
    if (accessCount == null) {
        accessCount = new Integer(0);
        heading = "Welcome, Newcomer";
    } else {
        heading = "Welcome Back";
        accessCount = new Integer(accessCount.intValue() + 1);
    }
    session.setAttribute("accessCount", accessCount);
First and Eleventh Visit to ShowSession Servlet

Welcome, Newcomer

Information on Your Session:

<table>
<thead>
<tr>
<th>Info Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>To1010mC6608409833952431At</td>
</tr>
<tr>
<td>Creation Time</td>
<td>Mon Apr 16 10:41:25 EDT 2001</td>
</tr>
<tr>
<td>Time of Last Access</td>
<td>Mon Apr 16 10:44:54 EDT 2001</td>
</tr>
<tr>
<td>Number of Previous Accesses</td>
<td>0</td>
</tr>
</tbody>
</table>

Welcome Back

Information on Your Session:

<table>
<thead>
<tr>
<th>Info Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>To1010mC6608409833952431At</td>
</tr>
<tr>
<td>Creation Time</td>
<td>Mon Apr 16 10:41:25 EDT 2001</td>
</tr>
<tr>
<td>Time of Last Access</td>
<td>Mon Apr 16 10:44:54 EDT 2001</td>
</tr>
<tr>
<td>Number of Previous Accesses</td>
<td>10</td>
</tr>
</tbody>
</table>
Servlets are efficient, portable, powerful, and widely accepted in industry

Regardless of deployment server, run a free server on your desktop for development

Getting started:

- Set your CLASSPATH
  - Servlet and JSP JAR files
  - Top of your package hierarchy
- Put class files in proper location
  - .../WEB-INF/classes with servlets 2.2/2.3
- Use proper URL; default is http://host/servlet/ServletName
Main servlet code goes in doGet or doPost:
- The HttpServletRequest contains the incoming information
- The HttpServletResponse lets you set outgoing information
  - Call setContentType to specify MIME type
  - Call getWriter to obtain a Writer pointing to client

One-time setup code goes in init
- Servlet gets initialized and loaded once
- Servlet gets invoked multiple times
- Query data comes from HTML forms as URL-encoded name/value pairs
- Servlets read data by calling request.getParameter("name")
  - Results in value as entered into form, not as sent over network.
  - Always check for missing or malformed data
  - Special case: query data that contains special HTML characters (filtering).
Many servlet tasks can only be accomplished by making use of HTTP headers coming from the browser.

- Use `request.getHeader` for arbitrary header.
- Cookies, authorization info, content length, and content type have shortcut methods.
- Most important headers you read directly:
  - `Accept`,
  - `Accept-Encoding`
  - `Connection`, `Referer`, `User-Agent`
Many servlet tasks can only be accomplished through use of HTTP status codes and headers sent to the browser

Two parts of the response

- **Status line**
  - In general, set via `response.setStatus`
  - In special cases, set via `response.sendRedirect` and `response.sendError`

- **Response headers**
  - In general, set via `response.setHeader`
  - In special cases, set via
    - `response.setContentType`
    - `response.setContentLength`
    - `response.addCookie`, ...
• Most important status codes
  ○ 200 (default)
  ○ 302 (forwarding; set via sendRedirect)
  ○ 401 (password needed)
  ○ 404 (not found; set via sendError)

• Most important headers you set directly
  ○ Cache-Control and Pragma
  ○ Content-Encoding
  ○ Content-Length
  ○ Expires, Refresh
  ○ WWW-Authenticate
Cookies involve name/value pairs sent from server to browser and returned when the same page, site, or domain is visited later, you can

- Track sessions (use higher-level API)
- Permit users to avoid logging in at low-security sites
- Customize sites for different users
- Focus content or advertising

**Setting cookies**
- Cookie constructor, set age, response.addCookie

**Reading cookies**
- Call request.getCookies
Although it usually uses cookies behind the scenes, the session tracking API is higher-level and easier to use than the cookie API.

Session information lives on server
- Cookie or extra URL info associates it with a user

Obtaining session
- request.getSession(true)

Associating values with keys
- session.setAttribute

Finding values associated with keys
- session.getAttribute
  - Always check if this value is null before trying to use it
With servlets, it is easy to
- Read form data
- Read HTTP request headers
- Set HTTP status codes and response headers
- Use cookies and session tracking
- Share data among servlets
- Remember data between requests
- Get fun, high-paying jobs

But, it sure is a pain to
- Use those println statements to generate HTML
- Maintain that HTML
Although JSP technically can't do anything servlets can't do, JSP makes it easier to:

- Write HTML, read and maintain the HTML

JSP makes it possible to:

- Use standard HTML tools such as HomeSite or UltraDev
- Have different members of your team do the HTML layout and the programming

JSP encourages you to

- Separate the (Java™ technology) code that creates the content from the (HTML) code that presents it
References

- http://java.sun.com/docs/books/tutorial/servlets/
- Core Servlets and JavaServer Pages
- More Servlets and JavaServer Pages
- The End.
- Thank you for patience!