**Chap 2.1: Servlet Technology**

1. **Overview of Servlet Technology**

* Servlets are Java programs that run on Web application servers
* They are the middle layer between Web browsers’ requests / other HTTP clients & databases / applications on the HTTP server

Dynamic Web Pages:

* Web page is based on data sent by the client
* Web page is derived from data that changes frequently
* Web page uses information from corporate databases or other server-side sources

|  |  |
| --- | --- |
| A Servlet Job | |
| Read the client’s data | Servlets receive data from:   * Data from HTML forms on a Web page, which is entered by users * Data from an applet or a custom HTTP client program |
| Read the browser’s HTTP request data | The (behind-the-scenes) HTTP data (cookies,media types information and compression schemes) which is understood by browsers |
| Generate the results | * Interact with databases, invoke web services, compute the response directly,… |
| Send the explicit data to the client | Return documents in a variety of formats:   * Text (HTML or XML) - an important servlets’ task is to wrap the results inside of HTML * Binary (GIF images) * Compressed format (gzip) |
| Send the implicit HTTP response data to browser | Sending HTTP response data involves:   * Telling the browser / other client the type of returned document is being – * Setting cookies and caching parameters, and other tasks |

|  |  |
| --- | --- |
| The Advantages of Servlets | |
| Efficiency | * The Java virtual machine stays running and handles each request with a lightweight Java thread * N threads to the same servlets 🡪 a single copy of the servlet would be loaded 🡪 reduces memory and saves time * Finish handling a request 🡪 remain in memory |
| Convenient | Servlets have an extensive infrastructure for automatically parsing and decoding HTML form data, reading and setting HTTP headers, handling cookies, tracking sessions, and many other such high-level utilities |
| Powerful | * Servlets can talk directly to the Web server 🡪 easier to translate relative URLs into concrete path names * Multiple servlets can share data 🡪 easy to implement database connection pooling and resource-sharing optimizations * Servlets can maintain information from request to request |
| Portable | * Servlets are written in the Java and follow a standard API * Servlets are supported directly or by a plugin on virtually every major Web server: Macromedia JRun, Apache Tomcat, Microsoft Internet Information Server, IBM WebSphere, iPlanet Enterprise Server, Oracle9i AS, or StarNine WebStar |
| Inexpensive | * With servlets & JSP you can start with a free / inexpensive server 🡪 migrate to more expensive servers with high-performance capabilities / advanced administration utilities later |
| Secure | * Servlets avoid using general-purpose operating system shells, which must be filtered out characters such as / \ { } that are treated specially by the shell * Servlets automatically check array or string bounds |

1. **Servlet Basics**

**2.1 Basic Servlet Structure –** handles get & post requests and generate plain text

* **get** (default method in HTML form) is basically for getting data 🡪 doGet()
* **post** (recommend method in HTML form) involves anything, such as: storing, updating data, ordering a product, sending E-mail) 🡪 doPost()
* doGet() and doPost() take 2 arguments:
  + **HttpServletRequest**: get at all of the incoming data
    - Form query data
    - HTTP request headers
    - Client's hostname
  + **HttpServletResponse:** specify outgoing information
    - HTTP status codes (200, 404, 500,…)
    - Response headers (Content-Type, Set-Cookie,…)
    - **PrintWriter** (important): use to send HTML/XHTML documents back to the client
* Important library in servlets:
  + **java.io.\*** for PrintWriter
  + **javax.servlet.\*** for HttpServlet
  + **javax.servlet.http.\*** for HttpServletRequest and HttpServletResponse

|  |
| --- |
| import java.io.\*;  import javax.servlet.\*;  import javax.servlet.http.\*;  public class ServletTemplate extends HttpServlet {  public void **doGet(HttpServletRequest request, HttpServletResponse response)**  **throws ServletException, IOException** {  // Use "request" to read incoming HTTP headers  // (e.g., cookies) and query data from HTML forms.  // Use "response" to specify the HTTP response status  // code and headers (e.g., the content type, cookies).  **PrintWriter out = response.getWriter();** // Use "out" to send content to browser.  out.println("Hello World");  }  } |
| **//Declare the same reposne for doPost & doGet**  public void doGet(HttpServletRequest request, HttpServletResponse response)  throws ServletException, IOException {  }  public void doPost(HttpServletRequest request, HttpServletResponse response)  throws ServletException, IOException {  **doGet(request, response);**  } |

**2.2 Servlets that generate HTML**

Responding HTML pages includes 2 steps:

* Tell the browser that returned document is a HTML
  + Syntax: **response.setContentType("value”)**
  + The **value = text/html** (HTML), **application/vnd.ms-excel** (Excel), **image/jpeg** (JPEG image), **text/xml** (XML),…
* Modify the out.println() statements to build a legal Web page

|  |
| --- |
| public class HelloServlet extends HttpServlet {  public void doGet(HttpServletRequest request, HttpServletResponse response)  throws ServletException, IOException {    **response.setContentType("text/html"); // set response headers before the**  **//PrintWriter**    PrintWriter out = response.getWriter();  String docType = "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +  "Transitional//EN\">\n";    **out.println(docType +**  **"<HTML>\n" +**  **"<HEAD><TITLE>Hello</TITLE></HEAD>\n" +**  **"<BODY BGCOLOR=\"#FDF5E6\">\n" +**  **"<H1>Hello</H1>\n" +**  **"</BODY></HTML>");**  }  } |

* 1. **Servlet Packaging**

Servlet **should be always in packages** (or else it doesn’t run). When you put your servlets in packages, need to perform 2 additional steps:

* Place the files in a package. Recommend:
  + coreservlets package for only servlets
  + corejsp package for all jsp
* Insert a package statement in the class file. For example: **package coreservlets**

|  |
| --- |
| **package coreservlets;**  import java.io.\*;  import javax.servlet.\*;  import javax.servlet.http.\*;  public class HelloServlet extends HttpServlet {  public void doGet(HttpServletRequest request, HttpServletResponse response)  throws ServletException, IOException {  }  } |

**2.4 HTML-Building Utilities**

If you're repeating the same HTML constructs many times, you might as well create a simple utility class that simplifies those constructs

|  |
| --- |
| // define a general format using this class **package coreservlets;**  import javax.servlet.\*;  import javax.servlet.http.\*;  public class **ServletUtilities** {  public static final String DOCTYPE ="<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +  "Transitional//EN\">";  public static String **headWithTitle**(String title) {  return(DOCTYPE + "\n" +  "<HTML>\n" +  "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n");  }  } |
| **package coreservlets;**  import java.io.\*;  import javax.servlet.\*;  import javax.servlet.http.\*;  public class HelloServlet3 extends HttpServlet {  public void doGet(HttpServletRequest request, HttpServletResponse response)  throws ServletException, IOException {    response.setContentType("text/html");  PrintWriter out = response.getWriter();  **String title = "Hello (3)";** // create a title name  out.println(**ServletUtilities.headWithTitle(title)** + **//using ServletUtilities class**  "<BODY BGCOLOR=\"#FDF5E6\">\n" + **//to format the title**  "<H1>" + title + "</H1>\n" +  "</BODY></HTML>");  }  } |

### 

### 2.5 The Servlet Life Cycle

1. When the servlet is created the **init** method is invoked
2. Then each user request becomes a thread that calls the **service** method
3. The service method then calls **doGet, doPost**, or another **doXxx** method
4. If the server decides to unload a servlet, it calls the **destroy** method

**3. Handling the Clients Request: Form Data**

**3.1 Reading Form Data**

|  |  |
| --- | --- |
| Method | Description |
| General rule | * The input:   + Must supply the parameter name exactly as in the HTML   + Parameter names are case-sensitive: “Param1”≠“param1” * The output:   + If a parameter exists but has no value (space character) 🡪 empty string   + If a parameter doesn’t exist (user left it blank) 🡪 null |
| **getParameter()** | * Read the value of a particular parameter * Syntax: **request.getParameter("paraName")** * The output: a string = value that users entered in the HTML form |
| **getParameterValues()** | * If a parameter appears many times in the HTML form 🡪 has many values * Syntax: **request.getParameterValues("paraName")** * The output:   + If a paremeter has many values 🡪 an array of strings   + If a parameter has 1 single value 🡪 1-element array |
| **getParameterNames()** | * Get the list of all parameters in the HTML form * Syntax: **Enumeration paramNames = request.getParameterNames();** * The output:   + Each parameter 🡪 an object in the enumeration   + No parameters 🡪 empty enumeration * Note: the parameters aren’t in the order that they appeared in the HTML |
| **getReader()**  **getInputStream()** | * Reading the raw data * Syntax:   **request.getReader("paraName")**  **request.getInputStream("paraName")** |
| **setCharacterEncoding()** | * Auto detect, accept and read multiple kind languages (Japanese, Korean,…) * Syntax:   **request.setCharacterEncoding("JISAutoDetect");**  String firstName = **request.getParameter("firstName");** |

|  |
| --- |
| <h1 align="center">collecting three parameters</h1>  <form action="/servlet/coreservlets.threeparams" method=”post”>  **first parameter:<input type="text" name="param1"><br>**  **second parameter:<input type="text" name="param2"><br>**  **third parameter:<input type="text" name="param3"><br>**  <center><input type="submit"></center>  </form> |
| ...  response.setContentType("text/html");  PrintWriter out = response.getWriter();    String title = "Reading Three Request Parameters";  String docType = "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +  "Transitional//EN\">\n";    out.println(docType +  "<html><head><title>" + title + "</title></head>\n" +  "<body bgcolor=\"#FDF5E6\">\n" +  "<h1 align=center>" + title + "</h1>\n" +  "<ul>\n" +  " <li><b>param1</b>: " + **request.getParameter("param1")** + "\n" +  " <li><b>param2</b>: " + **request.getParameter("param2")** + "\n" +  " <li><b>param3</b>: " + **request.getParameter("param3")** + "\n" +  "</ul>\n" +  "</body></html>");  ...  ... |

**Using the getParameterNames() method:**

* Step 1: call the getParameterNames() method
* Step 2: this method returns an enumeration that have the parameter names in an unspecified order
* Step 3: while() loops down the enumeration in the standard manner:
  + **Using hasMoreElements to determine when to stop**
  + **Using nextElement to get the parameter name 🡪 returns an object 🡪 cast to string and pass to getParameterValues() method**

|  |
| --- |
| <h1 align="center">a sample form using post</h1>  <form action="/servlet/coreservlets.showparameters" method="post">  item number: <input type="text" name="itemnum"><br>  description: <input type="text" name="description"><br>  price each: <input type="text" name="price" value="$"><br><hr>  first name: <input type="text" name="firstname"><br>  last name: <input type="text" name="lastname"><br>  middle initial: <input type="text" name="initial"><br>  shipping address: <textarea name="address" rows=3 cols=40></textarea><br>    credit card:<br>  <input type="radio" **name="cardtype"** value="visa">visa<br>  <input type="radio" **name="cardtype"** value="mastercard">mastercard<br>  <input type="radio" **name="cardtype"** value="amex">american express<br>  <input type="radio" **name="cardtype"** value="discover">discover<br>  <input type="radio" **name="cardtype"** value="java smartcard">java smartcard<br>  credit card number:<input type="password" name="cardnum"><br>  repeat credit card number:<input type="password" name="cardnum"><br><br>  <input type="submit" value="submit order">  </form> |
| ...  **import java.util.\*;**  //Present the output using table-------------------------------------------------------  out.println(docType +  "<html>" +  "<head><title>" + title + "</title></head>\n" +  "<body>\n" +  "<h1>" + title + "</h1>\n" +  **"<table border=1><tr><th>Parameter Name</th><th>Parameter Value</th>");**  //Enumeration loop---------------------------------------------------------------------  **Enumeration paramNames = request.getParameterNames();**    **while(paramNames.hasMoreElements()**) {  String paramName = (**String)paramNames.nextElement();** //get the parameter names  out.print("<tr><td>" + paramName + "</td>");  String[] paramValues = request.getParameterValues(paramName);  //import paravalue into an array string  if (paramValues.length == 1) {  String paramValue = paramValues[0];  if (paramValue.length() == 0)  out.println("<td>No Value</td>");  else  out.println(“<td>” + paramValue + “</td>”);  }  else { //param has more tan one values, output as list  out.println("<td><ul>");  for(int i=0; i<paramValues.length; i++) {  out.println("<li>" + paramValues[i]+”</li>”);  }  out.println("</ul></td>");  }  out.println("</tr");  }  out.println("</table></body></html>"); //end table  }  //Have doPost to call doGet------------------------------------------------------------  public void doPost(HttpServletRequest request, HttpServletResponse response)  throws ServletException, IOException {  doGet(request, response);  }  } |

**3.3 Checking Parameter Values – Solution for Missing Values**

* When users submit the wrong information 🡪 2 solutions: use default values or redisplay the form (prompting the user for missing values)

|  |  |
| --- | --- |
| 3 conditions need to be checked | |
| The value is null | * User uses an incorrect HTML form * A bookmarked URL containing GET data is used but the parameter names have changed * Solution: check for null in the beginning of the servlet |
| The value is an empty string | * Solution: compare the string to "" or compare the length of the string to 0 * Note: call trim to remove any white space before comparing   String param = request.getParameter("someName");  **if ((param == null) || (param.trim().equals("")))** {  doSomethingForMissingValues(...);  } else {  doSomethingWithParameter(param);  } |
| The value is in the wrong format | * Certain textfields have it own input format. Ex:   + Only numeric values   + Exactly 7 characters   + Only single letters * Users input the wrong format for the data |

**4. Handling the Client Request: HTTP Request Headers**

**4.1 HTTP 1.1 Request Headers**

|  |  |  |
| --- | --- | --- |
| Accept | Show the acceptable MIME-Types | text/plain |
| Accept-Charset | Show the acceptable character sets | utf-8 |
| Accept-Encoding | Show the acceptable encodings | compress | gzip | deflate | sdch | identity |
| Accept-Language | Show the acceptable languages | en-US |
| Authorization | Show the authentication credentials | QWxhZGRpbjpvcGVuIHNlc2FtZQ== |
| Cache-Control | Specify directives that MUST be obeyed by all caching mechanisms along the request/response chain | no-cache |
| Connection | Show the type of connection the user-agent prefers | Close, keep-alive |
| Cookie | Show the cookie previously sent by the server | $Version=1; Skin=new; |
| Content-Length | Show the length of the request body in octets | 348 |
| Content-Type | Show the MIME-Types of the body of the request (for post & put requests) | application/x-www-form-urlencoded |
| Date | Show the date and time that the message was sent | Tue, 15 Nov 1994 08:12:31 GMT |
| From | Show the email address of the user making the request | user@example.com |
| Host | Show the domain name of the server | en.wikipedia.org |
| If-Modified-Since | Return a 304\_Not\_Modified if content is unchanged since a specific time | Sat, 29 Oct 1994 19:43:31 GMT |
| If-Unmodified-Since | Send the response if the entity hasn’t been modified since a specific time | Sat, 29 Oct 1994 19:43:31 GMT |
| Referer | Show the address of the previous web page from which a link to the currently requested page was followed | http://en.wikipedia.org/wiki/Main\_Page |
| User-Agent | Show the user agent (browsers) | Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0) |
| Via | Show the server of proxies through which the request was sent | 1.0 fred, 1.1 nowhere.com |

**4.2 Reading Request Headers**

|  |  |
| --- | --- |
| Method | Description |
| General rule | * The input:   + Must supply the header name exactly as in HTTP 1.1   + Header names are not case-sensitivem * The output:   + If a **header value** exists 🡪 return a string   + If a **header value** doesn’t exist 🡪 “null” |
| **getHeader()** | * Read the value of a particular header * Syntax: **request.getHeader(String headName)** * The output: a string = value that is supplied in the current request |
| **getHeaderNames()** | * Get an Enumeration of **all header names** received on a **particular request** * Syntax: **Enumeration header = request.getHeaderNames()** |
| **getHeaders()** | * Obtain an Enumeration of **all values** of a **particular header** * Syntax: **Enumeration header = request.getHeaders(String headName)** |
| **getCookies()** | * Read the contents of the Cookie header * Syntax: **request.getCookies()** * The output: an array of Cookie objects |
| **getAuthType()**  **getRemoteUser()** | * Break the Authorization header into its component pieces * Syntax:   **request.getAuthType()**  **request.getRemoteUser()** |
| **getContentLength()** | * Returns the value of the Content-Length header (as an int) * Syntax: **request.getContentLength()** |
| **getContentType()** | * Returns the value of the Content-Type header (as a String) * Syntax: **request.getContentType()** |
| **getDateHeader()**  **getIntHeader()** | * Read the specified headers and then convert them to Date and int values * Syntax:   **request.getDateHeader()**  **request.getIntHeader()** |
| **getMethod()** | * Returns the main request method (GET, POST, HEAD, PUT, and DELETE) * Syntax: **request.getMethod()** |
| **getRequestURI()** | * Returns the part of the URL that comes after the host and port but before the form data. * Example: http://randomhost.com/servlet/search.BookSearch?subject =jsp, getRequestURI would return "/servlet/search.BookSearch" * Syntax: **request.getRequestURI()** |
| **getQueryString()** | * Returns the form data * Example: http://randomhost.com/servlet/search.BookSearch?subject =jsp, getQueryString would return "subject=jsp". * Syntax: **request.getQueryString()** |
| **getProtocol()** | * Returns the third part of the request line, which is generally HTTP/1.0 or HTTP/1.1 * Syntax: **request.getProtocol()** |

|  |
| --- |
| **//Making a Table of All Request Headers**  out.println(docType + "<html>\n" +  "<head><title>" + title + "</title></head>\n" +  "<body bgcolor=#FDF5E6>\n" +  "<h1 align=CENTER>" + title + "</h1>\n" +  "<b>Request Method: </b>" + request.getMethod() + "<br>\n" +  "<b>Request URI: </b>" + request.getRequestURI() + "<br>\n" +  "<b>Request Protocol: </b>" + request.getProtocol() + "<br><br>\n" +    "<table border=1><tr><th>Header Name<th>Header Value");    Enumeration headerNames = request.getHeaderNames();  while(headerNames.hasMoreElements()) {  String headerName = (String)headerNames.nextElement();  out.println("<tr><td>" + headerName + “</td>);  out.println("<td>" + request.getHeader(headerName) + “</td></tr>);  }  out.println("</table></body></html>"); |

**4.3 Sending a compressed webpage**

* Gzip is a text compression scheme that reduces the size of HTML (or plain text) pages

|  |
| --- |
| ...  ...  public class LongServlet extends HttpServlet {  public void doGet(HttpServletRequest request, HttpServletResponse response)  throws ServletException, IOException {    response.setContentType("text/html");  PrintWriter out;  //Change the definition of "out" depending on whether or not gzip is supported.  **if (isGzipSupported(request) && !isGzipDisabled(request)) {**  **out = getGzipWriter(response);**  **response.setHeader("Content-Encoding", "gzip");**  **} else {**  **out = response.getWriter();**  **}**  //Once "out" has been assigned appropriately, the rest of the page has no dependencies //on the type of writer being used  String docType = "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +  "Transitional//EN\">\n";    out.println(docType + "<html><head><title>Gzip HTML</title></head>" +  "<body>" +  "<h1 align=\"center\">Gzip HTML</h1>" +  "<p>");    String line = "Blah, blah, blah, blah, blah. " + "Yadda, yadda, yadda, yadda.";  for(int i=0; i<10000; i++) {  out.println(line);  }  out.println("</p></body></html>");  out.close(); // Needed for gzip  }  }  **//Check if the browser is gzip supported------------------------------------**  public static boolean isGzipSupported (HttpServletRequest request) {  String check = request.getHeader("Accept-Encoding");  return((check != null) && (check.indexOf("gzip") != -1));  }  **//Check if user disabled gzip-----------------------------------------------**  public static boolean isGzipDisabled (HttpServletRequest request) {  String isDisabled = request.getParameter("disableGzip");  return((isDisabled != null) && (!isDisabled.equalsIgnoreCase("false")));  }  **//Return gzipping PrintWriter for response----------------------------------**  public static PrintWriter getGzipWriter (HttpServletResponse response) throws IOException {  return(new PrintWriter (new GZIPOutputStream(response.getOutputStream())));  } |

**5.0 Generating the Server Response: HTTP Status Codes**

**5.1 Specifying Status Codes**

* The HTTP response status line consists:
  + HTTP version (determined by server)
  + A status code (determined by servlet)
  + An associated message (directly associated with the status code)
* Set a status code before sending any document with the PrintWriter
* Syntax :

**response.setStatus(status code)**

**response.setStatus(“SC\_...”)**

|  |  |  |
| --- | --- | --- |
| **1xx Informational** - indicating that the client should respond with some other action | | |
| 100 | SC\_CONTINUE | The client is asking if it can send an attached document in a follow-up request |
| **2xx Success** - the request was successful | | |
| 200 | SC\_OK | * Everything is fine * The document follows for get and post requests. * Default for servlets |
| 202 | SC\_ACCEPTED | The request is accpeted but processing is not yet complete |
| 204 | SC\_NO\_CONTENT | Tell the browser to display the previous document because no new document is available. |
| 205 | SC\_RESET\_CONTENT | * Reset the document view * This status code is used to instruct browsers to clear form fields |
| **3xx Redirection** | | |
| 301 | SC\_MOVED\_PERMANENTLY | * The requested document is elsewhere * The new URL for the document is given in the Location response header * Browsers automatically follow the link to the new URL |
| 302 | SC\_FOUND | Similar to 301, except that the URL given by the Location header should be interpreted as a temporary replacement |
| **4xx Error by the client** | | |
| 400 | SC\_BAD\_REQUEST | Bad syntax in the client request |
| 401 | SC\_UNAUTHORIZED | The client tried to access a password-protected page but that the request did not have proper identifying information in the Authorization header |
| 403 | SC\_FORBIDDEN | The server refuses to supply the resource, regardless of authorization |
| 404 | SC\_NOT\_FOUND | No resource could be found at that address |
| 405 | SC\_METHOD\_NOT\_ALLOWED | The request method (GET, POST, HEAD, PUT, DELETE,…) was not allowed for this particular resource |
| 415 | SC\_UNSUPPORTED\_MEDIA\_  TYPE | The request had an attached document of a type the server doesn't know how to handle |
| **5xx Error by the server** | | |
| 500 | SC\_INTERNAL\_SERVER\_ERROR | * "server is confused" status code * Servlets that crash or return improperly formatted headers |
| 501 | SC\_NOT\_IMPLEMENTED | The server doesn't support the functionality to fulfill the request |
| 503 | SC\_SERVICE\_UNAVAILABLE | The server cannot respond because of maintenance or overloading |
| 505 | SC\_HTTP\_VERSION\_NOT\_  SUPPORTED | The server doesn't support the version of HTTP named in the request line |

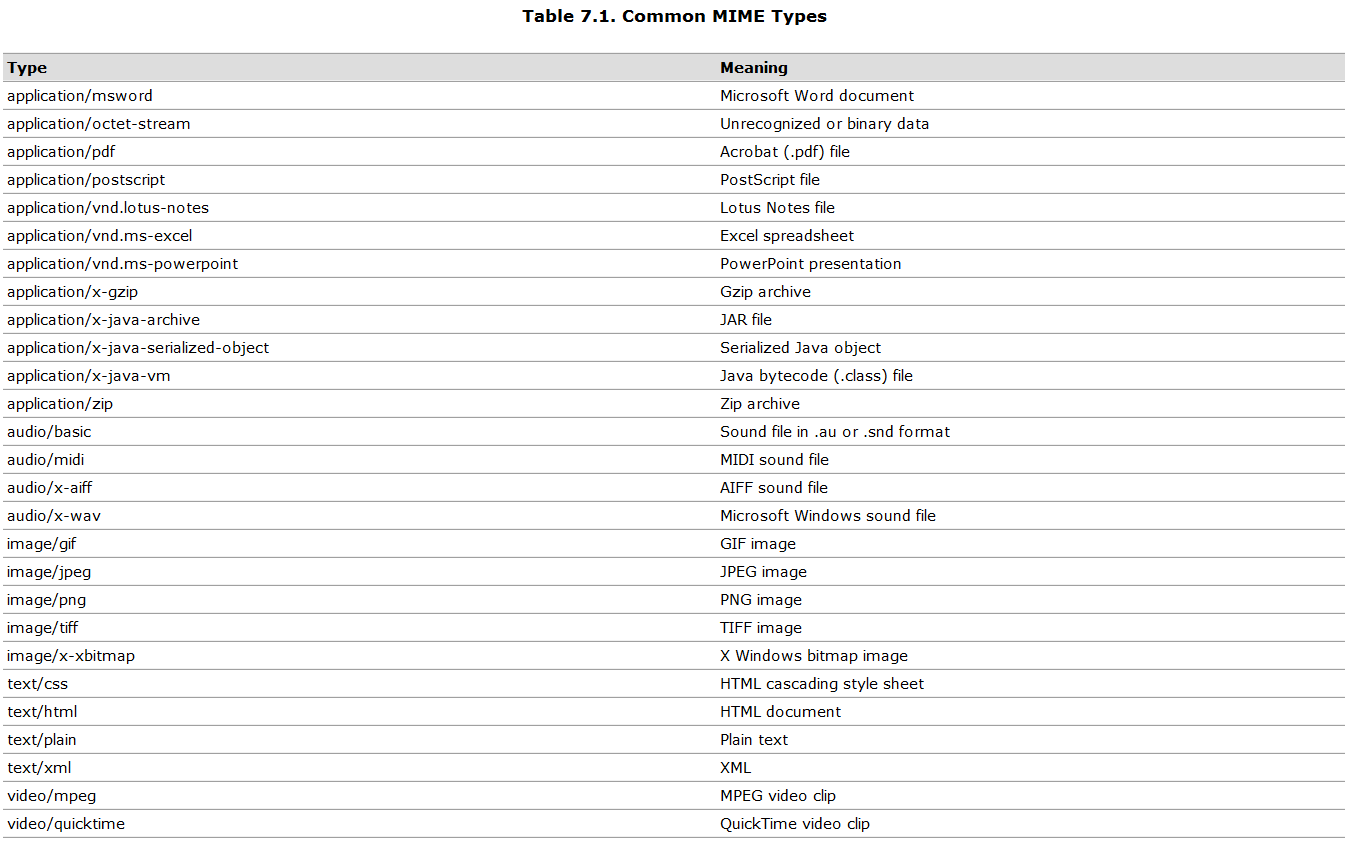
**5.2 Redirects Users to Browser-Specific Pages**

|  |
| --- |
| package coreservlets;  import java.io.\*;  import javax.servlet.\*;  import javax.servlet.http.\*;  /\*\* Servlet that sends IE users to the Netscape home page and  \* Netscape (and all other) users to the Microsoft home page.  \*/  public class WrongDestination extends HttpServlet {  public void doGet(HttpServletRequest request,  HttpServletResponse response)  throws ServletException, IOException {  **String userAgent = request.getHeader("User-Agent");**  **if ((userAgent != null) && (userAgent.indexOf("MSIE") != -1)) {**  **response.sendRedirect("http://home.netscape.com");**  **} else {**  **response.sendRedirect("http://www.microsoft.com");**  }  }  } |

**6. Generating the Server Response: HTTP Response Headers**

**6.1 HTTP 1.1 Response Headers**

|  |  |  |
| --- | --- | --- |
| Allow | Valid request method for a specified resource. To be used for a 405 Method\_Not\_Allowed | GET, HEAD |
| Content-Encoding | Set the type of encoding used on the data | gzip |
| Content-Language | Set the language the content is in | da |
| Content-Length | Set the length of the response body in octets | 348 |
| Content-Location | Set the location for the returned data | /index.htm |
| Content-Type | Set the mime type | text/html; charset=utf-8 |
| Expires | Set the expired date/time after which the response | Thu, 01 Dec 1994 16:00:00 GMT |
| Last-Modified | Set the last modified date for the requested object | Tue, 15 Nov 1994 12:45:26 GMT |
| Location | Used in redirection, or when a new resource has been created. | http://www.w3.org/pub/WWW/People.html |
| Pragma | Implementation-specific headers that may have various effects anywhere along the request-response chain | no-cache |
| Refresh | Used in redirection, or when a new resource has been created. This refresh redirects after 5 seconds. This is a proprietary, non-standard header extension introduced by Netscape and supported by most web browsers. | 5; url=http://www.w3.org/pub/WWW/People.html |
| Retry-After | If an entity is temporarily unavailable, this instructs the client to try again after a specified period of time. | 120 |
| Server | A name for the server | Apache/1.3.27 (Unix) (Red-Hat/Linux) |
| Set-Cookie | an HTTP cookie | UserID=JohnDoe; Max-Age=3600; Version=1 |
| WWW-Authenticate | Indicates the authentication scheme that should be used to access the requested entity. | Basic |



**6.2 Setting Response Headers**

|  |  |
| --- | --- |
| Method | Description |
| **setHeader()** | * Set the value of a particular header * Syntax: **response.setHeader(String headerName, String headerValue)** |
| **setDateHeader()** | * Specialized methods to set headers that contain dates. Translating a Java date in milliseconds into a GMT time string * Syntax: **response.setDateHeader(String header, long milliseconds)** |
| **setIntHeader()** | * Specialized methods to set headers that contain integers. Converting an int to a String before inserting it into a header * Syntax: **response.setDateHeader(String header, long milliseconds)** |
| **setContentType()** | * Syntax: **response.setContentType(String mimeType)** |
| **setContentLength()** | * Syntax: **response.setContentLength(int length)** |
| **addCookie()** | * Syntax: **response.addCookie(Cookie c)** |
| **sendRedirect()** | * Redirect browser to another URL * Syntax: **response.sendRedirect(String address)** |

**6.3 Building Excel Spreadsheets**

|  |
| --- |
| ...  public class ApplesAndOranges extends HttpServlet {  public void doGet(HttpServletRequest request, HttpServletResponse response)  throws ServletException, IOException {    **response.setContentType("application/vnd.ms-excel");**  PrintWriter out = response.getWriter();  out.println("\tQ1\tQ2\tQ3\tQ4\tTotal");  out.println("Apples\t78\t87\t92\t29\t=SUM(B2:E2)");  out.println("Oranges\t77\t86\t93\t30\t=SUM(B3:E3)");  }  } |