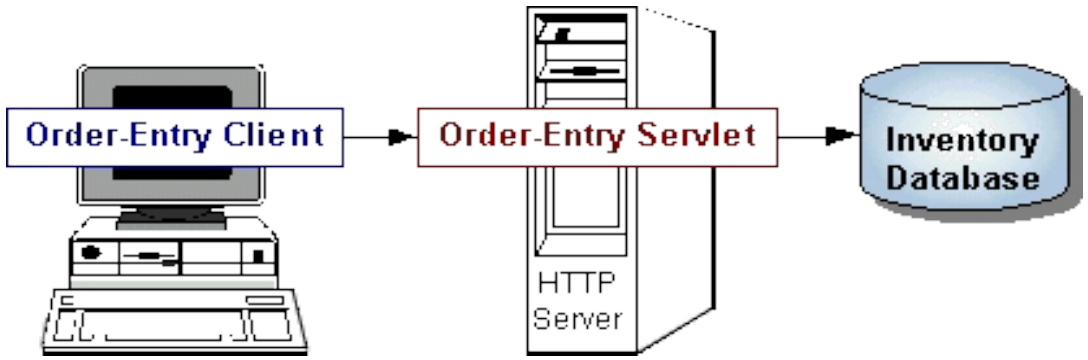
A crash course on Servlets

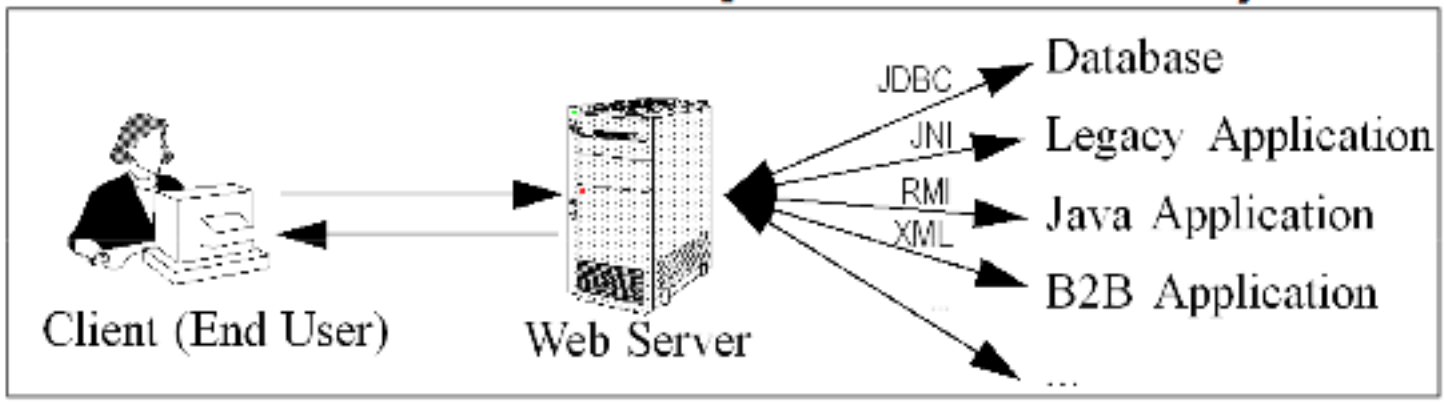
Servlets

Servlets are modules that extend Java-enabled web servers. For example, a servlet might be responsible for taking data in an HTML order-entry form and applying the business logic used to update a company's order database.

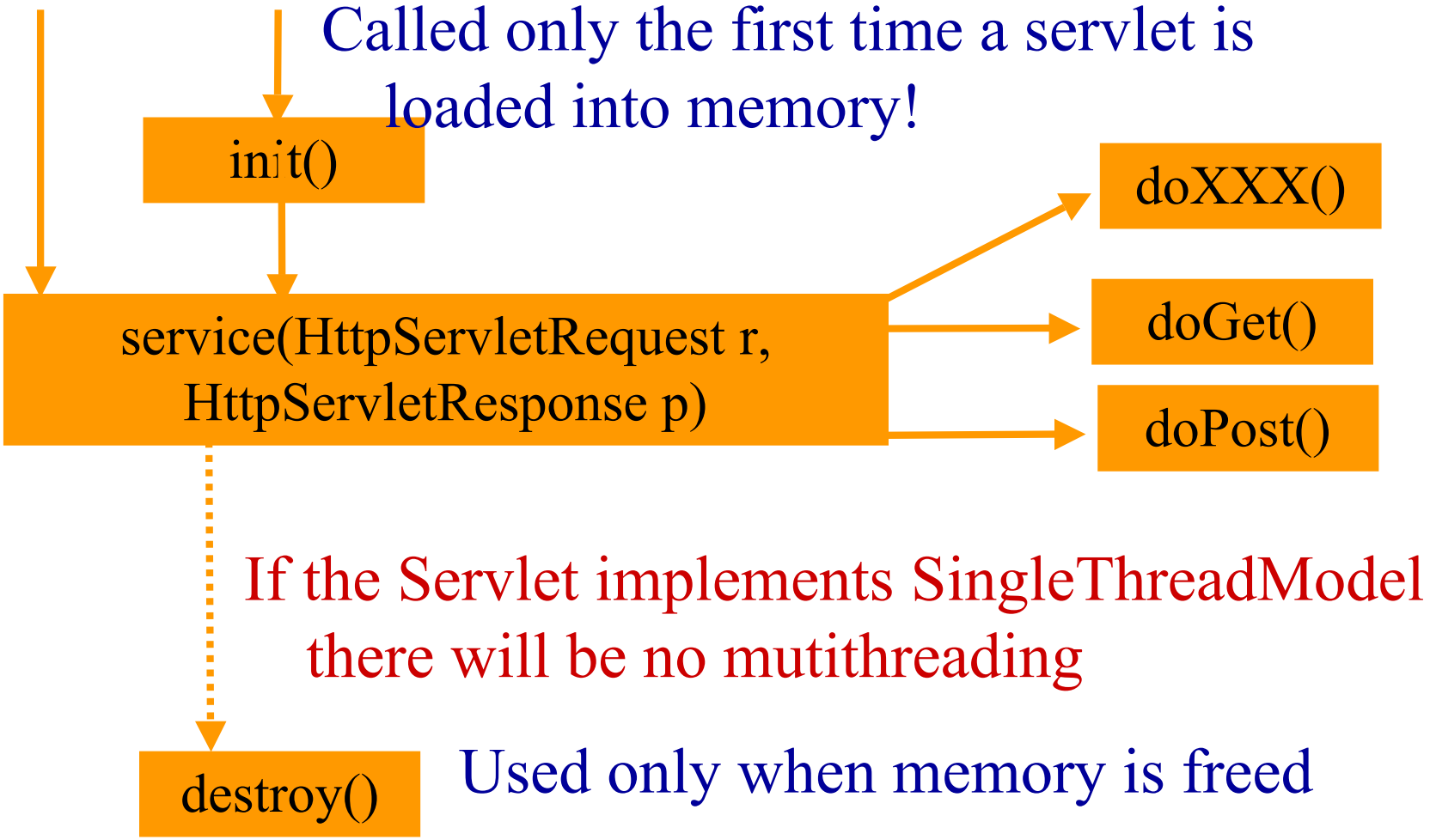


A Servlet's job

- **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)**



Servlet Lifecycle



Get vs Post

What are "Get" and "Post"?

Get and Post are methods used to send data to the server:
With the **Get** method, the browser appends the data onto the URL.
With the **Post** method, the data is sent as "standard input."

Why Do I Care?

It's important for you to know which method you are using. The Get method is the default, so if you do not specify a method, the Get method will be used automatically.

The Get method has several disadvantages:

- There is a limit on the number of characters which can be sent to the server, generally around 100 - 150 characters.
- Your user will see the "messy codes" when the data is sent.

service()

This code is part of the class HttpServlet

```
protected void service (HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException
    {
     String method = req.getMethod ();
     if (method.equals ("GET")) {
        long       ifModifiedSince; long       lastModified;   long      now;
        ifModifiedSince = req.getDateHeader ("If-Modified-Since");
        lastModified = getLastModified (req);
        maybeSetLastModified (resp, lastModified);
        if (ifModifiedSince == -1 || lastModified == -1) doGet (req, resp);
        else {
            now = System.currentTimeMillis ();
            if (now < ifModifiedSince || ifModifiedSince < lastModified)
                doGet (req, resp);
           else
               resp.sendError (HttpServletResponse.SC_NOT_MODIFIED);
        }
```

service()

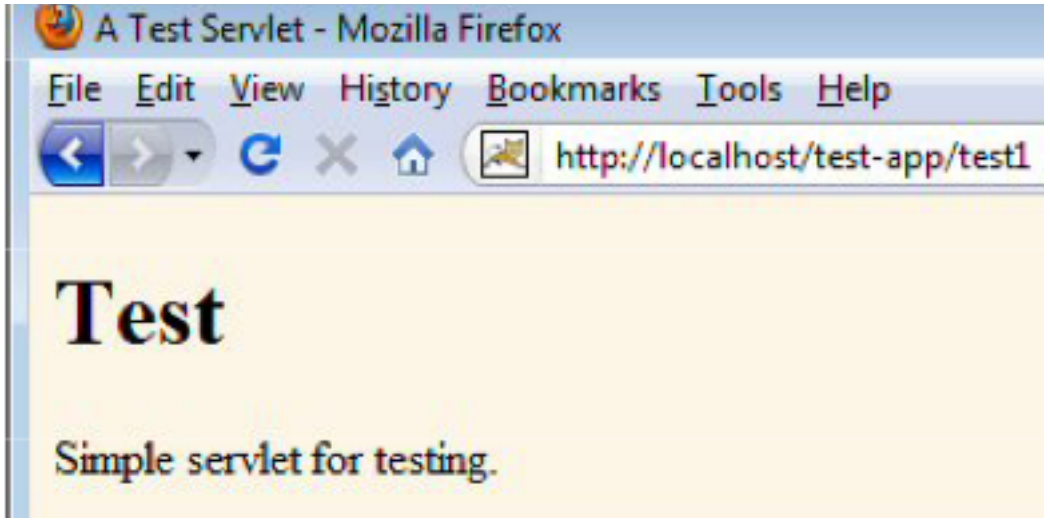
This code is part of the class HttpServlet

```
protected void service (HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException
    {
     String method = req.getMethod ();
     if (method.equals ("GET")) {
        long       ifModifiedSince; long       lastModified;   long      now;
        ifModifiedSince = req.getDateHeader ("If-Modified-Since");
        lastModified = getLastModified (req);
        maybeSetLastModified (resp, lastModified);
        if (ifModifiedSince == -1 || lastModified == -1) doGet (req, resp);
        else {
            now = System.currentTimeMillis ();
            if (now < ifModifiedSince || ifModifiedSince < lastModified)
                doGet (req, resp);
           else
               resp.sendError (HttpServletResponse.SC_NOT_MODIFIED);
        }
```

service()

```
    } else if (method.equals ("HEAD")) {
        long        lastModified;
        lastModified = getLastModified (req);
        maybeSetLastModified (resp, lastModified);
        doHead (req, resp);
    } else if (method.equals ("POST")) {
        doPost (req, resp);
    } else if (method.equals ("PUT")) {
        doPut(req, resp);
    } else if (method.equals ("DELETE")) {
        doDelete(req, resp);
    } else if (method.equals ("OPTIONS")) {
        doOptions(req,resp);
    } else if (method.equals ("TRACE")) {
        doTrace(req,resp);
    } else {
        resp.sendError (HttpServletResponse.SC_NOT_IMPLEMENTED,
            "Method '" + method + "' is not defined in RFC 2068");
    }
}
```

A taste of servlet programming




```
@WebServlet("/test1")
public class TestServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println
      ("<!DOCTYPE html>\n" +
       "<html>\n" +
       "<head><title>A Test Servlet</title></head>\n" +
       "<body bgcolor=\"#fdf5e6\">\n" +
       "<h1>Test</h1>\n" +
       "<p>Simple servlet for testing.</p>\n" +
       "</body></html>");
  }
```

Handling doPost

```
public void doPost (HttpServletRequest rq,
                                HttpServletResponse rp)
            throws ServletException,IOException
{
      doGet(rq,rp);
}
```

Configuring with web.xml

- **Java code**

```
package myPackage;  ...
public class MyServlet extends HttpServlet { ... }
```

- **web.xml entry (in <web-app...>...</web-app>)**
  - Give name to servlet

```
<servlet>
  <servlet-name>MyName</servlet-name>
  <servlet-class>myPackage.MyServlet</servlet-class>
</servlet>
```

  - Give address (URL mapping) to servlet

```
<servlet-mapping>
  <servlet-name>MyName</servlet-name>
  <url-pattern>/my-address</url-pattern>
</servlet-mapping>
```

- **Resultant URL**
  - http://*hostname*/*appName*/my-address

WebApps
(Tomcat configuration)

Static pages

To let Tomcat serve static pages, we must define a “Web Application”.

That is, in the Tomcat Document Root (by default *$CATALINA_HOME/webapps/*) we must create a folder named after our Web Application (e.g. myApp).

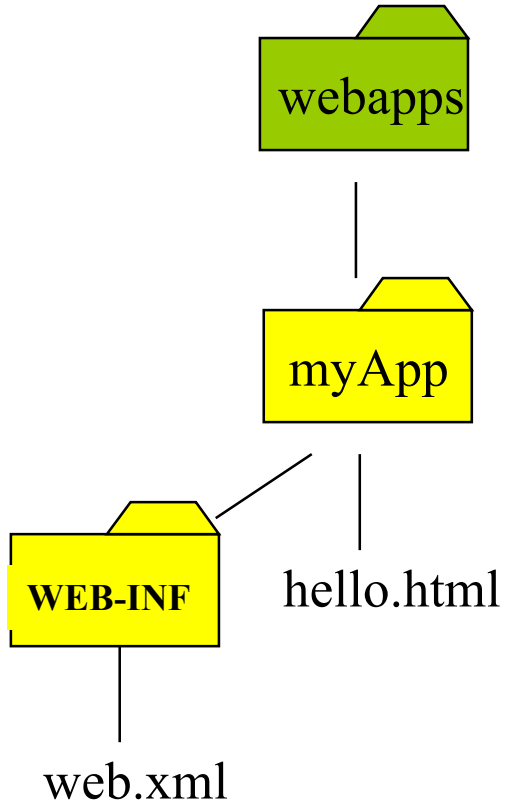
In that “myApp” folder, we MUST create a WEB-INF folder (that can be empy).

In the myApp folder we can then depost the static html files.

On our Tomcat server, the URL for the hello.html file becomes:

http://*machine/port*/myApp/hello.html

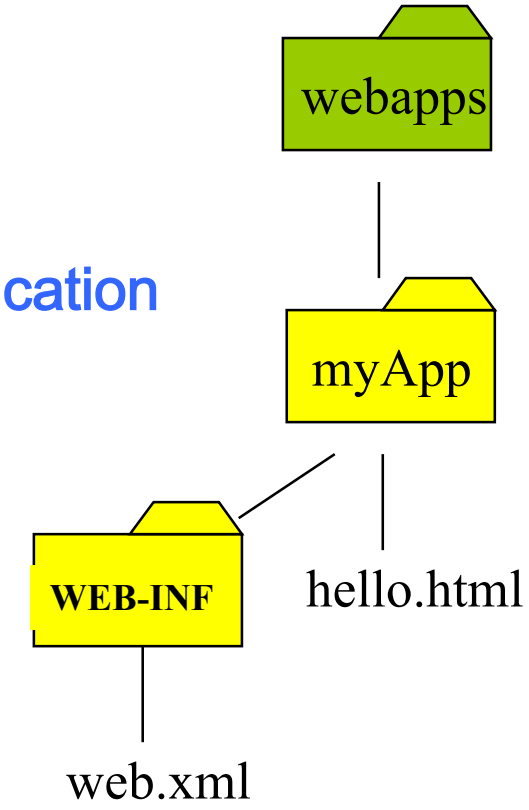
To actually see the webapp, we might have to restart Tomcat



Static pages

A web.xml file MUST be provided:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN“
    "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
</web-app>
```



Servlets

To let Tomcat serve servlet, we need add some info. The compiled servlets (.class) must be stored in a “classes” directory in WEB-INF.

Moreover, the web.xml file MUST contain at least:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <servlet-mapping>
        <servlet-name>invoker</servlet-name>
        <url-pattern>/magic/*</url-pattern>
    </servlet-mapping>
</web-app>
```

The “magic” word is the servlet activation keyword (you can of course customize this word). To execute the servlet called MyServlet.class, the URL will be:

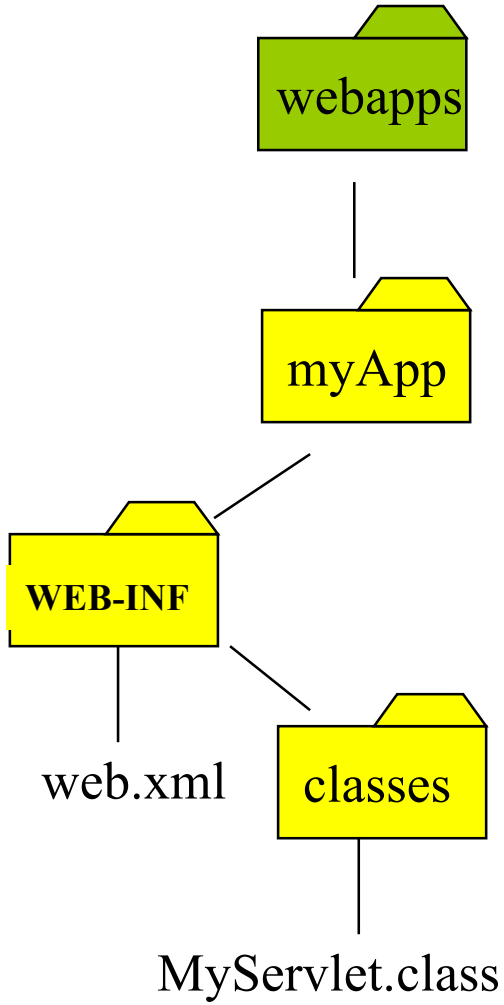
http://*machine/port*/myApp/magic/MyServlet

Servlets

The web.xml file CAN contain many additional info.
For instance, it can contain a section defining an alias name for the servlet:

```
…
   <servlet>
      <servlet-name>pippo</servlet-name>
      <servlet-class>Servlet1</servlet-class>
   </servlet>
…
```

In such case, the servlet called MyServlet.class
Can be activated ALSO by the URL:
http://*machine/port*/myApp/magic/pippo



Forms (a quick overview)

See also: http://www.cs.tut.fi/~jkorpela/forms/

Forms

Give to the user the possibility to di
send information to the Web server

The FORM tag defines a form and has the following attributes:

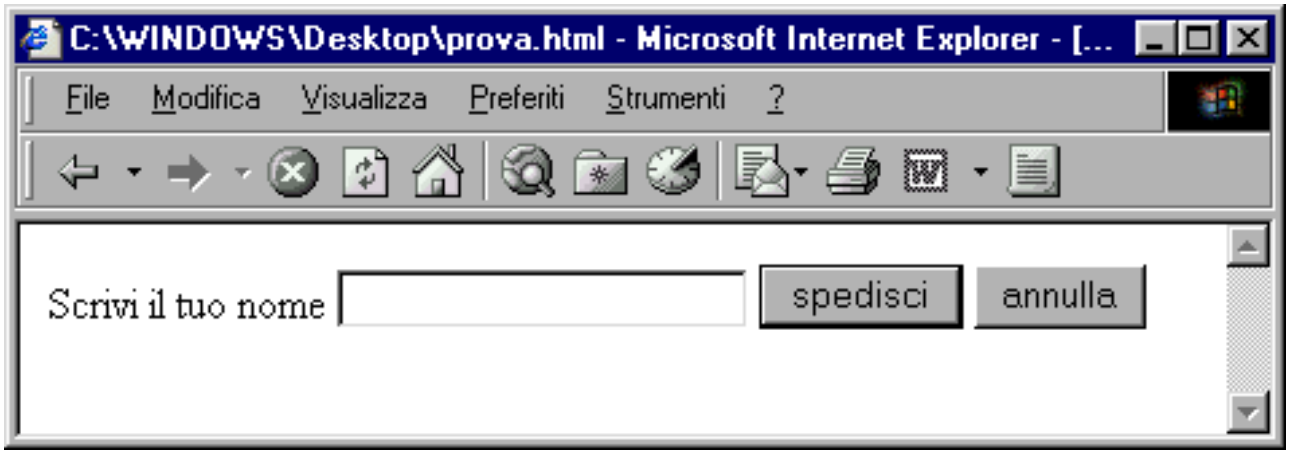
- ACTION identifies the processing engine
- ENCTYPE specificies the MIME type used to pass data to the server (Es. Text/html)

FORM contains the sub-tag:

- several tags for collecting data
- An INPUT tag must be of type SUBMIT for sending the data
- An INPUT can be of tye RESET to cancel all the gathered data

Form - input

```
<FORM method="POST" action="/cgi-bin/elabora">
   Scrivi il tuo nome
   <Input type="text" size“=25" maxlength="15“ name=“a”>
   <Input type="submit" value="spedisci">
   <Input type="reset" value="annulla">
</FORM>
```



Sends a url of type
http://…/cgi-bin/elabora?a=MarcoRonchetti&b=…

Reading parameters

- **request.getParameter("name")**
  - Returns 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 data
- **request.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
- **request.getParameterNames() or request.getParameterMap()**
  - Returns Enumeration or Map of request params
  - Usually reserved for debugging

Examples

For examples, see

http://courses.coreservlets.com/Course-Materials/pdf/csajsp2/03-Form-Data.pdf

Check for missing or wrong parameters!

1. Do not assume user will give the expected data
2. Do not show the user Java error messages!
   - Use default values
   - Redisplay the form

Tags for font aestetics

- **label**
  - If you use the label tag for prompts associated with fields, clicking on the label transfers focus to the input field
  - You can either use the "for" attribute or enclose the field within the label
    - <label for="fname">First name:</label>
      <input type="text" name="userFirstName" id="fname"/>
    - <label>First name:
      <input type="text" name="userFirstName"
      </label>
- **fieldset and legend**
  - Grouping all or part of a form inside fieldset draws attention to it and separates it from the rest of the page
  - Using style sheets for the legend is particularly useful

Tags for font aestetics

- **HTML**

```
<fieldset>
  <legend>ajax:updateField</legend>
  <form ...>
    <label for="f">Enter temperature in Fahrenheit:</label>
    <input type="text" id="f"/>
    <input type="button" id="convertButton" value="Convert"/>
    <hr width="500" align="left"/>
    <label for="c">Temperature in Celsius:</label>
    <input type="text" id="c"/>
    <label for="k">Temperature in Kelvin:</label>
    <input type="text" id="k"/>
  </form>
</fieldset>
```

- **CSS**

```
legend {
  font-weight: bold;
  color: black;
  background-color: white;
  border: 1px solid #cccccc;
  padding: 4px 2px;
```

HTTP Header & Status code

Request and response

- **Request**

GET /servlet/*SomeName* HTTP/1.1
Host: ...
***Header2*: ...**
...
***HeaderN*:**
(Blank Line)

- **Response**

HTTP/1.1 200 OK
Content-Type: text/html
***Header2*: ...**
...
***HeaderN*: ...**
(Blank Line)
<!DOCTYPE ...>
<HTML>
<HEAD>...</HEAD>
<BODY>
...
</BODY></HTML>

HTTP Header

```
GET /search-servlet?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/4.0 (compatible; MSIE 6.0; Windows N
```

Reading HTTP header

- **General**
  - getHeader (header name is not case sensitive)
  - getHeaders
  - getHeaderNames
- **Specialized**
  - getCookies
  - getAuthType and getRemoteUser
  - getContentLength
  - getContentType
  - getDateHeader
  - getIntHeader
- **Related info**
  - getMethod, getRequestURI , getQueryString, getProtocol

Resist!

Resist the temptation to adapt your page to the user's agent (i.e. the browser!)

Remember: headers can be faked!

Example

See

http://courses.coreservlets.com/Course-Materials/pdf/csajsp2/04-Request-Headers.pdf

For an example of a choice based on the HTTP header (wheater to send compressed or uncompressed data)

Common status codes

- **200 (OK)**
  - Everything is fine; document follows.
  - Default for servlets.
- **204 (No Content)**
  - Browser should keep displaying previous document.
- **301 (Moved Permanently)**
  - Requested document permanently moved elsewhere (indicated in Location header).
  - Browsers go to new location automatically.
  - Browsers are technically supposed to follow 301 and 302 (next page) requests only when the incoming request is GET, but do it for POST with 303. Either way, the Location URL is retrieved with GET.

Common status codes

- **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)**
  - No such page. Servlets should use sendError to set this.
  - Problem: Internet Explorer and small (< 512 bytes) error pages. IE ignores small error page by default.
  - Fun and games: http://www.plinko.net/404/

How to set the status code

- **response.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.
- **response.sendError(int code, String message)**
  - Wraps message inside small HTML document
- **response.sendRedirect(String url)**
  - Sets status code to 302
  - Sets Location response header also

Setting the response header

- **setContentType**
  - Sets the Content-Type header.
  - Servlets almost always use this.
  - See table of common MIME types.
- **setContentLength**
  - Sets the Content-Length header.
  - Used for persistent HTTP connections.
  - See Connection request header.
- **response.setHeader(String headerName, String headerValue)**
  - Sets an arbitrary header
- **response.setDateHeader(String name, long millisecs)**
  - Converts milliseconds since 1970 to a date string in GMT format

Mimetypes

Type	Meaning
application/msword	Microsoft Word document
application/octet-stream	Unrecognized or binary data
application/pdf	Acrobat (.pdf) file
application/postscript	PostScript file
application/vnd.ms-excel	Excel spreadsheet
application/vnd.ms-powerpoint	Powerpoint presentation
application/x-gzip	Gzip archive
application/x-java-archive	JAR file
application/x-java-vm	Java bytecode (.class) file
application/zip	Zip archive
audio/basic	Sound file in .au or .snd format
audio/x-aiff	AIFF sound file
audio/x-wav	Microsoft Windows sound file
audio/midi	MIDI sound file
text/css	HTML cascading style sheet
text/html	HTML document
text/plain	Plain text
text/xml	XML document
image/gif	GIF image
image/jpeg	JPEG image
image/png	PNG image
image/tiff	TIFF image
video/mpeg	MPEG video clip
video/quicktime	QuickTime video clip

Mimetypes

See examples in

http://courses.coreservlets.com/Course-Materials/pdf/csajsp2/06-Response-Headers.pdf

- How to generate an excel sheet
- How to generate a jpeg on the flight

Cookies

Writing cookies

- **Create a Cookie object.**
  - Call the Cookie constructor with a cookie name and a cookie value, both of which are strings.

```
Cookie c = new Cookie("userID", "a1234");
```

- **Set the maximum age.**
  - To tell browser to store cookie on disk instead of just in memory, use setMaxAge (argument is in seconds)

```
c.setMaxAge(60*60*24*7); // One week
```

- **Place the Cookie into the HTTP response**
  - Use response.addCookie.
  - If you forget this step, no cookie is sent to the browser!

```
response.addCookie(c);
```

Reading cookies

- **Call request.getCookies**
  - This yields an array of `Cookie` objects.
- **Loop down the array, calling getName on each entry until you find the cookie of interest**
  - Use the value (getValue) in application-specific way.

```
String cookieName = "userID";
Cookie[] cookies = request.getCookies();
if (cookies != null) {
  for(Cookie cookie: cookies) {
    if (cookieName.equals(cookie.getName())) {
      doSomethingWith(cookie.getValue());
    }
  }
}
```

Setting cookie properties

- **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. If you set the value of an incoming cookie, you still have to send it back out with response.addCookie.

setAge

setAge(x); x>0 set Time To Live (in sec)

setAge(0); tell the browser to delete the cookie

setAge(-1); use a session cookie

Replace a cookie value

- **Replacing a cookie value**
  - Send the same cookie name with a different cookie value
  - Reusing incoming Cookie objects
    - Need to call response.addCookie; merely calling setValue is not sufficient.
    - Also need to reapply any relevant cookie attributes by calling setMaxAge, setPath, etc.—cookie attributes are not specified for incoming cookies.
    - Usually not worth the bother, so new Cookie object used

Sessions

Home-made sessions

- **Idea: associate cookie with data on server**

```
String sessionID = makeUniqueString();
HashMap sessionInfo = new HashMap();
HashMap globalTable = findTableStoringSessions();
globalTable.put(sessionID, sessionInfo);
Cookie sessionCookie =
  new Cookie("JSESSIONID", sessionID);
sessionCookie.setPath("/");
response.addCookie(sessionCookie);
```

- **Still to be done:**
  - Extracting cookie that stores session identifier
  - Setting appropriate expiration time for cookie
  - Associating the hash tables with each request
  - Generating the unique session identifiers

Java HttpSessions

- **Access the session object**
  - Call request.getSession to get HttpSession object
    - This is a hashtable associated with the user
- **Look up information associated with a session.**
  - Call getAttribute on the HttpSession object, cast the return value to the appropriate type, and check whether the result is null.
- **Store information in a session.**
  - Use setAttribute with a key and a value.
- **Discard session data.**
  - Call removeAttribute discards a specific value.
  - Call invalidate to discard an entire session.

Using sessions (with cookies)

```
HttpSession session = request.getSession();
synchronized(session) {
  SomeClass value =
    (SomeClass)session.getAttribute("someID");
  if (value == null) {
    value = new SomeClass(...);
  }
  doSomethingWith(value);
  session.setAttribute("someID", value);
}
```

Session methods

- **getAttribute**
  - Extracts a previously stored value from a session object. Returns null if no value is associated with given name.
- **setAttribute**
  - Associates a value with a name. Monitor changes: values implement HttpSessionBindingListener.
- **removeAttribute**
  - Removes values associated with name.
- **getAttributeNames**
  - Returns names of all attributes in the session.
- **getId**
  - Returns the unique identifier.

Session methods

- **isNew**
  - Determines if session is new to *client* (not to *page*)
- **getCreationTime**
  - Returns time at which session was first created
- **getLastAccessedTime**
  - Returns time at which 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 current session

Session methods

- **getAttribute**
  - Extracts a previously stored value from a session object. Returns null if no value is associated with given name.
- **setAttribute**
  - Associates a value with a name. Monitor changes: values implement HttpSessionBindingListener.
- **removeAttribute**
  - Removes values associated with name.
- **getAttributeNames**
  - Returns names of all attributes in the session.
- **getId**
  - Returns the unique identifier.

Using sessions (with URL rewriting)

- **Session tracking code:**
  - No change
- **Code that generates hypertext links back to same site:**
  - Pass URL through response.encodeURL.
    - If server is using cookies, this returns URL unchanged
    - If server is using URL rewriting, this appends the session info to the URL
    - E.g.:

```
String url = "order-page.html";
url = response.encodeURL(url);
```

- **Code that does sendRedirect to own site:**
  - Pass URL through response.encodeRedirectURL