11.1 Overview of Servlets

- servlet = compiled Java class
- Servlets
  - executed on server system
    - under control of Web server
  - managed by *servlet container*,
    - or *servlet engine*
  - called through HTML
  - receive requests and return responses
    - both supported by HTTP protocol
Servlets are used as alternatives to

- 1. CGI, and
- 2. Apache modules

When Web server receives request for servlet

- request passed to servlet container
- container
  - makes sure servlet loaded
  - calls it
- servlet call has two parameter objects
  - 1. w/ request
  - 2. for response
- When servlet finished
  - container
    - reinitializes itself
    - returns control to Web server
Servlet Advantages:

- Can be faster than CGI
  - because run in server process
- Have direct access to Java APIs
- continue to run
  - unlike CGI programs
  - can save state information
- Have usual benefits of Java
  - platform independence
  - ease of programming

11.2 Servlet Details

- All servlets are classes that either
  - implement Servlet interface, or
  - extend class that implements Servlet interface
- Servlet interface
  - provides interfaces for methods that
    - manage servlets
    - and their interactions with clients
The **Servlet** interface

- declares three methods
  - called by servlet container
    - “life-cycle method”
  - `init` –
    - initializes servlet
    - prepares it to respond to client requests
  - `service` –
    - controls how servlet responds to requests
  - `destroy` –
    - takes servlet out of service

- declares two methods
  - used by servlet:
    - `getServletConfig`
    - `getServletInfo`
      - allow servlet to return info about itself to clients
        - author, version #, etc.
Most user-written servlet classes

- extensions to
- HttpServlet
  - = extension of GenericServlet
  - implements ServletInterface

**TABLE 11.1**
Commonly Used Methods of HttpServlet

<table>
<thead>
<tr>
<th>Method</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>doGet</td>
<td>To handle HTTP GET requests</td>
</tr>
<tr>
<td>doPost</td>
<td>To handle HTTP POST requests</td>
</tr>
<tr>
<td>doPut</td>
<td>To handle HTTP PUT requests</td>
</tr>
<tr>
<td>doDelete</td>
<td>To handle HTTP DELETE requests</td>
</tr>
<tr>
<td>init</td>
<td>To initialize resources used by the servlet</td>
</tr>
<tr>
<td>destroy</td>
<td>To delete resources used by the servlet</td>
</tr>
<tr>
<td>getServletInfo</td>
<td>To allow the servlet to provide information about itself</td>
</tr>
</tbody>
</table>
Two other necessary interfaces:

- `ServletResponse`
  - encapsulate communications
  - client to server
- `ServletRequest`
  - encapsulate communications
  - server to client
  - Provides servlet access to `ServletOutputStream`

---

**HttpServlet**

- abstract class
- Extends `GenericServlet`
Every subclass of `HttpServlet`

- MUST override at least one of methods of `HttpServlet`
  
  `doGet`
  `doPost`
  `doPut`
  `doDelete`
  `init`
  `destroy`
  `getServletInfo`

* Called by the server

**doGet protocol**

- `protected void doGet(HttpServletRequest request
  HttpServletResponse response)
  throws ServletException,
  java.io.IOException`

- `ServletException thrown if GET request could not be handled`
- `doPost protocol similar`
Servlet output – HTML

- 1. Use `setContent_type` method of response object to set content type to `text/html`
  - `response.setContentType("text/html");`
- 2. Create a `PrintWriter` object with `getWriter` method of response object
  - `PrintWriter servletOut = response.getWriter();`
- Example – Respond to GET request with no data
- Show `tstGreet.html` and `Greeting.java`

FIGURE 11.1 Display of `tstGreet.html`
11.3 A Survey Example

- --> Show `conelec2.html` and its display (w/browser)
- **The servlet:**
  - To accumulate voting totals
    - must write file on server
file will be read and written as object
  • array of vote totals
  • Using `ObjectInputStream`
  • object of this class created with its constructor
  • passing object of class `FileInputStream`
    • whose constructor is called with file variable name as parameter
On input
- contents of file will be cast to integer array

```java
ObjectInputStream indat =
    new ObjectInputStream(
    new FileInputStream(File_variable_name));
```

Servlet
- must access form data from client
  - done with `getParameter` method of request object
  - passing literal string with name of form element
    - e.g., if form has an element named `zip`
      ```java
      zip = request.getParameter("zip");
      ```
If an element has no value

- and its value is requested by `getParameter`
- ➔ returned value is `null`

If form value is not a string

- ➔ returned string must be parsed to get value
- e.g., suppose value is integer literal
  - string that contains integer literal
  - can be converted to integer
  - with `parseInt` method of wrapper class for `int`,
    - `Integer`

- `price = Integer.parseInt(request.getParameter("price"));`
file structure

- array of 14 integers
  - 7 votes for females
  - 7 votes for males
- Servlet actions:
  - If votes data array exists
  - read votes array from data file
  - else
  - create votes array

- Get gender form value
- Get form value for new vote
  - convert to integer
- Add vote to votes array
- Write votes array to votes file
- Produce return HTML document
  - shows current results of survey
• Every voter get current totals
• --> Show servlet, Survey.java
• --> Show Figure 11.4

FIGURE 11.4
Results of the Survey servlet

Thank you for participating in the Consumer Electronics Survey

Current Survey Results:
For female respondents
Conventional TV: 1
HDTV: 2
VCR: 2
CD Player: 3
Mini CD player/recorder: 3
DVR player: 4
Other: 4

For male respondents
Conventional TV: 1
HDTV: 5
VCR: 0
CD Player: 2
Mini CD player/recorder: 1
DVR player: 2
Other: 0
11.4 Storing Information about Clients

- **session** =
  - collection of all requests made by particular browser
    - from time browser started
    - until user exits browser
  - HTTP protocol is stateless

- But, several reasons why useful for server to relate request to session
  - Shopping carts for many different simultaneous customers
  - Customer profiling for advertising
  - Customized interfaces for specific clients
Approaches to storing client information:

- Store it on server
  - too much to store!
- Store on client machine ➔ this works
- ➔ Cookies
  - = object sent by server to client

Every HTTP communication between browser and server

- includes
  - in header
  - information about message
- At time cookie created
  - given a lifetime
- Every time browser sends request to server that created cookie
  - while cookie still alive
  - cookie included
• browser can be set to reject all cookies
• cookie object has
  • Data members
    • store lifetime, name, and value
    • the cookies’ value
  • Methods:
    • setComment, setMaxAge, setValue,
      getMaxAge, getName, and getValue
• Cookies created with Cookie constructor

• Cookie newCookie = new Cookie(gender, vote);

cookie’s lifetime

• By default = current session
• setMaxAge to make longer
Attach cookie response

- with addCookie
- Order in which response must be built:
  - 1. Add cookies
  - 2. Set content type
  - 3. Get response output stream
  - 4. Place info in the response
- browser does nothing with cookies
  - other than
    - storing
    - passing back

Servlet gets cookie from browser

- with getCookies method
  - Cookie theCookies[];
    ...
    theCookies = request.get_cookies();

- Vote Counting Example
- → Show ballot.html and display
FIGURE 11.5 Display of ballot.html

Vote counting servlet activities:

- See if vote was cast
- Make sure voter hasn’t voted before
- Tally real votes and give client totals
- Store votes in file
- Show VoteCounter.java
**FIGURE 11.6** output of VoteCounter servlet for form with no vote

```
You submitted a ballot with no vote marked
Please mark the ballot and resubmit
```

**FIGURE 11.7** output of VoteCounter for form with second vote from same client

```
Your vote is illegal - you have already voted!
```
FIGURE 11.8 output of VoteCounter for form with legitimate vote

Session Tracking

- alternative to cookies
  - Use HttpSession object
    - can store list of names and values
Create Session object

- Put value in session object
  - with `putValue`
    - `mySession.putValue("iVoted", "true");`

- kill session with `invalidate` method
- remove value with `removeValue`
- get value with `getValue(name)`
- get all names of values with `getValueNames`

- SHOW `VoteCounter2.java`
11.5 Java Server Pages

- Motivation
  - Servlets require mixing of XHTML into Java
  - JSP mixes code into XHTML
    - can be in separate file
  - Servlets more appropriate
    - when most of document to be returned dynamically generated
  - JSP more appropriate
    - when most of document to be returned predefined

JSP Documents

- Converted to servlets
- Consist of four different kinds of elements:
  - 1. Directives – messages to JSP container
  - 2. XHTML or XML markup – “template text”
    - static part of document
  - 3. Action elements …
  - 4. Scriptlets
**Action elements**

- Dynamically create content
- Output of JSP document is combination of
  - template text, and
  - output of action elements

**Appear in three different categories:**

- 1. Standard
  - defined by JSP spec
  - limited scope and value
- 2. Custom
  - defined by organization for their particular needs
- 3. JSP Standard Tag Library (JSTL)
  - created to meet frequent needs not met by standard action elements
  - Consists of five libraries
Differences between

- JSTL action elements and programming language:
  - 1. syntax different
  - 2. action elements much easier to use than programming language

Directives

- Tags that use `<%@ and %> delimiters
- Most common:
  - page
    - specify attributes such as contentType
  - `<%@ page contentType = "text/html" %>`
  - taglib
    - specify library of action elements
  - `<%@ taglib prefix = "c" uri = "http://java.sun.com/jsp/jstl/core" %>`
Scriptlets

- Java code scripts
  - embedded in JSP documents
  - copied into output of JSP document

Four kinds of things can appear in scriptlet:

1. Comments (in Java form)
2. Scriptlet code (Java code in \(<% \ldots \%angle\) tag)
3. Expressions …
4. Declarations (not discussed here)
Expressions

• used to insert values into response
  • <%= expression %>
  • SHOW tempconvert0.html and tempconvert0.jsp

can be combined

• Need to be able to determine which call it is
  • One way:
    • use getParameter and test against null
  • SHOW tempconvert1.jsp
In JSP 1.1

- all dynamic parts were created with scriptlets
  - but → lots of Java in documents
  - → not better than servlets
- Since Expression Language and JSTL added to JSP
  - scriptlets no longer needed

JSP Expression Language

- Similar to expressions of JavaScript
- E.g.,
  - arithmetic between string and number
- no control statements
- Syntax: `_${expression}_`
- Consist of literals, arithmetic operators, implicit variables (for form data), and normal variables
- EL used to set attribute values of action elements (always strings)
EL data often comes from forms

- implicit variable, \texttt{param}, stores collection of all form data values
  - \texttt{$\{param.address\}$}
- If form data name has special characters:
  - \texttt{$\{param[\textquoteleft cust-address\textquoteright]\}$}

Another implicit variable:

- \texttt{pageContext}
- Has lots of info about request
  - e.g.,
    - contentType, contentLength, remoteAddr
- Output usually created with \texttt{out}
  - \texttt{\<c:out value = "$\{param.address\}$"/>}
- \rightarrow SHOW \texttt{tempconvert2.html} and \texttt{tempconvert2.jsp}
JSTL Control Action Elements

- Flow control elements
  - Core library of JSTL
- Selection
  - if element

Often used to choose

- whether first call of combined document
- `<c:if test = "${pageContext.request.method == 'POST'}">` ...
  `</c:if>`
- ➔ SHOW tempconvert3.jsp
FIGURE 11.9 listing of initial version of tempconvert3.jsp

FIGURE 11.10 listing of tempconvert3.jsp after submitting original
Loops

- `forEach` element (an iterator)
- Often used for checkboxes and menus
  - to determine values of parts
- `parmValues` implicit variable
  - has array of values in checkboxes and menus

`forEach` has two attributes

- `items` and `var`
  - which get specific item and its value
- If we had a collection of checkboxes named `topping`
  - ...
<c:forEach items = "${paramValues.topping}' var = "top">
    <c:out value = "${top}" > <br />
</c:forEach>

forEach can also be used for counting loops

* <c:forEach begin = "1" end = "10">
    ...
    </c:forEach>
choose **element**

- to build switch constructs
  - choose
    - has no attributes
    - uses two other elements
      - when and otherwise
  - when
    - has test attribute
      - has control expression
  - Radio buttons require switch construct
    - SHOW `testradio.jsp`