Servlets

Server-Side Programming

What is a Servlet

- A Java application that is being run by a Web server
- Can receive parameters in an HTTP request
- Generates an HTTP response

![Servlet Diagram]

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

public class TimeServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        Calendar cal = new GregorianCalendar();
        String username = req.getParameter("username");
        out.println("<html><head><title>Time</title></head>"
                  + "<body style="text-align:center">Hello " + username + "&lt;br/&gt;The time is:" + "&lt;span style="color:red">" + cal.get(Calendar.HOUR_OF_DAY) + ":" + cal.get(Calendar.MINUTE) + ":" + cal.get(Calendar.SECOND) + "</span>" + "</body>
                  + "</html>");
    }
}
```

http://localhost:8080/Examples/time/show?username=Homer

Hello Homer.
The time is: 1:6:58
Reload/Refresh

- Trying to refresh the content created by a servlet will lead to fetching a new content from the server.
- This is not the case with static resources.
- Response headers of a static (as opposed to a servlet generated) resource contain Etag, Last-Modified.
- While trying to refresh a resource that is dynamically generated:
  - Cache-Control: max-age=0 is set, and that means the server/proxies will try to revalidate the resource.
  - Only in the static case the resource could be revalidated against some values the client holds.
  - So in the static case the client sends the Etag value attached to the If-None-Match header, and the Last-Modified value is sent in If-Modified-Since.

The Servlet Interface

- Java provides the interface Servlet.
- Specific Servlets implement this interface.
- Whenever the Web server is asked to invoke a specific Servlet, it activates the method service() of an instance of this Servlet.

Servlet Hierarchy

- Called by the servlet container to allow the servlet to respond to any request method
  - service(ServletRequest, ServletResponse)
- A generic, protocol-independent class, implementing Servlet
  - HttpServlet
- Called by the servlet container for allowing the servlet to respond to a specific request method
  - doGet(HttpServletRequest, HttpServletResponse)
  - doPost(HttpServletRequest, HttpServletResponse)
  - doPut
  - doTrace

Class HttpServlet

- Class HttpServlet handles requests and responses according to the HTTP protocol.
- The service() method of HttpServlet checks the request method and calls the appropriate HttpServlet method:
  - doGet, doPost, doPut, doDelete, doTrace, doOptions or doHead.
Creating a Servlet

- Extend the class `HTTPServlet`
- Implement `doGet` or `doPost` (or both; also maybe others...)
- Both methods get:
  - `HttpServletRequest`: methods for getting form (query) data, HTTP request headers, etc.
  - `HttpServletResponse`: methods for setting HTTP status codes, HTTP response headers, and get an output stream used for sending data to the client
- Many times, we implement `doPost` by calling `doGet`, or vice-versa

```
public class TextHelloWorld extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        PrintWriter out = res.getWriter();
        out.println("Hello World");
    }
     public void doPost(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        doGet(req, res);
    }
}
```

HelloWorld.java

Getting HTTP Headers

- Values of the HTTP request can be accessed through the `HttpServletRequest` object
- Get the value of the header `hdr` using `getHeader("hdr")` of the request argument
- Get all header names: `getHeaderNames()`
- Methods for specific request information: `getCookies`, `getContentType`, `getMethod`, `getProtocol`, etc.

```
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("<html><head><title>");
        out.println( title + "</title></head><body>");
        out.println("<h1>");
        out.println("Request Method: ");
        out.println(request.getMethod());
        out.println("</h1>");
        out.println("Request URI: ");
        out.println(request.getRequestURI());
        out.println("</h1>");
        out.println("Servlet Path: ");
        out.println(request.getServletPath());
        out.println("</h1>");
        out.println("Request Protocol: ");
        out.println(request.getProtocol());
        out.println("</h1>");
        Enumeration headerNames = request.getHeaderNames();
        while (headerNames.hasMoreElements()) {
            String headerName = (String) headerNames.nextElement();
            out.println("<tr><td>");
            out.println("<td>");
            out.println(request.getHeader(headerName));
            out.println("</td></tr>");
        }
        out.println("</table><hr><body></html>".toString());
    }
     public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        doGet(request, response);
    }
}
```

ShowRequestHeaders.java
User Input in HTML

• Using HTML forms, we can pass parameters to Web applications
• \(<form \text{ action=... method=...} > \ldots <\!\text{/form}>\) comprises a single form
  • action: the address of the application to which the form data is sent
  • method: the HTTP method to use when passing parameters to the application (e.g. \text{get} or \text{post})

The \texttt{<input>} Tag

• Inside a form, INPUT tags define fields for data entry
• Standard input types include: \texttt{buttons}, \texttt{checkboxes}, \texttt{password fields}, \texttt{radio buttons}, \texttt{text fields}, \texttt{image buttons}, \texttt{text areas}, \texttt{hidden fields}, etc.
• Each one associates a single (string) value with a named parameter

Getting the Parameter Values

• To get the (first) value of a parameter named \texttt{x}
  • \texttt{req.getParameter("x")}
  where \texttt{req} is the service request argument
• If there can be multiple values for the parameter, an array (or a Map) is returned
  • \texttt{req.getParameterValues("x")}
• To get parameter names
  • \texttt{req.getParameterNames()}

---

\[
\begin{align*}
\text{User Input in HTML} \\
\text{• Using HTML forms, we can pass parameters to Web applications} \\
\text{• \(<form \text{ action=... method=...} > \ldots <\!\text{/form}>\) comprises a} \\
\text{single form} \\
\text{• action: the address of the application to which the form data is sent} \\
\text{• method: the HTTP method to use when passing parameters to the application (e.g. \text{get} or \text{post})} \\
\text{The \texttt{<input>} Tag} \\
\text{• Inside a form, INPUT tags define fields for data entry} \\
\text{• Standard input types include: \texttt{buttons}, \texttt{checkboxes}, \texttt{password fields}, \texttt{radio buttons}, \texttt{text fields}, \texttt{image buttons}, \texttt{text areas}, \texttt{hidden fields}, etc.} \\
\text{• Each one associates a single (string) value with a named parameter} \\
\text{Getting the Parameter Values} \\
\text{• To get the (first) value of a parameter named \texttt{x}} \\
\text{• \texttt{req.getParameter("x")}} \\
\text{where \texttt{req} is the service request argument} \\
\text{• If there can be multiple values for the parameter, an array (or a Map) is returned} \\
\text{• \texttt{req.getParameterValues("x")}} \\
\text{• To get parameter names} \\
\text{• \texttt{req.getParameterNames()}}
\end{align*}
\]
**Handling HTTP Response**

```java
public class SetColors extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String bg = request.getParameter("bgcolor");
        String fg = request.getParameter("fgcolor");
        String size = request.getParameter("size");

        out.println("<html><head><title>Set Colors Example</title></head>");
        out.println("<body style="color:" + fg + ";background-color:" + bg + ";font-size:" + size + "px">");
        out.println("<h1>Set Colors Example</h1>");
        out.println("<p>You requested a background color " + bg + "</p>" );
        out.println("<p>You requested a font color " + fg + "</p>" );
        out.println("<p>You requested a font size " + size + "</p>" );
        out.println("</body></html>" );
    }
}
```

**Setting the Response Status**

- Use the following `HttpServletResponse` methods to set the response status:
  - `setStatus(int sc)`  
    - Use when there is no error, like 201 (created)  
    - No need to send 200 OK explicitly...  
  - `sendError(int sc)`  
    - Use in erroneous situations, like 400 (bad request)  
    - The server may return a formatted message  
  - `sendRedirect(String location)`  
    - As opposed to forwarding which is done within the server side completely, on redirect the client gets the "Location" header and a special code (302) and sends another request to the new location

**Setting Response Headers**

- Use the following `HttpServletResponse` methods to set the response headers:
  - `setHeader(String hdr, String value)`,  
    - If a header with the same name exists, it is overridden.  
  - `addHeader(String hdr, String value)`,  
    - The header is added even if another header with the same name exists.

**Specific Response Headers**

- Class `HttpServletResponse` provides setters for some specific headers:
  - `setContentType`  
  - `setContentLength`  
    - automatically set if the entire response fits inside the response buffer  
  - `setDateHeader`  
  - `setCharacterEncoding`
**The Response Content Buffer**
- The response body is buffered.
- Data is sent to the client when the buffer is full or the buffer is explicitly flushed.
- Once the first data chunk is sent to the client, the response is committed.
  - You cannot set the response line nor change the headers. Such operations are either ignored or cause an exception to be thrown.

**Buffer Related Methods**
- `setBufferSize`, `getBufferSize`
  - What are the advantages of using big buffers? what are the disadvantages?
- `flushBuffer`
- `resetBuffer`
  - Clears the unsent body content.
- `reset`
  - Clears any data that exists in the buffer as well as the status code and headers (if not yet sent).
- `isCommitted`

**Servlet Life Cycle**
- When a URL that a servlet is mapped to is requested, the server loads the Servlet class and initializes one instance of it.
- Each client request is handled by the Servlet instance in a separate thread.
  - The server can remove the Servlet.
  - The Servlet can remain loaded to handle additional requests.

**Servlet Life Cycle**
- When the Servlet is instantiated, its method `init()` is invoked (in our case, by Tomcat).
  - External parameters can be provided.
- Upon a request, its method `service()` is invoked.
- Before the Servlet removal, its method `destroy()` is invoked.

**Servlet Life Cycle**
- Calling the `init` method.
- Deal with requests: call the `service` method.
- Destroy the Servlet: call the `destroy` method.
  - Servlet Class
  - Servlet Instance
  - ServletConfig
  - Garbage Collection
  - In our case by servlet we refer to any class extending HttpServlet.
Initializing Servlets

- The method `init` has a parameter of type `ServletConfig`.
- `ServletConfig` has methods to get external initialization parameters (`getInitParameter()`).
- In Tomcat, these parameters are set in `web.xml`.
- To make initializations, override `init()` and not `init(ServletConfig)`.
- The former is automatically called by the latter after performing default initializations.

If we use `init()`, how can we obtain a reference to the `ServletConfig`?

```
Servlet.getServletConfig()
```

A web.xml Example

```
<web-app>
    ...
    <servlet>
        <servlet-name>InitExample</servlet-name>
        <servlet-class>ServletInit</servlet-class>
        <init-param>
            <param-name>login</param-name>
            <param-value>Homer</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    ...
</web-app>
```

Servlet Context

- Allows servlets that belong to the same application to share data and communicate.
- For instance, can be used to store details for a JDBC connection (details that are shared by different servlets).
- Allows setting and getting attributes.
- Provides information about the server.
- Can be used for writing messages to a log file.

Loading a Servlet on Startup

- A Servlet is usually loaded when it is first being called.
- You can set Tomcat to load a specific Servlet on startup in the Servlet declaration inside `web.xml`.

```
<web-app>
    ...
    <servlet>
        <servlet-name>InitExample</servlet-name>
        <servlet-class>ServletInit</servlet-class>
        <init-param>
            <param-name>login</param-name>
            <param-value>Homer</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    ...
</web-app>
```
A web.xml Example

```xml
<web-app>
  <context-param>
    <param-name>db-server</param-name>
    <param-value>ibm200.cs.technion.ac.il</param-value>
  </context-param>
  <servlet>
    <servlet-name>InitExample</servlet-name>
    <servlet-class>ServletInit</servlet-class>
    <init-param>
      <param-name>login</param-name>
      <param-value>Homer</param-value>
    </init-param>
  </servlet>
</web-app>
```

Destroying Servlets

- The server may remove a loaded Servlet
  - asked to do so by an administrator (e.g. Server shutdown)
  - Servlet was idle for a long time
  - server needs to free resources
- The server removes a Servlet only if all threads have finished or a grace period has passed
- Before removing, calls the destroy() method
  - can perform cleanup, e.g., close database connections
- Is it possible for the Servlet to end without its destroy being called?
  - You can do it if you kill the process explicitly

Thread Synchronization

- Multiple threads are accessing the same Servlet object at the same time
- Therefore, you have to deal with concurrency
- init() and destroy() are guaranteed to be executed only once (before/after all service executions)
- For servlets that implement the SingleThreadModel interface, it is guaranteed that no two threads will execute concurrently in the servlet’s service method (is it sufficient for handling concurrency correctly?)

Sessions

- HTTP is stateless, hence for implementing sessions we need to send back and forth a session identifier
- The session identifier is generated by the server and being sent to the client
- Each request that is part of the session is sent back to the server
- Sending session ids
  - By using cookies
  - URL rewriting
  - Hidden form fields

Session Objects

- Information related to a session is encapsulated in HttpSession objects
- Data is set and retrieved by setAttribute and.getAttribute
Links

- API: http://docs.oracle.com/javaee/6/api/