Servlets

Static Pages

Web browser

request

response

Web server
What is a Servlet?

- Servlets are Java programs that can be run dynamically in a Web Server
- Servlets are a server-side technology
- A Servlet is an intermediating layer between an HTTP request of a client and the Web server
What do Servlets do?

- Read data sent by the user (e.g., form data)
- Look up other information about request in the HTTP request (e.g. authentication data, cookies, etc.)
- Generate the results (may do this by talking to a database, file system, etc.)
- Format the results in a document (e.g., make it into HTML)
- Set the appropriate HTTP response parameters (e.g. cookies, content-type, etc.)
- Send the document to the user

Supporting Servlets

- The Web server must support Servlets (since it must run the Servlets):
  - Apache Tomcat
  - Sun’s Java System Web Server and Java System Application Server
  - IBM’s WebSphere Application Server
  - Allaire Jrun – an engine that can be added to IIS, PWS, old Apache Web servers etc...
  - Oracle Application Server
  - BEA WebLogic
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

- `Servlet`
  - `Generic Servlet`
    - `HttpServlet`
      - `YourOwnServlet`

```
service(ServletRequest, ServletResponse)

doGet(HttpServletRequest, HttpServletResponse)
doPost(HttpServletRequest, HttpServletResponse)
doPut

doTrace
...```

MyServlet
Class HttpServlet

- Class HttpServlet handles requests and responses of 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
- This class is abstract.

Creating a Servlet

- Extend the class HTTPServlet.
- Implement doGet or doPost (or both).
- 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.
- Usually implement doPost by calling doGet, or vice-versa.
Returning HTML

- By default a text response is generated (text/plain)
- In order to generate HTML
  - Tell the browser you are sending HTML, by setting the Content-Type header (text/html)
  - Modify the printed text to create a legal HTML page
- You should set all headers before writing the document content.

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

public class HelloWorld extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();

        out.println("<HTML><HEAD><TITLE>Hello World</TITLE></HEAD>"),
        out.println("<BODY><H1>Hello World</H1></BODY></HTML>");
        out.close();
    }
}
```
Configuring the Server

- After you code your servlet, you need to compile it to generate class files.
- When your Servlet classes are ready, you have to configure the Web server to recognize it.
- This includes:
  - Telling the Server that the Servlet exists
  - Placing the Servlet class file in a place known to the server
  - Telling the server which URL should be mapped to the Servlet
- These details are server specific.
The Big Picture

J2EE Enterprise Application (EAR file)

J2EE Web Component (WAR file)

Getting Information From the Request
Getting HTTP Data

- 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, getContentLength, getContentType, getMethod, getProtocol`, etc.

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

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>" + title + 
"</TITLE></HEAD>" + 
"<BODY BGCOLOR="#AACCAA" TEXT="#990000">\n" + 
"<H1 ALIGN=CENTE>" + 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 ALIGN=CENTER>\n" + 
"<TR BGCOLOR="#88AA88">\n" + 
"<TH>Header Name<TH>Header Value>\n";

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("</TABLE>\n</BODY></HTML>");

public void doPost(HttpServletRequest request, 
    HttpServletResponse response) 
    throws ServletException, IOException {
    doGet(request, response);
}
User Input in HTML

- Using HTML forms, we can pass parameters to web applications

```
<form action=… method=…> …</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. GET or POST)
GET Example

```html
<HTML>
  <FORM method="GET"
      action="http://www.google.com/search">
    <INPUT name="q" type="text">
    <INPUT type="submit">
    <INPUT type="reset">
  </FORM>
</HTML>
```

http://www.google.com/search?q=servlets

POST Example

```html
<HTML> <FORM method="POST"
      action="http://www.google.com/search">
    <INPUT name="q" type="text">
    <INPUT type="submit">
    <INPUT type="reset">
  </FORM> </HTML>
```

POST /search HTTP/1.1
Host: www.google.com

Content-type: application/x-www-form-urlencoded
Content-length: 10

q=servlets
Getting the Parameter Values

- To get the value of a parameter named \( x \):
  - \( \text{req.getParameter("x")} \)
    where \( \text{req} \) is the service request argument
- If there can be multiple values for the parameter:
  - \( \text{req.getParameterValues("x")} \)
- To get parameter names:
  - \( \text{req.getParameterNames()} \)

```
<HTML>
<HEAD>
    <TITLE>Sending Parameters</TITLE>
</HEAD>
<BODY BGCOLOR="#CC90E0">
<H1 ALIGN="LEFT">Please enter the parameters</H1>

<FORM ACTION="SetColors" METHOD="GET">
    <TABLE>
        <TR><TD>Background color:</TD><TD><INPUT TYPE="TEXT" NAME="bgcolor"></TD></TR>
        <TR><TD>Font color:</TD><TD><INPUT TYPE="TEXT" NAME="fgcolor"></TD></TR>
        <TR><TD>Font size:</TD><TD><INPUT TYPE="TEXT" NAME="size"></TD></TR>
    </TABLE>  <BR>
    <INPUT TYPE="SUBMIT" VALUE="Show Page">
</FORM>
</BODY>
</HTML>
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

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");
}
Set Colors Example

You requested a background color yellow
You requested a font color blue
You requested a font size 6
You don't have to do anything different to read POST data instead of GET data!!

```java
public void doPost(HttpServletRequest request,
                  HttpServletResponse response)
  throws ServletException, IOException {
    doGet(request, response);
}
```

```html
<FORM ACTION="SetColors"
       METHOD="POST"> ...
```

Creating the Response of the Servlet
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)
- `sendError(sc), sendError(sc, message)`
  - Use in erroneous situations, like 400 (bad request)
  - The server may return a formatted message
- `sendRedirect(String location)`
  - Redirect to the new location

Class `HttpServletRequest` has static integer variables for popular status codes for example:

- `SC_OK(200), SC_NOT_MODIFIED(304), SC_UNAUTHORIZED(401), SC_BAD_REQUEST(400)`
- Status code 200 (OK) is the default
Setting Response Headers

- Use the following HTTPServletResponse methods to set the response headers:
  - `setHeader(String hdr, String value)`, `setIntHeader(String hdr, int value)`
    - Override existing header value
  - `addHeader(String hdr, String value)`, `addIntHeader(String hdr, int value)`
    - The header is added even if another header with the same title 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`
More Header Methods

- `containsHeader(String header)`
  - Check existence of a header in the response
- `addCookie(Cookie)`
- `sendRedirect(String url)`
  - Automatically sets the Location header
- Do not write information to the response after `sendError` or `sendRedirect`

Reference

- Representation and Management of Data on the Internet (67633), Yehoshua Sagiv, The Hebrew University - Institute of Computer Science.