



Using Applets as Front Ends to Server-Side Programs

Originals of Slides and Source Code for Examples:

<http://courses.coreservlets.com/Course-Materials/java5.html>

Customized Java EE Training: <http://courses.coreservlets.com/>

Servlets, JSP, JSF 1.x & JSF 2.0, Struts Classic & Struts 2, Ajax, GWT, Spring, Hibernate/JPA, Java 5 & 6.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.



For live Java training, please see training courses at <http://courses.coreservlets.com/>. Servlets, JSP, Struts Classic, Struts 2, JSF 1.x, JSF 2.0, Ajax (with jQuery, Dojo, Prototype, Ext, etc.), GWT, Java 5, Java 6, Spring, Hibernate/JPA, and customized combinations of topics.



Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and this tutorial. Available at public venues, or customized versions can be held on-site at your organization. Contact hall@coreservlets.com for details.

Agenda

- Sending GET data and having the browser display the results
- Sending GET data and processing the results within the applet (HTTP tunneling)
- Using object serialization to exchange high-level data structures between applets and servlets
- Sending POST data and processing the results within the applet (HTTP tunneling)
- Bypassing the HTTP server altogether

4

Sending GET Request and Displaying Resultant Page

- Applet requests that browser display page – `showDocument`

```
try {  
    URL programURL =  
        new URL(baseUrl + "?" + someData);  
    getAppletContext().showDocument(programURL);  
} catch (MalformedURLException mue) { ... };
```

- URL-encode the form data

```
String someData =  
    name1 + "=" + URLEncoder.encode(val1) + "&" +  
    name2 + "=" + URLEncoder.encode(val2) + "&" +  
    ...  
    nameN + "=" + URLEncoder.encode(valN);
```

5

GET Request Example: Applet

```
public class SearchApplet extends Applet
                               implements ActionListener {
    ...
    public void actionPerformed(ActionEvent event) {
        String query =
            URLEncoder.encode(queryField.getText());
        SearchSpec[] commonSpecs =
            SearchSpec.getCommonSpecs();
        for(int i=0; i<commonSpecs.length-1; i++) {
            try {
                SearchSpec spec = commonSpecs[i];
                URL searchURL =
                    new URL(spec.makeURL(query, "10"));
                String frameName = "results" + i;
                getAppletContext().showDocument(searchURL,
                                                frameName);
            } catch(MalformedURLException mue) {}
        }
    }
}
```

6

GET Request Example: Utility Class

```
public class SearchSpec {
    private String name, baseURL, numResultsSuffix;

    private static SearchSpec[] commonSpecs =
        { new SearchSpec("google",
                        "http://www.google.com/search?q=",
                        "&num="),
        ... };

    public String makeURL(String searchString,
                          String numResults) {
        return(baseURL + searchString +
               numResultsSuffix + numResults);
    }
    ...
}
```

7

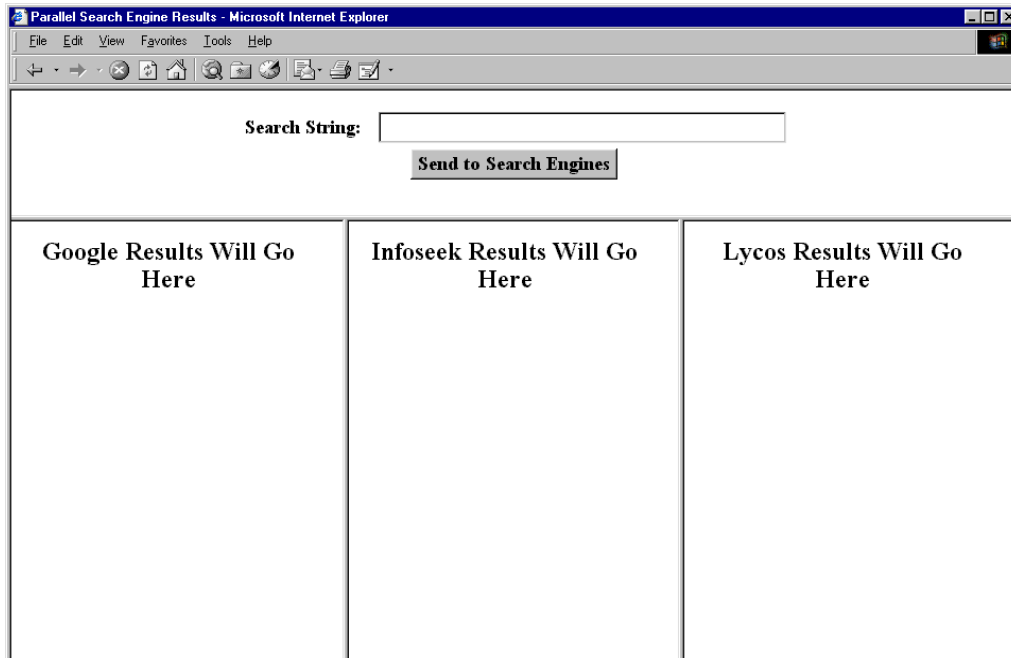
Get Request Example: HTML File

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN">
<HTML>
<HEAD>
  <TITLE>Parallel Search Engine Results</TITLE>
</HEAD>

<FRAMESET ROWS="120,*">
  <FRAME SRC="SearchAppletFrame.html" SCROLLING="NO">
  <FRAMESET COLS="*,*,*">
    <FRAME SRC="GoogleResultsFrame.html" NAME="results0">
    <FRAME SRC="InfoseekResultsFrame.html" NAME="results1">
    <FRAME SRC="LycosResultsFrame.html" NAME="results2">
  </FRAMESET>
</FRAMESET>
```

8

Get Request: Initial Result



9

GET Request: Submission Result



10

HTTP Tunneling

- **Idea**
 - Open a socket connection to port 80 on the server and communicate through HTTP
- **Advantages**
 - Communicate through firewalls
 - Server-side programs only needs to return the data, not a complete HTML document
- **Disadvantages**
 - Can only tunnel to server from which the applet was loaded
 - *Applet*, not *browser*, receives the response
 - Cannot easily display HTML

11

HTTP Tunneling and GET Requests

- **Create URL object referring to applet's host**
`URL dataURL = new URL(...);`
- **Create a URLConnection object**
`URLConnection connection = dataURL.openConnection();`
- **Instruct browser not to cache URL data**
`connection.setUseCaches(false);`
- **Set any desired HTTP headers**
- **Create an input stream**
 - Call `connection.getInputStream`; wrap in higher-level stream
- **Read data sent from server**
 - E.g., call `readLine` on `BufferedReader`
- **Close the input stream**

12

HTTP Tunneling Template: Client Side

```
URL currentPage = getCodeBase();
String protocol = currentPage.getProtocol();
String host = currentPage.getHost();
int port = currentPage.getPort();
String urlSuffix = "/servlet/SomeServlet";
URL dataURL = new URL(protocol, host, port, urlSuffix);
URLConnection connection = dataURL.getConnection();
connection.setUseCaches(false);
connection.setRequestProperty("header", "value");

BufferedReader in = new BufferedReader(
    new InputStreamReader(connection.getInputStream()));
String line;
while ((line = in.readLine()) != null) {
    doSomethingWith(line);
}
in.close();
```

13

Using Object Serialization with HTTP Tunneling

- **Idea**

- Server-side program (servlet) sends complete Java object
- Client-side program (applet) reads it

- **Client-side program (applet) template:**

```
ObjectInputStream in =  
    new ObjectInputStream(  
        connection.getInputStream());
```

```
SomeClass object = (SomeClass)in.readObject();  
doSomethingWith(object);
```

14

Using Object Serialization with HTTP Tunneling (Continued)

- **Server-side program (servlet) template:**

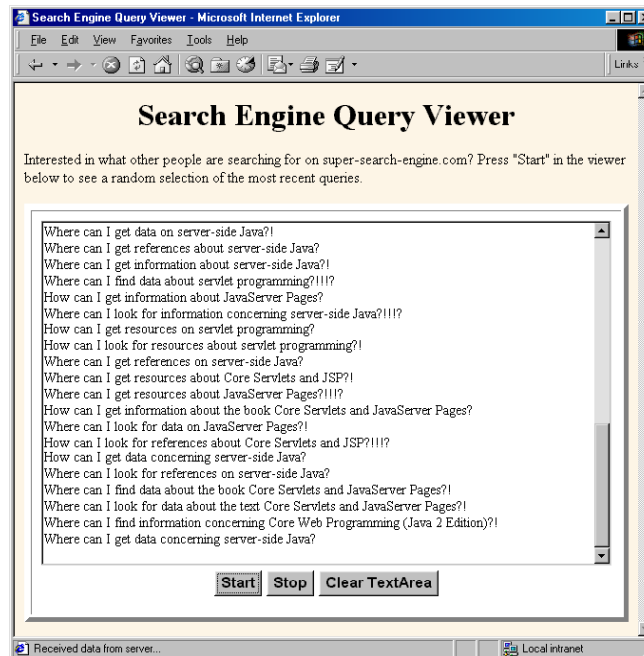
```
String contentType =  
    "application/x-java-serialized-object";  
response.setContentType(contentType);
```

```
ObjectOutputStream out =  
    new ObjectOutputStream(  
        response.getOutputStream());
```

```
SomeClass object = new SomeClass(...);  
out.writeObject(value);  
out.flush();
```

15

Example: Live Scrolling Data



16

Sending POST Data to Server

- Applet sends POST request to server
- Processes the response directly

```
Url currentPage = getCodeBase();
String protocol = currentPage.getProtocol();
String host = currentPage.getHost();
int port = currentPage.getPort();
String urlSuffix = "/servlet/SomeServlet";
URL dataURL = new URL(protocol, host, port,
    urlSuffix);
```

```
URLConnection connection =
    dataURL.openConnection();
connection.setUseCaches(false);
connection.setDoOutput(true);
```

17

Sending POST Data to Server (Continued)

- **Character or Binary Data**

```
ByteArrayOutputStream byteStream =
    new ByteArrayOutputStream(512);
PrintWriter out = new PrintWriter(byteStream, true);
out.print(data);
out.flush();

connection.setRequestProperty(
    "Content-Length",
    String.valueOf(byteStream.size()));
connection.setRequestProperty(
    "Content-Type",
    "application/x-www-form-urlencoded");
byteStream.writeTo(connection.getOutputStream());
```

18

Sending POST Data to Server

- **Serialized Data**

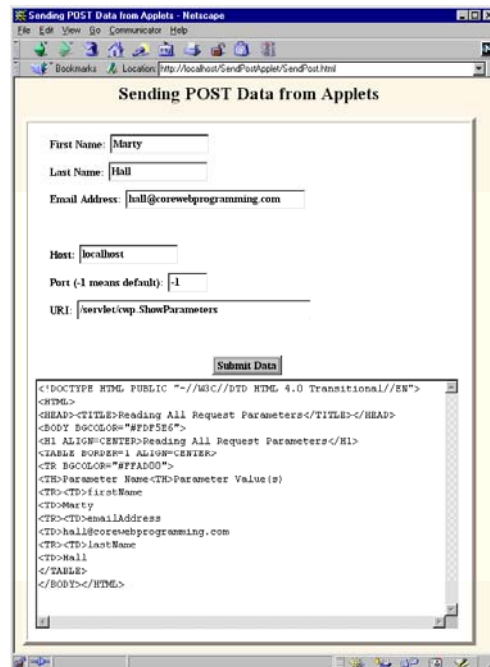
```
ByteArrayOutputStream byteStream =
    new ByteArrayOutputStream(512);
ObjectOutputStream out =
    new ObjectOutputStream(byteStream);
out.writeObject(data);
out.flush();

connection.setRequestProperty(
    "Content-Length",
    String.valueOf(byteStream.size()));
connection.setRequestProperty(
    "Content-Type",
    "application/x-java-serialized-object");
byteStream.writeTo(connection.getOutputStream());
```

19

Sending POST Data: Example

- Sends data to a servlet that returns an HTML page showing form data it receives
 - Displays result in an AWT TextArea



20

Bypassing the HTTP Server

- If you are using applets, you don't have to communicate via HTTP
 - JDBC
 - RMI
 - SOAP (perhaps via JAX-RPC)
 - Raw sockets
- Advantages
 - Simpler
 - More efficient
- Disadvantages
 - Can only talk to server from which applet was loaded
 - Subject to firewall restrictions
 - You have to have a second server running

21

Summary

- **Send data via GET and showDocument**
 - Can access any URL
 - Only browser sees result
- **Send data via GET and URLConnection**
 - Can only access URLs on applet's home host
 - Applet sees results
 - Applet can send simple data
 - Server can send complex data (including Java objects)
- **Send data via POST and URLConnection**
 - Can only access URLs on applet's home host
 - Applet sees results
 - Applet can send complex data (including Java objects)
 - Server can send complex data (including Java objects)
- **Bypass Web Server**

22

© 2009 Marty Hall



Questions?

Customized Java EE Training: <http://courses.coreservlets.com/>

Servlets, JSP, JSF 1.x & JSF 2.0, Struts Classic & Struts 2, Ajax, GWT, Spring, Hibernate/JPA, Java 5 & 6.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

23