

Servlets - Part 2

How Can a CGI Remember Something?

Suppose a CGI program has some data from the user during a request/response cycle – a username, or a cart item – and wants to have that information for future requests.

How can the CGI remember things for the future?

This hits at the basic lack-of-continuity problem of HTTP

Store it in a global variable?

No – the CGI quits after each transaction

Write it into a server side file?

Yes, that can work. Need to somehow keep this user's data separate from the other users.

Have The Client Store It

This is basically what hidden fields do – send the data back to the client and have them store it.

The client sends the data back (reminds) the server on the next request.

A cute finesse of the continuity problem.

There are other techniques that store things on the client side: URL re-writing and cookies.

We'll study those techniques soon, for now using hidden fields as the “store on client” example.

Client Data Problem

Storing the data on the client has the problem that as they use the back button, they are going back to earlier versions of the data.

e.g. cart stored entirely by hidden fields

Add 2 items to cart

Now hit back button to start where cart has 1 item

Reload at this point – they still just see the 1 item

Contrast to survey example – data on server side, so reload got up-to-date info

Solution: Session

Store the important information in a “session” on the server side. The session is like a hash-table on the server side.

The session has an arbitrary “id”, like “123xyz”, that identifies it.

Maybe just stored in the memory of the server

Maybe written to a “session” database, so even if the server crashes, or there are multiple servers, the session-id still works

Write back Session id

The only thing written back to the client is the session-id, for example, as a hidden field.

In this way, all the different pages of the session all “point” to the one version of the data in the server-side session object. The id is conceptually analogous to a pointer in C. Now, using the back button still works ok, since even the earlier pages have the right session id.

1. Session ID

Arbitrary identifier for session - “123xyz”

2. Server

Create an empty session when first see the client

Create an id

Server side - store the session under its id

Send the id to the client for storage (e.g. hidden field id=123xyz, or use cookie)

3. Client

Contact the server

Get back HTML etc. with the id in it (e.g. hidden field)

Send the info back to the server on each request

4. Server again

On later client request

See the id=123xyz binding from the client

Look up the appropriate session object

Look up or edit state in the session

e.g. Cart

Store the “cart” collection of items in the server side session

The client surfs through various product pages

When they click “buy”, the server adds that item to the session cart collection

The only state the client maintains is id=123xyz, so even if they use the back button and then start forward again, the server still pulls up the right cart info from the one, current state in the server side session

This is an example of the “never have two copies of things” rule - the one authoritative copy of the data is the session - all the client pages just point to that one copy.

e.g. Auth

Client logs in to the server with user= xxxx and password=xxx bindings

If the password is correct, server notes auth=true, and auth-time=current-time in the session

Server generates a random session id, and sends that back to the client

Any client request that comes in with the right id, we conclude is part of this authorized session

This works, since we only send the random id back to the client that had the right password. That's the only party that has the id

The id is a temporary shared secret - known only to the client and server.

Could have problems if a bad guy observes the transaction. Could use secure HTTP to avoid that problem.

Auth notes

Do not store auth=true on client side (what would be the problem with that strategy?)

Store id on client side

Server has the data, the client just points to it

Server can note the passage of time, and after, say, an hour delete the server session, or just set auth=false so that the client will be asked for their password again

Servlet Sessions

Servlet system can associate a "session" with a sequence of client requests

The servlet system can do this with cookies to store the id to the client, although you don't really need to know that detail yet.

The session object is like a hash table that persists from one request to the next

Store things you want to remember in it

request.getSession(boolean)

Check for/create a Session Object

getSession(false) checks for an existing session

getSession(true) checks, and then creates one if necessary

session.isNew()

True if the session is new - do one-time setup

session.getAttribute(key)

Returns the object stored in the session under the given String key, or null

The return type is Object - cast it to its proper type

session.setAttribute(key, value);

Install a key/value binding in the session

Song Example

Store a list of disliked words in the session

When asked for "favorite things", echo the song text, but beep out the disliked words

Song Code

```
// Song.java
/*
 * An example servlet that uses a session to store a list of
 * words built up by the client.
 * Echo the song with the words beeped out.
 */
import java.io.*;
import java.text.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

```
public class Song extends HttpServlet {

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

        out.println("<html>");
        out.println("<head>");

        String title = "Song";
        out.println("<title>" + title + "</title>");
        out.println("</head>");
        out.println("<body bgcolor=white>");

        out.println("<h1>" + title + "</h1>");

        // Retrieve session
        // true => create new session if none exists
        HttpSession session = request.getSession(true);

        // Check if this is the start of the session
        if (session.isNew()) {
            out.println("Hello there! I haven't seen you before have I? Creating a
session.");
            session.setAttribute("words", new ArrayList());
        }

        // invariant: at this point, the words ArrayList exists

        // Retrieve our arraylist from the session
        ArrayList words = (ArrayList) session.getAttribute("words");

        // Add button -> add word to list
        if (request.getParameter("add") != null) {
            String word = request.getParameter("word");
            if (word!=null && !word.equals("")) {
                words.add(word);
                out.println(word + " added");
            }
        }
        else if (request.getParameter("reset") != null) {
            words.clear();
        }

        // Print out current words
        out.println("<p>Words...");
        for (int i=0; i<words.size(); i++) {
            out.println((String)words.get(i));
        }

        if (request.getParameter("favorite") != null) {
            doFile(out, "favorite.txt", words);
        }

        // Print the form with the buttons last
        doForm(out);

        out.println("</body>");
        out.println("</html>");
    }
}
```

```
    }

    // Separated out utilities for the HTML generation...
    public void doForm(PrintWriter out) {
        out.println("<p><hr><form><input type=text name=word>");
        out.println("<input type=submit name=add value=\"Add Non-Favorite Word\">");
        out.println("<input type=submit name=reset value=\"Reset Words\">");
        out.println("<input type=submit name=favorite value=\"Show Favorite
Things\"></form>");
    }

    // Given a filename and words list, echo the file to out with
    // the given words beeped out.
    private void doFile(PrintWriter out, String fname, ArrayList words) {
        String text = readFile(fname);

        for (int i=0; i<words.size(); i++) {
            text = censor(text, (String)words.get(i));
        }

        out.println("<p>Favorite things...\n<pre>");
        out.println(text);
        out.println("</pre>");
    }

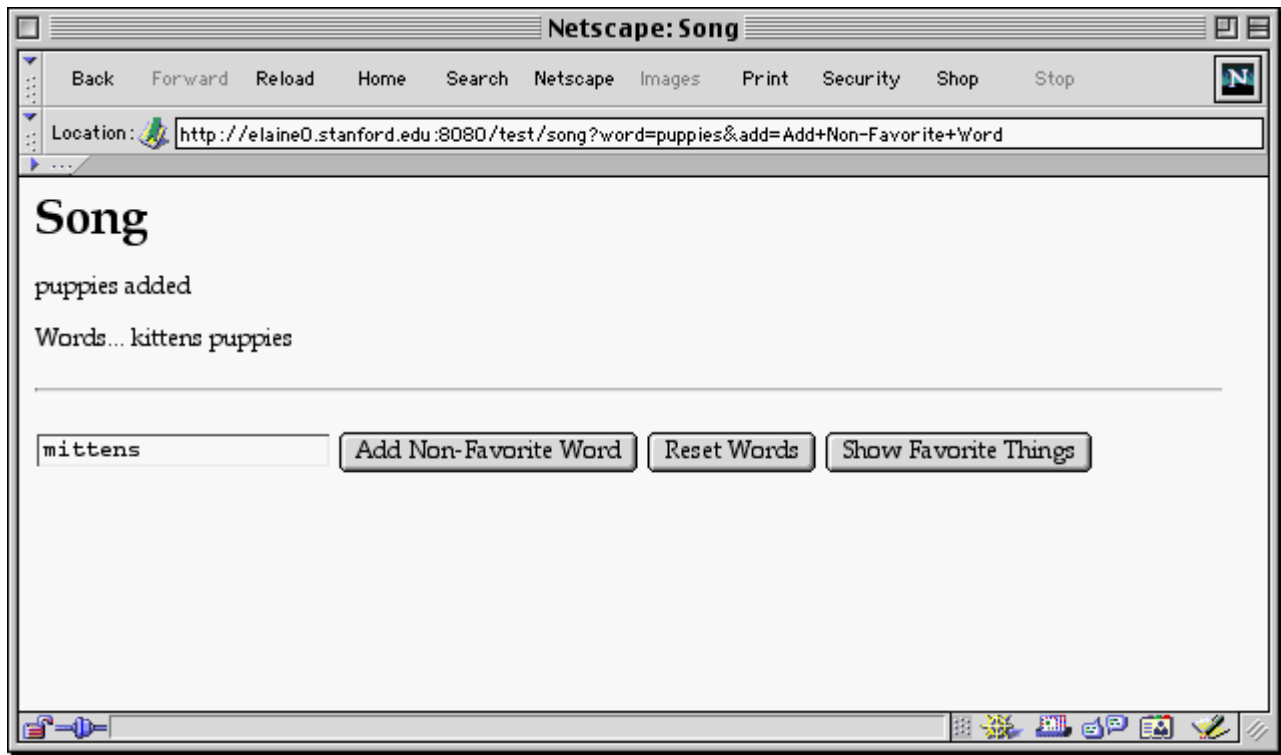
    // Given a filename, return its text as a single String
    private String readFile(String fname) {
        StringBuffer buff = new StringBuffer();
        try {
            // Build a reader on the fname, (also works with File object)
            BufferedReader in = new BufferedReader(new FileReader(fname));

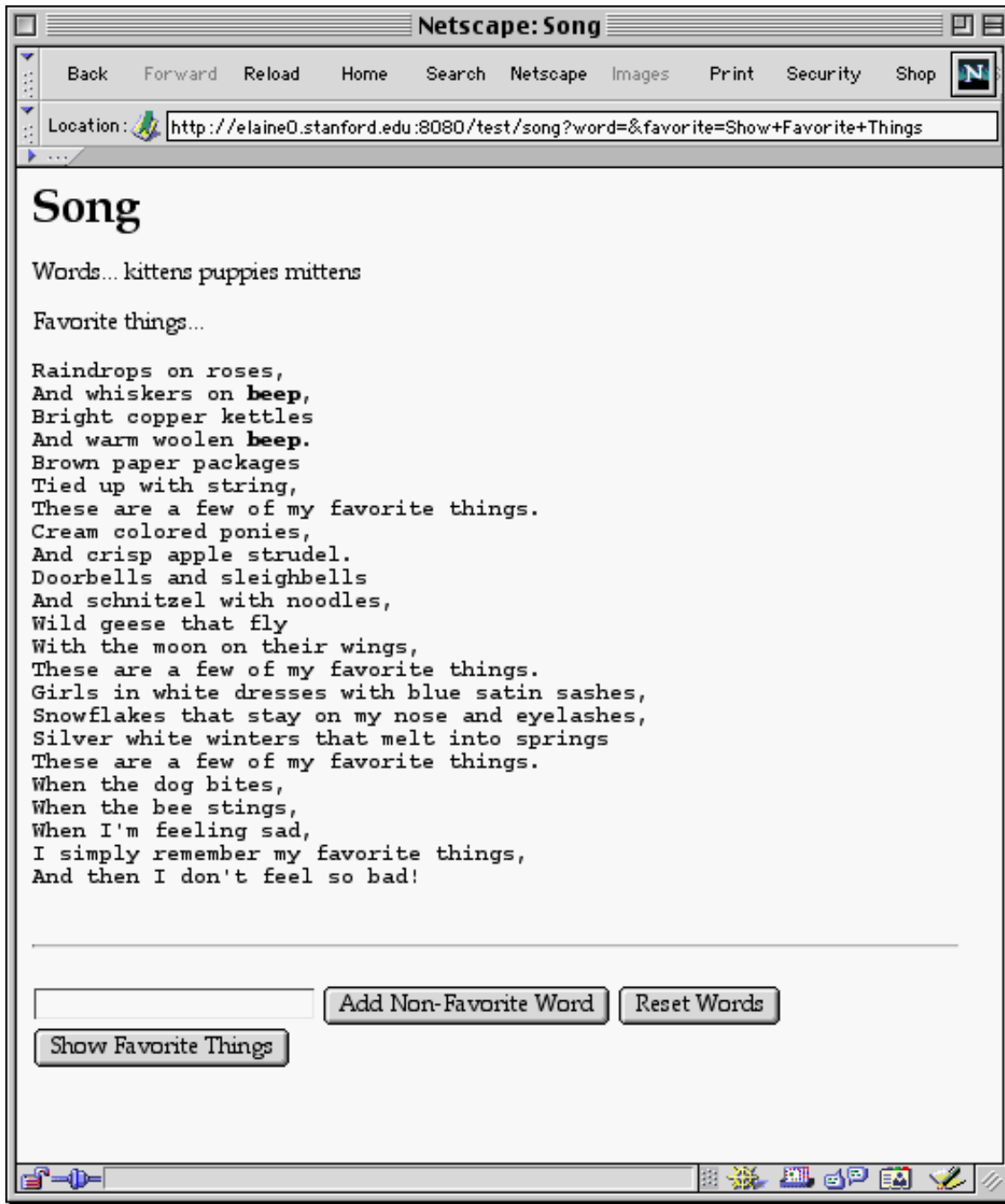
            String line;
            while ((line = in.readLine()) != null) {
                buff.append(line);
                buff.append("\n");
            }

            in.close();
        }
        catch (IOException e) {
            e.printStackTrace();
        }

        return(buff.toString());
    }

    // Given a string, return a version with the given word beeped out
    // note: could use new RE stuff
    private String censor(String text, String word) {
        int i = 0;
        int found;
        while ((found = text.indexOf(word, i)) != -1) {
            text = text.substring(0, found) + "<b>beep</b>" + text.substring(found +
word.length());
            i = found + word.length();
        }
        return(text);
    }
}
```





Cookies

Ways to keep track of state for web app...

1. Hidden fields (hw3)
2. URL re-writing
3. Cookies

Server sends the client a little bit of data (a cookie). The client stores the data for the server

Client sends the data back to the server on each request.

Privacy issues – generally way overblown.

Set-Cookie: Server Response

Cookies are set on the **server response**

The server adds the following to the **HTTP response** header

Set-cookie: id=123xyz

The client notes the cookie, and the server it came from

Get-Cookie: Client Request

When sending a request back to the server, include a Cookie: binding id=123 ; pi=314 ... line on the **HTTP request**

“Here's the cookie binding you sent to me earlier. I'm sending it back to you as a reminder.”

The server can look at the cookie binding in the header if it chooses to.

Cookie features

Binding – just a name=value pair, stick to a-z0-9 chars

Lifetime = session: not written to file system, just works while the browser is running.

Used for session id's. (max-age = -1)

Lifetime = persistent: written to the users prefs, so that later runs of the browser will remember it. (max-age = number of seconds)

Cookie Domain

The browser will only send the cookie back to the domain that it came from..

e.g. .stanford.edu

e.g. .foo.com

This keeps sites from messing up each other's cookies, and makes it difficult for sites to collaborate (privacy issue)

Cookie Applications

Session tracking – use cookies to store the session id.

Works with the back button (at least as well as hidden fields did)

Works, even if they surf through another site.

Works without messing with the HTML

Preferences – use persistent cookies to recognize the user when the first hit the site, and so get some things right for them.

e.g. slashdot remembers your username and reading prefs.

e.g. amazon knows it's you when you go to their site (but they still know to ask for the password if you try to do something)

Cookie Code

```
Cookie cookie = new Cookie(name_str, value_str);  
cookie.setName(name_str);  
cookie.setValue(value_str);  
cookie.setMaxAge(seconds)
```

negative = non-persistent lifetime (i.e. "last for the session" – this is the default)
0 = delete the cookie
1000000000 = long time
response.addCookie(c)

Cookie Before Content

For sending, the cookie must be defined and added to the response before the servlet starts using the PrintWriter to send the HTTP body to the client.

Characters

The cookie name and value should not contain the following chars...

[] () = , " / ? @ : ;

It may be simplest to just stick to a-z0-9

Cookie Reading

cookie.getValue()

cookie.getName()

request.getCookies() => Cookie[] array of all the cookies

Cookie Tricky Timing

Set on response

Read on later request

Timing: must plan ahead, send the cookie now, so you can get it back later

Cookie vs. Param

Cookies come in on the request

So do form parameters (bindings)

Web app must be able to deal with one or the other or both

I check for parameters before cookies, so the parameters take precedence

Mozilla Cookie Features

The open source "mozilla" & mozilla firefox browsers (mozilla.org) have good cookie features

Turn on the "warn about cookies" option

Use the "manage cookies" option in the security preferences

Application: Session

Servlet sessions are implemented with a session cookie

Application: Remember login

e.g. Amazon

e.g. Slashdot

A persistent user=xxx cookie is stored, so the site knows its you when you hit it.

Amazon – Half Auth

Amazon stores a persistent cookie to remember the user

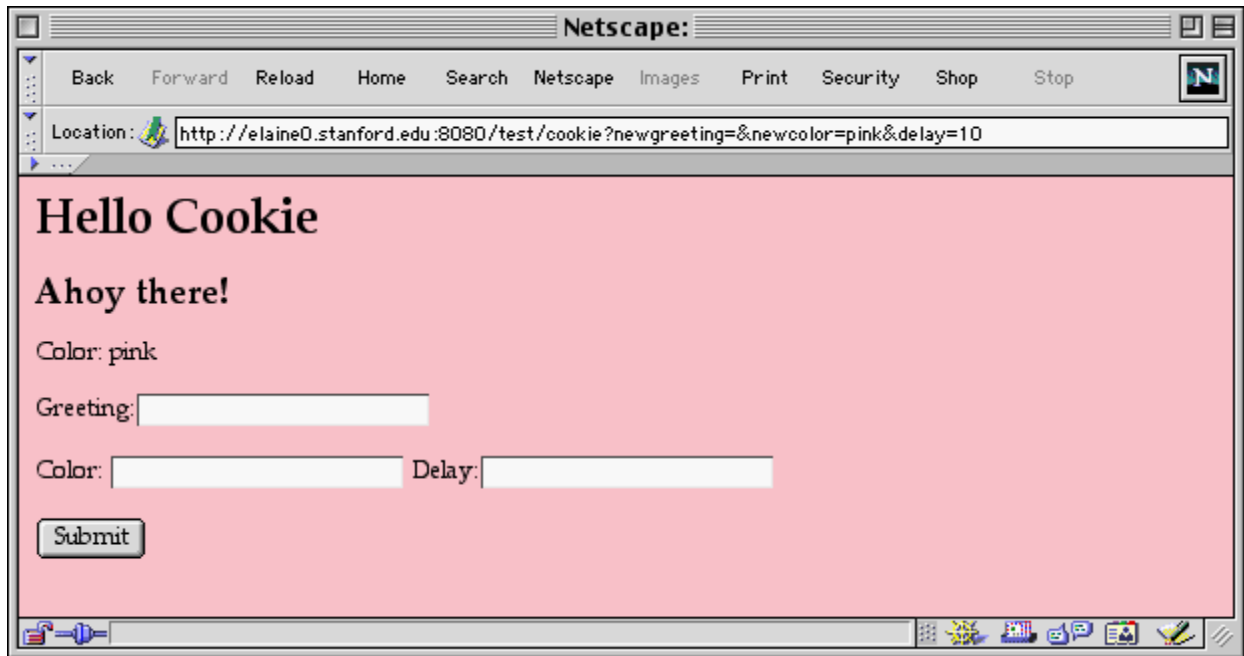
However, amazon does not really know who is on the browser
Therefore, amazon treats it as a "half auth" - the user can see the cart, but not complete an order

To complete an order, the user must give the password (full auth)

Cookie Example

Stores greeting as persistent cookie - 1e9 seconds

Stores color with user defined lifetime



```
// HelloCookie.java
/*
 * A servlet that demonstrates the basic features of cookies.
 */
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloCookie extends HttpServlet
{
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws IOException, ServletException
    {
        String color = "white"; // default color
        String greeting = "Hello"; // default greeting

        String param = null;
        Cookie cookie = null;

        // 1. Deal with greeting
        // Note: check for param before cookie,
        // addCookie() before response is sent
        param = request.getParameter("newgreeting");
        if (param!=null && !param.equals("")) {
            greeting = param;
        }
    }
}
```

```

    // Store a persistent cookie back to the client on the response
    Cookie c = new Cookie("greetingCookie", greeting);
    c.setMaxAge(1000000000); // 1 billion secs = (30 yrs)
    response.addCookie(c);
  }
  else {
    // Check for the greeting cookie
    cookie = findCookie(request, "greetingCookie");
    if (cookie != null) greeting = cookie.getValue();
  }

  // 2. Deal with color preference
  param = request.getParameter("newcolor");
  if (param!=null && !param.equals("")) {
    color = param;

    // Store back a cookie, using the given max-age
    cookie = new Cookie("colorCookie", color);

    String delay = request.getParameter("delay");
    if (delay!=null && !delay.equals("")) {
      cookie.setMaxAge(Integer.parseInt(delay));
    }

    response.addCookie(cookie);
  }
  else {
    cookie = findCookie(request, "colorCookie");
    if (cookie != null) color = cookie.getValue();
  }
  response.setContentType("text/html");
  PrintWriter out = response.getWriter();

  out.println("<html><body>");
  out.println(" bgcolor=\"\" + color + "\"");
  out.println(">");
  out.println("<h1>Hello Cookie</h1>");

  out.println("<h2>" + greeting + "</h2>");
  out.println("<p>Color: " + color);

  out.println("<form><p>Greeting:<input type=text name=newgreeting>");
  out.println("<p>Color: <input type=text name=newcolor>");
  out.println("Delay:<input type=text name=delay><p><input type=submit
value=Submit>");
  out.println("</form>");

  out.println("</body></html>");
}

// Finds the cookie with the given name, or returns null.
// (The cookies!=null test should not be necessary, but I
// I saw one case where it was needed because of a bug in the
// servlet container)
public Cookie findCookie(HttpServletRequest request, String name) {
  Cookie[] cookies = request.getCookies();

  if (cookies!=null) {
    for (int i=0; i<cookies.length; i++) {
      if (cookies[i].getName().equals(name)) return(cookies[i]);
    }
  }

  return(null);
}

```

```
}  
}
```