

Servlets

HTTP Stateless

Appearance of continuity between many pages
From server's point of view, the HTTP requests all come in independently

Solution 1 -- Hidden field (Breadcrumbs)

(e.g. HW3)
Note the information in a hidden field
Stored on the client side
Sent back to the server on a later request
This works ok

Solution 2 -- Session

Store information about the series of pages -- the session -- on the server side.
i.e. a hash table on the server side
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

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
Pass the id to the client (e.g. hidden field id=154xyz, 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

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

HTTP is susceptible to eavesdropping -- could use secure HTTP to avoid that problem

Auth notes

Do not store auth=true on client side

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, although you don't really need to know that

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

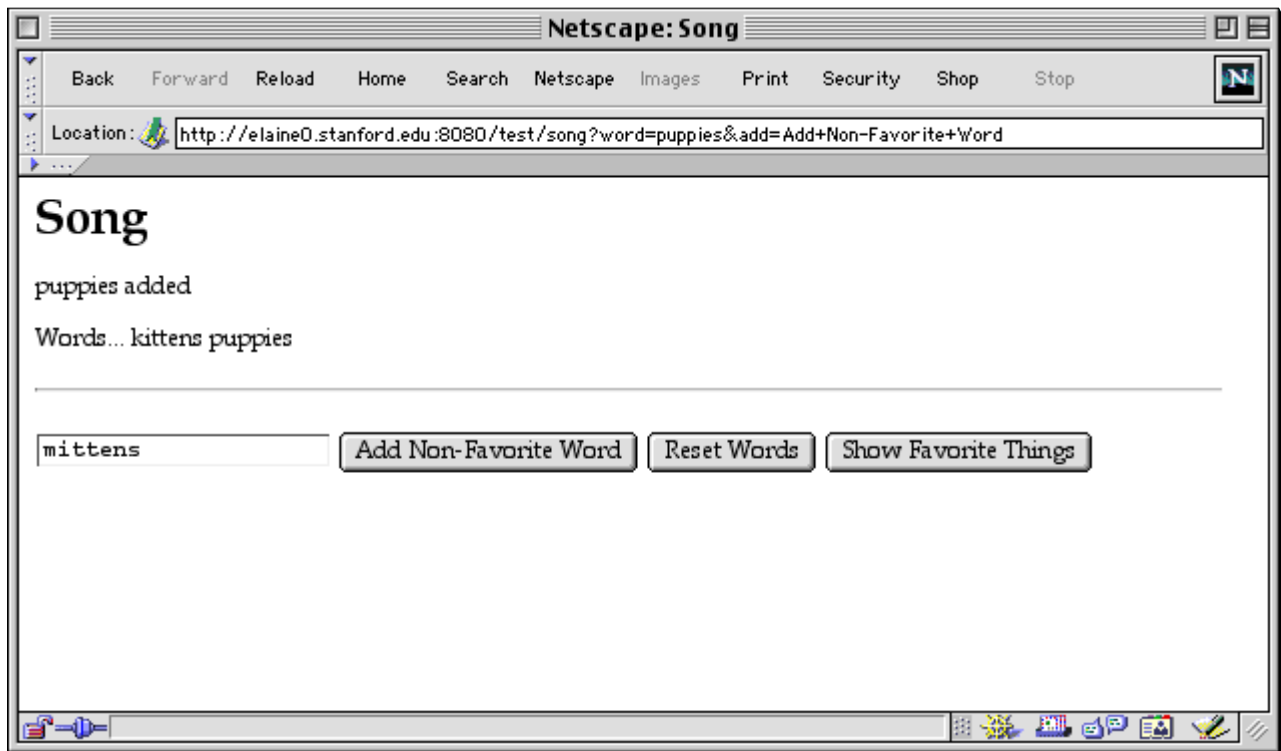
session.setAttribute(key, value);

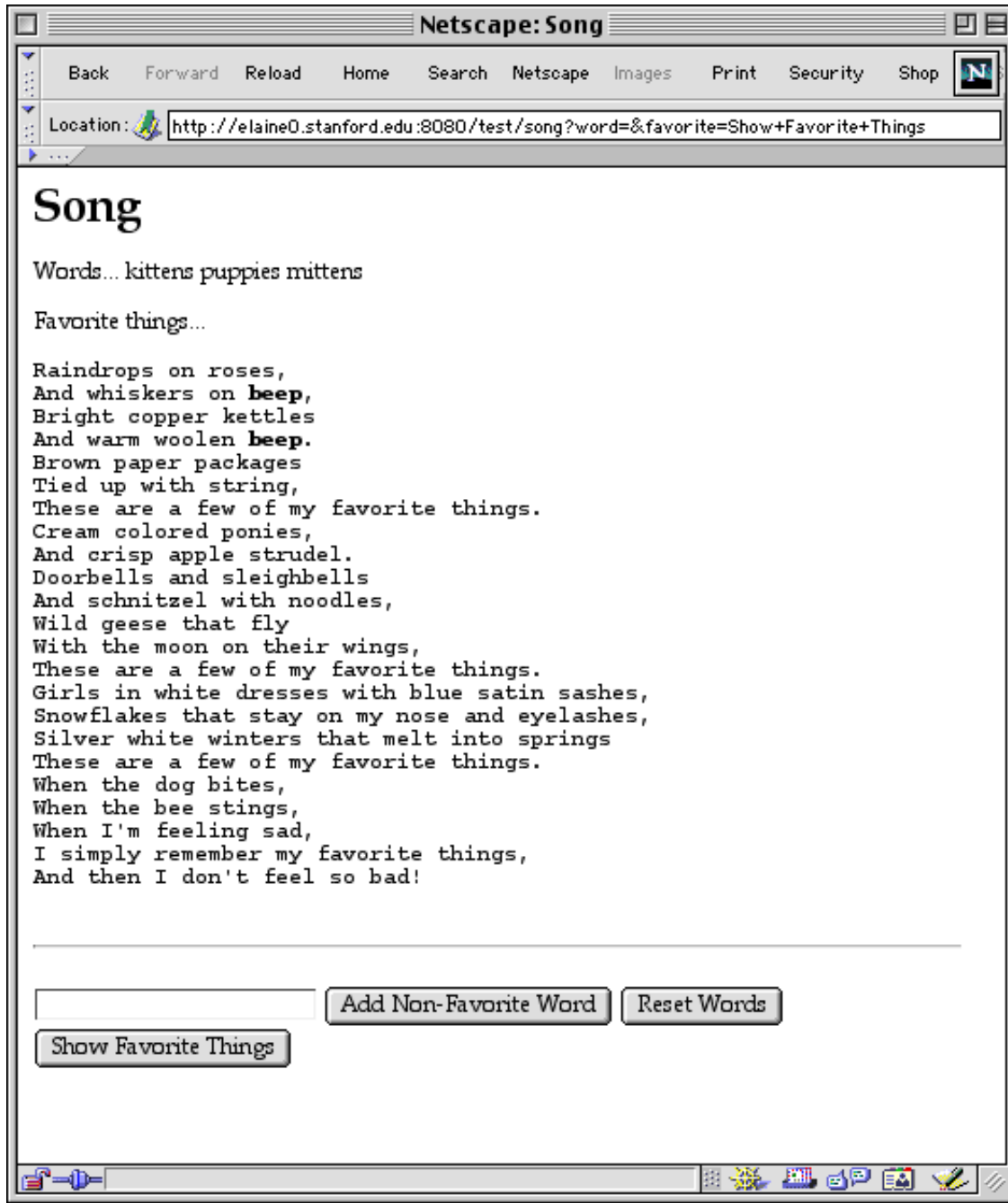
Install a key/value binding in the session

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

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

/*
 * Use a session to keep track of the users list of non-favorite words.
 * Echo the song with the words beeped out.
 */

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());
        }

        // 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) {
    doFavorite(out, words);
}

// Print the form with the A B C 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>");
}

private void doFavorite(PrintWriter out, ArrayList words) {
    String text = readFile("favorite.txt");
    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 ignored) { }
return(buff.toString());
}

// Given a string, return a version with the given word beeped out
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);
}
}
}

```

Storing State

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

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

Cookies

Better than hidden fields

The client stores a little data for the server

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 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 they 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 (session) lifetime (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

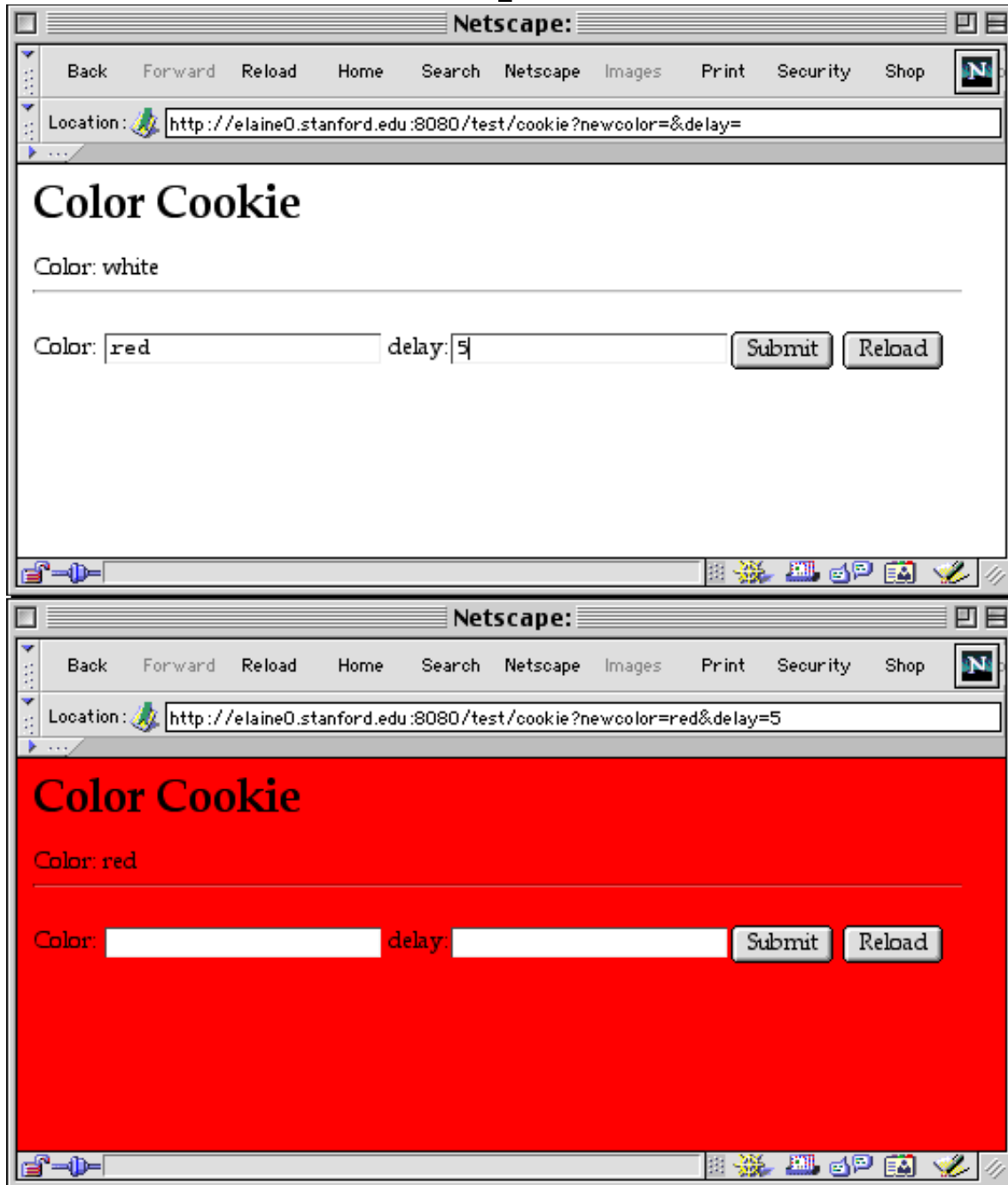
Cookie Reading

`cookie.getValue()`

`cookie.getName()`

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

Color Cookie Example



Cookie Code

```

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 param = null;
        String color = "white";
        Cookie cookie = null;

        // Do cookie operations BEFORE anything else

        // Check for color form param --> make cookie
        param = request.getParameter("newcolor");
        if (param!=null && !param.equals("")) {
            color = param;
            cookie = new Cookie("color", color);

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

            // Store the cookie back to client on the response
            response.addCookie(cookie);
        }
        else { // see if there's a cookie on the client request

            cookie = findCookie(request, "color");
            if (cookie != null) color = cookie.getValue();
        }

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        //System.out.println("Hello world is leaving the building...");

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

        if (color!=null) out.println("Color: " + color);

        out.println("<hr><form>Color: <input type=text name=newcolor>");
        out.println("delay:<input type=text name=delay><input type=submit
value=Submit>");
        out.println("<input type=submit value=Reload></form>");
    }
}

```

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

    // Finds the cookie with the given name, or returns null.
    public Cookie findCookie(HttpServletRequest request, String name) {
        Cookie[] cookies = request.getCookies();

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

        return(null);
    }
}
```