

CGI 5

Back Behavior

Mostly, when the back button is clicked, the browser shows the previous state,
but does not make a request
Will tend to show "stale" state, but that's not too bad for users

Reload / Submit

User can use the "reload" button to re-do the request that made that page
User can press a button on a (stale) form page to re-do its request (GET method
deso the request, POST method may ask the user with a dialog if they really
want to re-submit the request)

Back Example

Survey app -- consider this sequence

page1 shows some state

(clear)

page2 -- cleared

(vote row 0)

page 3 -- shows vote

Second client will see the new state when they next load a page -- ok

What happens if client goes back to page2 and does a "reload" to see what other
votes people have put in?

Problem -- Double Order

Suppose the user is at the "Submit order" page of a web store

They hit the button -- submits the order

Before getting the response, they hit the button again

Or they use the back button to get back to the page, and then hit the button again

Problem: order submitted twice

Attempted Solution 1

When putting out the "submit order page" -- include a serial number which is
remembered on the server side.

Keep track of which numbers have already been submitted -- notice if this is a
second submit

Alternately: track recent orders, and compare this order to recent ones to see if it
appears to be a duplicate

Back Problem 2 -- Add Record

Suppose the user is at the "add record" page for a database, and they use it to add
a record

Then they hit the back button, fill in different form values, and click add-record again, intending to add a 2nd record. From the user's point of view, this is a pretty reasonable interpretation of the interface.

Fundamental Problem

The server may get two submits, the 2nd of which is not intentional
The server may get two submits, the 2nd is intentional

Middleground Solution

Impossible to truly reconcile the two cases -- decide on a case by case basis

Follow principle of "least damage"

e.g. losing their record data is bad, so take the 2nd submit for the record=add case

e.g. getting a dup order is bad, so screen out that case

In general, it's probably best to take the 2nd submit, and put in special case code to screen out common and/or costly cases, such as the double order submit

Remember -- HTML forms are a bit of a hack -- limits on how high quality an GUI we can build

Maybe users can be "trained" over time not to use the back button with web apps

Servlets

Similar to CGI

Servlet container runs on the server side

More features

Run faster, since avoid "startup" costs per hit

(there are Apache modules that make this same optimization for Perl CGI's)

Java oriented-- portable, robust, etc.

Servlet Structure

Java servlet objects exist on the server side -- managed by a "servlet container" which is integrated with the web server.

The servlet container manages loading/unloading servlets and directing requests to them. (We're not going to worry about how it works too much. We just write the servlet, install it, and it should work.)

Servlet containers are available from multiple vendors, and "tomcat" is an open source version.

One servlet object can handle multiple requests

Once created, it can sit in memory on the server side, waiting for additional requests (contrast to CGI)

Request -> doGet()

When a client request comes in, it is sent to the servlet for handling, the doGet() notification is sent to the servlet. The request is sent on its own thread, other requests can execute at the same time on their own threads.

It's fast -- the servlet object is probably already exists in memory, so there's no setup -- just send the doGet()

Because the servlet object is present in memory the whole time, it can build "session" (details later) state in memory that exists across multiple requests

Threading / ivars

By default, each request comes in on its own thread

There is one servlet object

However, there can be multiple requests running doGet() on that servlet at one time (the threading can be turned off, but it's better to leave it on if performance is important under a heavy load).

For the most part, this is not a problem and gives better performance

However, it means that you should not use instance vars in the servlet object to store things that are part of a particular session -- use the session object itself (below) to store session information

1. Subclass off HttpServlet

2. Override doGet() message

Sent on client hit with the GET method

Also, override doPost(), and have it call doGet()

3. HttpServletRequest

Represents the client request

send getParameter("paramName") to search for form bindings -- returns null if no binding

(later) We'll get cookies the client sends us from the request as well

4. HttpServletResponse

1. Set the content type and other HTTP header attributes (complete this before going to later steps)

2. Get PrintWriter from response object

3 Send println() messages to the printwriter to send text to the client

HTTP header before content

It's important that (1) be completed before doing (2) and (3), since (2) will start the sending of the HTTP header/content system. Anything that's going to be in the HTTP header must be set before servlet starts sending the HTTP header/content bytes.

5. No Instance Variables

This goes against the OOP style, but servlets should not have instance variables -- we'll use the "session" object to store state instead of instance variables.

HelloServlet.java

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

public class HelloServlet extends HttpServlet
{

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws IOException, ServletException
    {
        // Set the content type on the HTTP response
        response.setContentType("text/html");

        // Generate the HTML part of the response
        PrintWriter out = response.getWriter();

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

        String title = "Hello World";

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

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

        out.println("<p>Hello World has left the building");

        // Pull a binding off the request (may be null)
        String param = request.getParameter("param");

        if (param != null) {
            out.println("<p>Thanks for the lovely param='" + param + "' binding.");
        }

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

    // Override doPost() -- just have it call doGet()
    public void doPost(HttpServletRequest request,
                      HttpServletResponse response)
        throws IOException, ServletException
    {
        doGet(request, response);
    }
}
```