

Servlets

Web App—Big Picture

Web App Structure

CGI runs server side

Put out HTML/Form to present data and controls for user to take further actions

Coarse interaction

Plan Ahead

Need a plan: pages, forms – what paths the user can take through the system

Need to plan ahead

- If the user can take some action on Page2

- Need to plan for that ahead of time, when processing the submit of Page1 which results in Page2

Breadcrumbs

- Need to use hidden-fields and other tricks to maintain the illusion of continuity

Security

Do not trust data from the user – `$q->param('var')`

For example, never pass such a string to the Perl `open()` command

Perl has a 'taint' feature that can be used to track data from the user (untrusted) vs. trusted data.

CGI-Testing

Run from the command line – pass bindings using the `=&` syntax in quotes on the command line

```
> hello.pl "a=2"
```

Change your action= form to `dumpform.pl` temporarily to see what it is sending.

Use the 'sboxd' variant url to get debugging output for your cgi...

Normal URL: `http://cgi.stanford.edu/~user/cgi-bin/prog.pl`

Debug URL: `http://cgi.stanford.edu/cgi-bin/sboxd/~user/cgi-bin/prog.pl`

Servlets

2nd generation CGI system

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 CGIs)

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, the servlet can sit in memory on the server side, waiting for additional requests (contrast to CGI startup costs)

Request -> doGet()

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

It's fast – the servlet object probably already exists in memory, so there's no setup – just process the doGet(). The first time a servlet object is loaded, it is a little bit slower.

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

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 variables 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/method

Sent on client hit with the GET method
Can 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 on, 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

```
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 reponse
        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);
    }
}
```