

# JSPs

---

## Cookie Scenarios

Session cookie

Persistent cookie

## Cookies vs. Security

Don't store sensitive data in the cookie -- the cookie is stored on the client

e.g. don't store password on cookie

e.g. don't store credit-card number on cookie

## Cookies vs. Privacy

Session cookies enable a site to track that the HTTP requests over, say, an hour, all go together.

Persistent cookie enable a site to track that several sessions belong to the same person.

## Doubleclick -- Web Bugs

Suppose the foo.com page includes an small image from doubleclick.com.

bar.com also includes the image.

The image response can set a cookie, which is included on all later requests.

Doubleclick can build up a picture of the set of sites that one person visits.

This annoys some people, but in reality it's not done that much and you're not giving up much privacy.

Counterintuitive economics: if a vendor knows more about you, you can actually benefit

## JSP/ASP/PHP Theory

CGI world

CGI -- looks like code, possibly with some HTML thrown in

Servlet -- same thing

Flexible but difficult (expensive)

JSP/ASP/PHP world

Have HTML, with little bits of "markup" thrown in to do a little database lookup or other small computation.

HTML is easier to deal with for most purposes.

No re-compile is required -- edit the HTML and go

Works well if the problem is small -- simpler than writing (and understanding) a CGI

Not appropriate for large computations.

## Separate Computation and Presentation

### 1. Programming -- "business logic"

Code, algorithms, database operations --things that look like code

### 2. HTML -- "presentation"

HTML, images, tables -- complex in its way, but it's not code

### Result

Keep code and presentation separate

Servlets, CGI's, etc. for heavy computation

JSP etc. for presentation -- JSPs etc. refer to servlet/CGI system for complex computations

### Cheapness hypothesis

Perhaps this allows you to have just 2 coders and 10 HTML people -- save money

It's not clear that this is true -- the HTML people are also expensive.

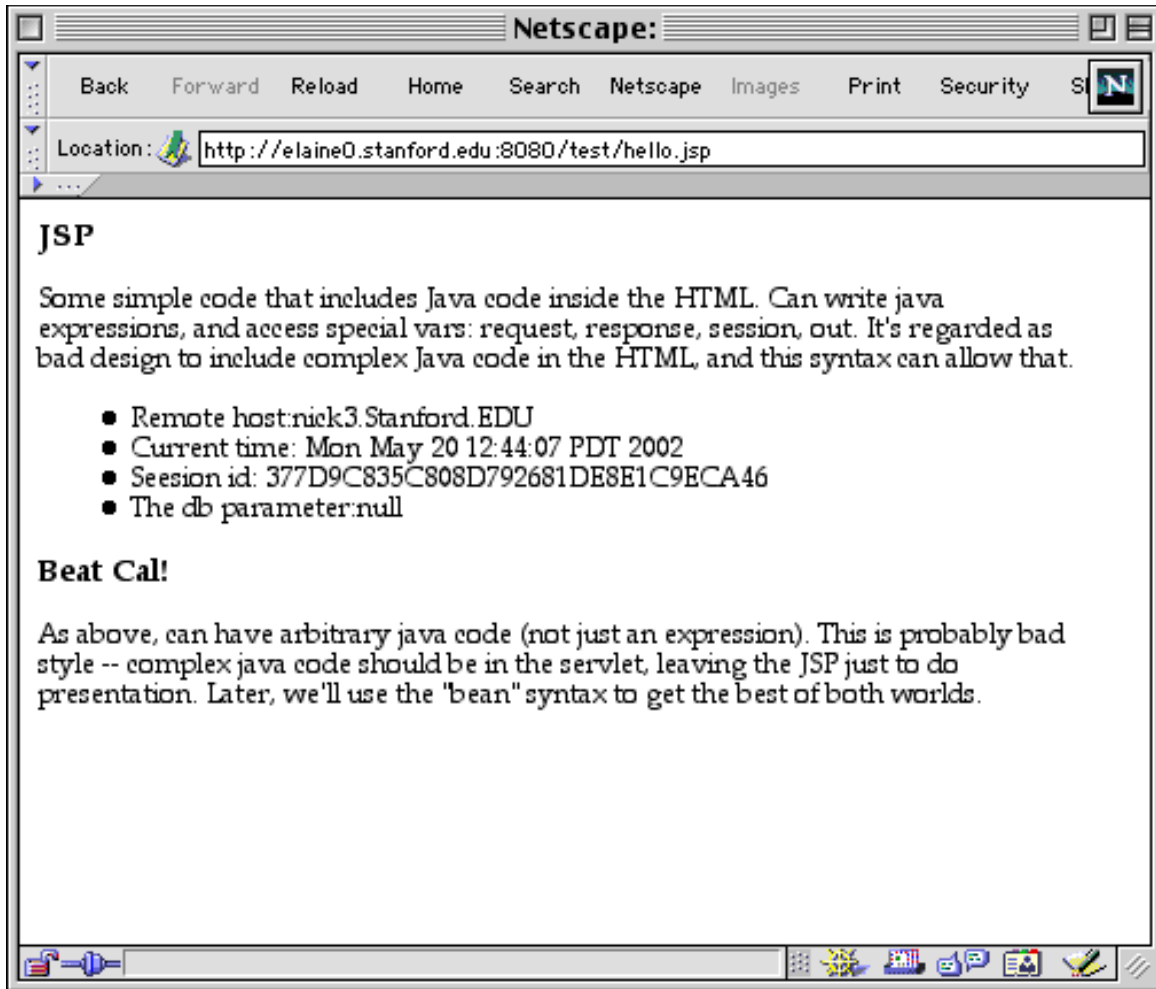
But perhaps it's still better to keep the two activities logically separate.

### HTML markup technologies

ASP -- Microsoft proprietary -- looks like visual basic

PHP -- open source -- works well for connecting a web site to a database such as the open source MySQL

JSP -- java answer to ASP -- uses Java/servlets for the back-end



## JSP `<%= expr %>`

A little Java expression -- output spliced into the HTML  
Special vars: request, response, session

## `request.getParameter("db")`

Pull a binding off the request  
Returns "null" if the binding is not present

## `request.getAttribute("foo")`

Pull an attribute off the request (set earlier with a `setAttribute()` in the servlet)

## JSP `<% code %>`

Can write arbitrary code -- use `out.print()` to output HTML  
Probably bad style -- mixing too much code into the presentation

# hello.jsp

```
<html>
<body bgcolor=white>
<!-- hello.jsp -->
<h3>JSP</h3>
```

<p>Some simple code that includes Java code inside the HTML. Can write java expressions, and access special vars: request, response, session, out. It's regarded as bad design to include complex Java code in the HTML, and this syntax can allow that.

```
<ul>
  <li>Remote host:<%= request.getRemoteHost() %>
  <li>Current time: <%= new java.util.Date() %>
  <li>Seesion id: <%= session.getId() %>
  <li>The db parameter:<%= request.getParameter("db") %>
<!-- this only works if there is a ?db=xxx binding on the request -->
</ul>
```

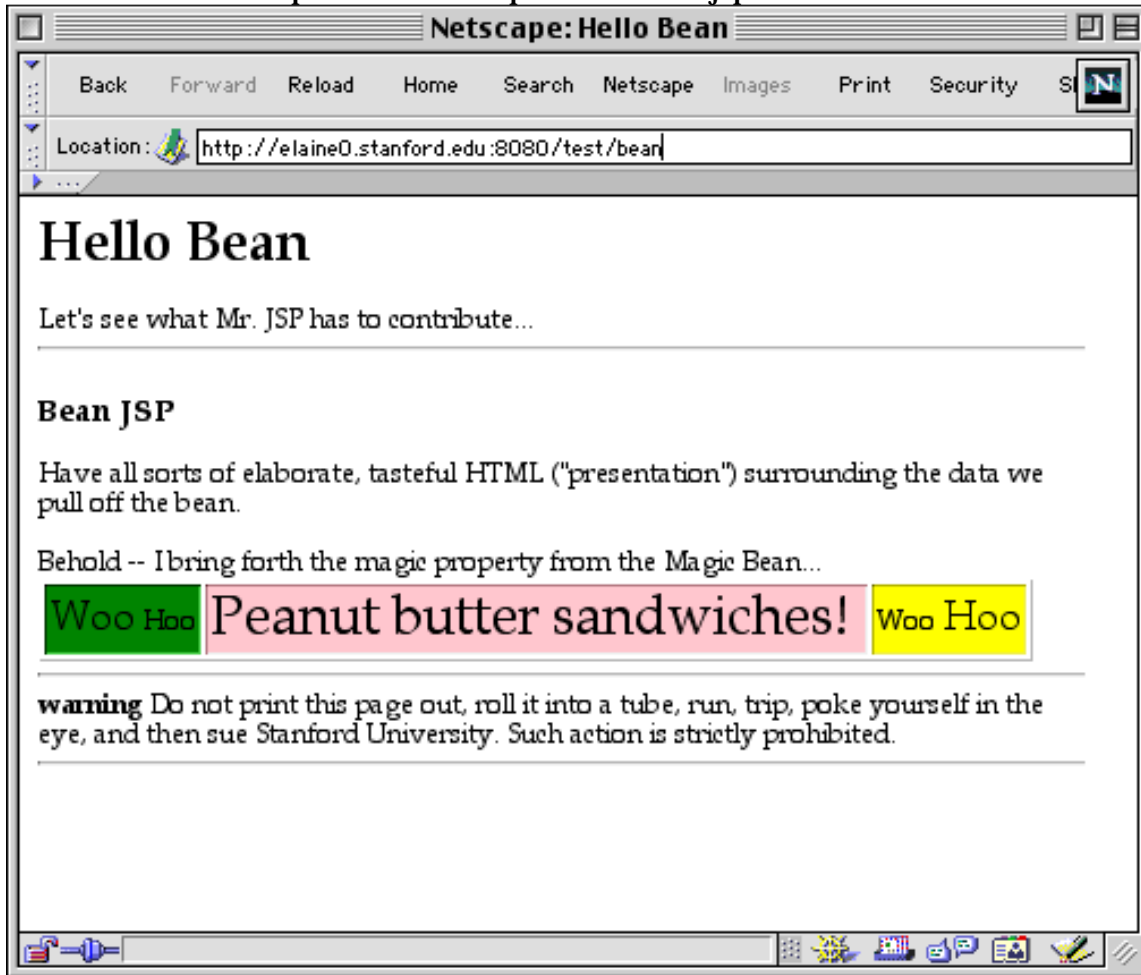
```
<%
    if (request.getRemoteHost().toLowerCase().indexOf("stanford")!=-1)
        out.println("<h3>Beat Cal!</h3>");
%>
```

<p>As above, can have arbitrary java code (not just an expression). This is probably bad style -- complex java code should be in the servlet, leaving the JSP just to do presentation. Later, we'll use the "bean" syntax to get the best of both worlds.

```
</body>
</html>
```

# HelloBean

1. Put an attribute on the request for the JSP to see -- a simple way to communicate to the JSP  
`request.setAttribute("foo", "Binky");`
2. Put a bean on the request. The JSP can access the bean properties with the `useBean` tag below  
`request.setAttribute("bean", bean);`
3. Use `rd.include` to paste in the output of `"/hello.jsp"`



# HelloBean Code

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

public class HelloBean extends HttpServlet
{
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
```

```

throws IOException, ServletException
{
    response.setContentType("text/html");

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

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

    String title = "Hello Bean";

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

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

    out.println("<p>Let's see what Mr. JSP has to contribute...");

    // 0. There may already be form bindings on our request --
    // the JSP can access those with request.getParameter("name")

    // 1. Put an arbitrary binding on the request for the JSP
    // to see -- JSP accesses with getAttribute()
    request.setAttribute("foo", "Binky");

    // 2. Put a bean on the request for the JSP to see
    MagicBean bean = new MagicBean("Peanut butter sandwiches!");
    request.setAttribute("bean", bean);

    // 3. Paste in the output of the JSP
    RequestDispatcher rd = getServletContext().
        getRequestDispatcher("/bean.jsp");
    rd.include(request, response);

    out.println("<hr>");
    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);
}
}

```

## JavaBean

Simple object that collects some named properties

Has a no-argument constructor

The properties are accessible with get/set messages

For a "foo" property, responds to getFoo(), setFoo()

## MagicBean

```
// MagicBean.java
/*
 * A simple bean that contains a single "magic" string.
 */
public class MagicBean {
    private String magic;

    public MagicBean(String string) {
        magic = string;
    }

    public MagicBean() {
        magic = "Woo Hoo";// default magic string
    }

    public String getMagic() {
        return(magic);
    }

    public void setMagic(String magic) {
        this.magic = magic;
    }
}
```

## Improved: useBean

Set up the bean in servlet Java code

There's a fairly simple syntax for the JSP to read values out of the bean

## jsp:usebean

```
<jsp:usebean id="bean" class="MagicBean" />
```

Pull in the bean, prep for getProperty

```
<jsp:getProperty name="bean" property="magic" />
```

Pull the given property out of the given bean

This maps to bean.getMagic()

## jsp:include

Use to splice in content from another page

# bean.jsp

```

<!-- bean.jsp -->
<hr>

<h3>Bean JSP</h3>

<p>Have all sorts of elaborate, tasteful HTML ("presentation")
surrounding the data we pull off the bean.

<p>
Behold -- I bring forth the magic property from the Magic Bean...

<!-- bring in the bean under the name "bean" -->
<jsp:usebean id="bean" class="MagicBean" />

<table border=1>
<tr>
<td bgcolor=green><font size=+2>Woo</font> Hoo</td>

<td bgcolor=pink>
<font size=+3>
  <!-- the following effectively does bean.getMagic() -->
  <jsp:getProperty name="bean" property="magic" />
</font>
</td>

<td bgcolor=yellow>Woo <font size=+2>Hoo</font></td>

</tr>
</table>

<!-- pull in content from another page at request time
with a relative URL ref to another page -->
<jsp:include page="trailer.html" flush="true" />

```