We describe how to implement Java servlets. Based on


1 Dynamic Content

- Servlets are a way of serving web pages that change, that is, dynamic web pages.
- Java servlets are written in Java.
- They were preceded by other technologies.

1.1 Common Gateway Interface

- CGI, which we saw before, runs a program for each request. They use a lot of resources.
- Perl is popular. Adds an interpreter.
- Cannot interact with server, for example, write to log file.

1.2 FastCGI

- A modification that creates only a single persistent process.
- There is one process per program. For concurrent requests it needs one process per request.
- No server interaction.
1.3 **mod_perl**
- mod_perl is an Apache module that embeds a copy of Perl into Apache.
- CGI program can access Apache functionality.
- CGI program is precompiled and executes without forking.
- It only works with Apache. You must use Perl.

1.4 **Server Extensions**
- Some web servers provide APIs for extending it.
- Write C++ code and re-link the server.
- Can bring down whole server or steal information from other parts.

1.5 **Embedding Code in HTML**
- Code can be included in HTML file. This code gets executed by the server and the result placed in its place before the page is sent to browser.
- Microsoft’s Active Server Pages is an example. It requires COM for advanced functionality, so MS Windows. The language looks like:

```vbnet
Sub Page_Load(sender as Object, e as EventArgs)
    If Not Page.IsPostBack Then
        ' Default BG color to white to match the way it
        ' looks when the page first loads.
        back_red.Value = "FF"
        back_green.Value = "FF"
        back_blue.Value = "FF"
    End If
End Sub
```
- JavaServer Pages does the same but uses Java. PHP3 also does the same but uses a C-like language.
- If you want the code to execute at the client then you use JavaScript, which has nothing to do with Java.

1.6 **Servlet Process**
- Run within JVM within web server.
- Each request can be handled by a separate *thread*.
- Can communicate with server.
2 Why Servlets?

- Portable. Servlet code is part of Java. Run on any platform that runs Java.
- Power of Java APIs is available: networking, URL, multithreading, image manipulation, data compression, database connectivity, serialization, RMI, etc.
- Efficient invocation. Once a servlet is loaded it remains in memory as one instance.
- Safety. Type safety of Java language. Exception mechanism of Java.
- Object oriented language with clean Servlet API.
- Tight integration with server.
- Flexible. Servlets can create content by `println`, Java objects, or XML-to-XHTML transformation.

3 Basic HTTP Servlet

- Servlets use the `javax.servlet` and `javax.servlet.http` packages.
- Every servlet must implement the `javax.servlet.Servlet` interface, usually by extending either `javax.servlet.GenericServlet` or `javax.servlet.http.HttpServlet`.
- In a generic you override `service()`.
- In an `HttpServlet` you override `doGet()` and `doPost()`.

3.1 HelloWorld

```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {

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

        out.println("<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/strict.dtd">);
        out.println("<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">");
        out.println("<head><title>Hello World</title></head>";
```

out.println("<body>");
out.println("<h1>Hello World</h1>");
out.println("</body></html>");
}
}

- HttpServletRequest has information about the client, like parameter.
- HttpServletResponse is sent back to the client.

3.2 Hello Forms

- We can handle a form:

<html>
<head>
<title>Introductions</title>
</head>
<body>
<form method="get" action="/servlet/hello">
  If you don’t mind me asking, what is your name?
  <input type="text" name="name"/>
  <p/>
  <input type="submit"/>
</form>
</body>
</html>

- With this code:

```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Hello extends HttpServlet {

  public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {

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

    String name = req.getParameter("name");
    out.println("<html>");
    out.println("<head><title>Hello, " + name + "</title></head>");
    out.println("<body">");
    out.println("Hello, " + name);
    out.println("</body></html>");
  }

  public String getServletInfo() {
    return "A servlet that knows the name of the person to whom it’s saying hello";
  }
}
```
• The code gets the value of the parameter and prints it.

3.3 Handling POST
• Generally, POST and GET should do exactly the same, so all we need is:

```java
public void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
    doGet(req, res);
}
```

3.4 Handling HEAD
• There is no `doHead()`, instead the `service()` method calls `doGet()`
• It then returns only the headers.
• Improve performance with

```java
import java.io.*;
import javax.servlet.);
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
        res.setContentType("text/html");
        if (req.getMethod().equals("HEAD")) return;

        //its really a GET.....
    }
}
```

3.5 Web Applications
• A collection of servlets, JSPs, HTML documents, images, styles sheets, etc. is called a web application.
• To simply deployment, they can be bundled into a web application archive (.war).
• .war is just a .jar, with some extra structure:

  - `index.html`
  - `picture.png`
  - `WEB-INF/web.xml`
  - `WEB-INF/lib/xerces.jar`
  - `WEB-INF/classes/HelloWorld.class`
• The WEB-INF contains configuration information.
• WEB-INF/classes/ contains the servlets.
• WEB-INF/lib/ contains any libraries (.jar) used.
• WEB-INF/web.xml is the deployment descriptor.

3.6 Deployment Descriptor

```xml
<web-app>
  <servlet>
    <servlet-name>hi</servlet-name>
    <servlet-class>HelloWorld</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>hi</servlet-name>
    <url-pattern>/hello.html</url-pattern>
  </servlet-mapping>
</web-app>
```

• Order matters
• http://server:8080/servlet/hi will now access HelloWorld servlet.
• Also, hi is also set to handle URLs that match /hello.html.
• Can use * in pattern to match anything, for example
  - /item/*
  - *.jsp
  - / matches everything.

4 Threads in the Server

• The life-cycle of a servlet is:
  1. Create and initialize servlet.
  2. Handle zero or more calls from clients.
  3. Destroy servlet and garbage collect it.
• Only one instance is kept in memory. No need to keep creating instances.
• Helps the program maintain state (local variables, connections, etc.).
4.1 Simple Counter

```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SimpleCounter extends HttpServlet {
    int count = 0;

    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        count++;
        out.println("Since loading, this servlet has been accessed " +
                        count + " times.");
    }
}
```

- Each request is handled by a different thread.

**Note:**
If you have not used Java threads recently I suggest you review them. Remember that all threads can have access to the same static variables and even the same objects if the threads can communicate with each other (for example, via a socket or a file). Each thread has a start() method which gets called when it is created and a run() method which does the actual work of the thread.

4.1.1 Synchronicity Problems

- Two users could get the same number.

```java
count++;  //Thread 1
count++;  //Thread 2
out.println;  //Thread 1
out.println;  //Thread 2
```

- This can be fixed four ways:

1. Synchronize method:

   ```java
   public synchronized void doGet(HttpServletRequest req, HttpServletResponse res) {
   }
   ```

2. Synchronize code:

   ```java
   PrintWriter out = res.getWriter();
   synchronized(this) {
       count++;
       out.println(...);
   }
   ```
3. Synchronize only needed functionality:

```java
PrintWriter out = res.getWriter();
int local_count;
synchronized(this) {
    local_count = ++ count;
}
out.println(..);
```

4. Live with it.

### 4.2 Counting with Globals

- Actually, each registered name of a servlet is associated with one instance of the servlet.
- All instances have, of course, access to the same static variables (right?).
- So we could count all the instances with:

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

public class HolisticCounter extends HttpServlet {

    static int classCount = 0; // shared by all instances
    int count = 0; // separate for each servlet
    static Hashtable instances = new Hashtable(); // also shared

    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        count++;
        out.println("Since loading, this servlet instance has been accessed " +
                    count + " times.");

        // Keep track of the instance count by putting a reference to this
        // instance in a Hashtable. Duplicate entries are ignored.
        // The size() method returns the number of unique instances stored.
        instances.put(this, this);
        out.println("There are currently " +
                    instances.size() + " instances.");

        classCount++;
        out.println("Across all instances, this servlet class has been " +
                    "accessed " + classCount + " times.");
    }
}
```
4.3 Init and Destroy

- **init()** is called when the server starts, or is first requests, or at the request of the server administrator.
- You can set init parameters for it in the web.xml, then use `getInitParameter("paramname");`
- **destroy()** is called prior to destruction.
- These can be used together to same state across web server activations (crashes):

```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class InitDestroyCounter extends HttpServlet {

    int count;

    public void init() throws ServletException {
        // Try to load the initial count from our saved persistent state
        FileReader fileReader = null;
        BufferedReader bufferedReader = null;
        try {
            fileReader = new FileReader("InitDestroyCounter.initial");
            bufferedReader = new BufferedReader(fileReader);
            String initial = bufferedReader.readLine();
            count = Integer.parseInt(initial);
            return;
        }
        catch (FileNotFoundException ignored) { } // no saved state
        catch (IOException ignored) { } // problem during read
        catch (NumberFormatException ignored) { } // corrupt saved state
        finally {
            // Make sure to close the file
            try {
                if (bufferedReader != null) {
                    bufferedReader.close();
                }
            }
            catch (IOException ignored) { }
        }

        // Default to an initial count of "0"
        count = 0;
    }

    public void doGet(HttpServletRequest req, HttpServletResponse res)
                throws ServletException, IOException {
            res.setContentType("text/plain");
            PrintWriter out = res.getWriter();
            count++;
            out.println("Since the beginning, this servlet has been accessed " +
                        count + " times.");
        }
```
public void destroy() {
    super.destroy(); // entirely optional
    FileWriter fileWriter = null;
    PrintWriter printWriter = null;
    try {
        fileWriter = new FileWriter("InitDestroyCounter.initial");
        printWriter = new PrintWriter(fileWriter);
        printWriter.println(count);
        return;
    }
    catch (IOException e) { // problem during write
        // Log the exception. See Chapter 5.
    }
    finally {
        // Make sure to close the file
        if (printWriter != null) {
            printWriter.close();
        }
    }
}

4.4 In the Background
• A servlet can use the init() to launch a thread that will run continuously.

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class PrimeSearcher extends HttpServlet implements Runnable {

    long lastprime = 0; // last prime found
    Date lastprimeModified = new Date(); // when it was found
    Thread searcher; // background search thread

    public void init() throws ServletException {
        searcher = new Thread(this);
        searcher.setPriority(Thread.MIN_PRIORITY); // be a good citizen
        searcher.start();
    }

    public void run() {
        //do work, say, search for primes.
    }

    public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        out.println("The last prime discovered was " + lastprime);
        out.println("at " + lastprimeModified);
    }
}
public void destroy() {
    searcher.stop();
}

4.5 Avoid Sending Unchanged Data

- Clients send If-Modified-Since header, so, what is the Last-Modified of a document sent by a servlet?
- You can set this with:

```java
public long getLastModified(HttpServletRequest req) {
    return lastPrimeModified.getTime() / 1000 * 1000;
}
```
- Return a time in milliseconds since epoch, but round to nearest second to avoid problems in comparison.
- You can expand on this idea to do server-side caching of your servlet responses. (how?)

5 Getting Information

- Servlet can access information about the server and client.
- For example, in the web.xml you can specify init parameter values, as in

```xml
<web-app>
    <servlet>
        <servlet-name>hi</servlet-name>
        <servlet-class>HelloWorld</servlet-class>
        <init-param>
            <param-name>msg</param-name>
            <param-value>
                A can of ASPARAGUS, 73 pigeons, some LIVE ammo, and a FROZEN DAQUIRI!!
            </param-value>
        </init-param>
    </servlet>
</web-app>
```
![](image.png)

these can be retrieved with code like

```java
public void init() throws ServletException {
    //get msg string
    String msg = getInitParameter("msg");
```
Examine all init parameters

```java
Enumeration enum = getInitParameterNames();
while (enum.hasMoreElements()){
  String name = (String) enum.nextElement();
}
```

- Similarly, a servlet can fetch its own name with

```java
public String ServletConfig.getServletName();
```

### 5.1 Information About the Server

- There are five function to get info on the server

```java
//Return the name of the server
public String ServletRequest.getServerName();

//Return the port number for this request
public int ServletRequest.getServerPort();

//Return name and version of server software
public String ServletContext.getServerInfo();

//Return the value of name server attribute
public Object ServletContext.getAttribute(String name);
```

- The host and port can change for different requests.
- The only mandatory attribute is `javax.servlet.context.tempdir` which is a `java.io.File` reference to a temp directory.

### 5.2 Context Init Parameters

- The inits we saw are per servlet.
- Context init parameters operate per web application. They are defined like:

```xml
<?(xml version="1.0" encoding="ISO-8859-1"?)>
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
  <context-param>
    <param-name>rmiregistry</param-name>
    <param-value>myrmi.jmvidal.cse.sc.edu</param-value>
  </context-param>
</web-app>
```

and accessed using
5.3 What Machine?

- You can get info about the client machine with

```java
// get the client’s IP number
public String ServletRequest.getRemoteAddr();
```

```java
// get the client’s host name
public String ServletRequest.getRemoteHost();
```

- Information comes from the socket connection, so it might be a proxy.

5.4 Who Are You?

- User authentication is handled by the server (more later).
- Once a user has logged in with user/password you can get his name and authentication type:

```java
// get the user’s name
public String HttpServletRequest.getRemoteUser();
```

```java
// get the auth type
public String HttpServletRequest.getAuthType();
// it is one of BASIC, DIGEST, FORM, CLIENT-CERT.
```

- These work only after user has been authorized.

5.5 Getting at the Parameters

- If you have an xhtml file like

```html
<form method="get" action="/servlet/search">
    Question: <input type="text" name="query"><p>
    <input type="submit"></p>
</form>
```

You can get the value of the `query` parameter with

```java
String query = req.getParameter("query");
```

- If the form has multiple values as when using `select`, then you can get all the values with

```java
String[][] queries = req.getParameterValues("query");
```
• Finally, you can get the names of the parameter with

    public Enumeration ServletRequest.getParameterNames();

5.6 Getting the Path
• Once can call a servlet with an extra path:

    http://server:port/servlet/HelloWorld/someextra/path.html

• Get this path with

    public String HttpServletRequest.getPathInfo();

• Useful for pretending you are using static content, backward compat., pretty URLs.
• To get the full (physical) path of the file use

    public String HttpServletRequest.getPathTranslated();

• It could be that the path points to a real file, but in a WAR so you can’t read it.
• You can still pretend it is a regular file by opening it as a URL with

    public URL ServletContext.getResource(String uripath);

    which you use like

    URL url = getServletContext().getResource("/path/to/file.html");

5.7 What Did You Want?
• You can construct a URI that contains the whole path the user requested with:

    public String HttpServletRequest.getRequestURI();

• The scheme part (http, https, ftp) the user used is given by

    public String ServletRequest.getScheme();

• The protocol (with version number) being used is given by

    public String ServletRequest.getProtocol();
5.8 Request Headers

- Remember request headers?
- You can get them with

```java
//returns the value if it is a string, or null if none
public String HttpServletRequest.getHeader(String name);
```

```java
//returns the value as long, -1 if not present,
//or throws IllegalArgumentException
public long HttpServletRequest.getDateHeader(String name);
```

```java
//returns it as int, -1 if not present,
//or throws NumberFormatException
public int HttpServletRequest.getIntHeader(String name);
```

5.9 Reading the Input Stream

- To upload files you use POST:

```html
<form action="/servlet/upload" enctype="multipart/form-data" method="post">
    Filename: <input type="file" name="file="/><br/>
    <input type="submit"/>
</form>
```

- You can then read the raw data by getting a reader from

```java
public BufferedReader ServletRequest.getReader();
```

- To read binary data you use an input stream

```java
public ServletInputStream ServletRequest.getInputStream();
```

- The raw data from the POST above is encoded as in RFC 1867, so it needs further parsing (code in book).

6 Sending A Response

```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/html");
    }
}
```
PrintWriter out = res.getWriter();

out.println("<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Strict//EN\" \"http://www.w3.org/TR/xhtml1/strict.dtd\">");
out.println("<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">");
out.println("<head><title>Hello World</title></head>");
out.println("<body>");
out.println("<h1>Hello World</h1>");
out.println("</body></html>");
}

• We set content type and get a writer. This works for character data.

• For binary data you use

   public ServletOutputStream ServletResponse.getOutputStream() throws IOException;

6.1 Persistent Connections

• HTTP 1.1 allows for persistent connection: don’t close socket.

• Receiver needs to know when one replay ends and another one starts.

• Use Content-Length header. It is set with

   public void ServletResponse.setContentLength(int len);

• Must call it before sending response body. Must be correct.

6.2 Buffering

• Guessing the right length is hard. Instead, write to a buffer. Server sets the Content-Length automatically. Set by:

   public void ServletResponse.setBufferSize(int size);

• You can get the current buffer size with

   public int ServletResponse.getBufferSize();

• You can find out it has already been committed with

   public boolean ServletResponse.isCommitted();

• And call reset() to clear it. Here is an example:

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class Buffering extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException {
        res.setBufferSize(8 * 1024); // 8K buffer
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();

        int size = res.getBufferSize(); // returns 8096 or greater
        out.println("The client won’t see this");
        res.reset();
        out.println("Nor will the client see this!");
        res.reset();
        out.println("And this won’t be seen if sendError() is called");
        if (req.getParameter("important parameter") == null) {
            res.sendError(res.SC_BAD_REQUEST, "important parameter needed");
        }
    }
}

6.3 Response Code

- You can set the status code with:

    public void HttpServletResponse.setStatus(int sc);

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC_OK</td>
<td>200</td>
<td>Everything’s OK.</td>
</tr>
<tr>
<td>SC_NO_CONTENT</td>
<td>204</td>
<td>There is nothing to return.</td>
</tr>
<tr>
<td>SC_MOVED_PERMANENTLY</td>
<td>301</td>
<td>The requested resource has moved. Put new URL in Location header.</td>
</tr>
<tr>
<td>SC_MOVED_TEMPORARILY</td>
<td>302</td>
<td>Temporary move. Put new URL in Location header.</td>
</tr>
<tr>
<td>SC_UNAUTHORIZED</td>
<td>401</td>
<td>Not authorized.</td>
</tr>
<tr>
<td>SC_NOT_FOUND</td>
<td>404</td>
<td>Document was not found.</td>
</tr>
<tr>
<td>SC_INTERNAL_SERVER_ERROR</td>
<td>500</td>
<td>Server is broken.</td>
</tr>
</tbody>
</table>

6.4 Setting Headers

- The most common headers you want to set are
<table>
<thead>
<tr>
<th>Header</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache-Control</td>
<td>Either no-cache or no-store (no proxy store) and max-age=fresh for these many seconds</td>
</tr>
<tr>
<td>Connection</td>
<td>keep-alive to keep socket open or close. Servers handle this automagically.</td>
</tr>
<tr>
<td>Retry-After</td>
<td>Date or seconds to wait for server to start working.</td>
</tr>
<tr>
<td>Expires</td>
<td>Date when the documents will(may) change.</td>
</tr>
<tr>
<td>Location</td>
<td>The URL to which the document moved to.</td>
</tr>
<tr>
<td>WWW-Authenticate</td>
<td>Authorization scheme and realm.</td>
</tr>
<tr>
<td>Content-Encoding</td>
<td>Can be gzip or compress.</td>
</tr>
</tbody>
</table>

- They can all be set with

```java
public void HttpServletResponse.setHeader(String name, String value);
```

- To avoid unnecessary conversions to string you can also use

```java
public void HttpServletResponse.setDateHeader(String name, long date);
public void HttpServletResponse.setIntHeader(String name, int value);
```

6.5 Client Refresh

- The client can be instructed to wait and then reload something else using the **Refresh** header:

```java
setHeader("Refresh", "3; URL=http://slashdot.org");
```

- An easy way to do slide show animation.

6.6 Handling Errors

- When something goes wrong, you must consider
  - How much to tell the client.
  - How to record the problem.
  - How to recover from the problem.

- You can change the standard error page in **web.xml**

```xml
<error-page>
  <error-code>400</error-code>
  <location>/400.html</location>
</error-page>
```
6.7 Log Files

- An indispensable debugging tool.
- You can write a log entry with

  ```java
  public void GenericServlet.log(String msg);
  public void GenericServlet.log(String msg, Throwable t);
  ```
- Timestamp automatically added.

6.8 Exceptions

- A servlet can propagate only exceptions that subclass IOException, ServletException or RuntimeException (because of service()’s signature).
- The ServletException constructor takes a root cause as argument:

  ```java
  javax.servlet.ServletException(Throwable rootCause);
  javax.servlet.ServletException(String msg, Throwable rootCause);
  ```
- So, you can pass it on up, then fetch it with:

  ```java
  public Throwable ServletException.getRootCause();
  ```
- The UnavailableException is the only one that subclasses ServletException. Used to indicate servlet is unavailable:

  ```java
  javax.servlet.UnavailableException(String msg); //permanently unavailable
  javax.servlet.UnavailableException(String msg, int seconds); //be back in seconds
  ```

6.9 The Stop Button

- Beckons the user to press it.
- We find out via IOException when writing.
- As such, finally gets called often. Make sure to close all other files you might be using.
- Remember that finally gets executed at the end of try/catch/finally block, always.
7  WAP

<table>
<thead>
<tr>
<th>Web</th>
<th>WAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML, JavaScript</td>
<td>WML, WMLScript</td>
</tr>
<tr>
<td>HTTP</td>
<td>WSP: Session, WTP: Transaction</td>
</tr>
<tr>
<td>TLS/SSL</td>
<td>WTLS</td>
</tr>
<tr>
<td>TCP</td>
<td>WDP</td>
</tr>
<tr>
<td>IP</td>
<td>Bearer</td>
</tr>
</tbody>
</table>

- **Wireless Application Protocol** is used to talk to cell phones.
- Optimized for low bandwidth and small screens with slow CPUs.
- Layers mirror Web Stack.

### 7.1 WAP Gateway

- WAP gateways translate from WSP/WTP to HTTP.
- As a web server designer, you only have to worry about serving pages in WML.

### 7.2 WML

- **Wireless Markup Language** is the markup language.
- **Wireless Bitmap** encodes monochromatic images.
- **WMLScript** is the scripting language.
- WML is mostly a subset of HTML that instead defines multiple cards (web pages). Looks like:
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN" "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card id="Developer" title="Developer Home">
<p>
<b>Developer Home</b>
<anchor>
Web Sites <go href="#Web" />
</anchor><br/>
<anchor>
WML Examples <go href="#WML" />
</anchor><br/>
<anchor>
HDML Examples <go href="#HDML" />
</anchor><br/>
<anchor>
Developer Info <go href="#info" />
</anchor><br/>
</p>
</card>

<card id="Web" title="Web Sites">
<p>
</p>
21
Here they are:

```html
<a href="http://test.com">test</a><br/>
<a href="http://sample.com">sample</a><br/>
</p>
</card>

<card id="WML" title="WML Examples">
  <p>
    If you want to learn more<br/>
    read a book<br/>
  </p>
</card>
</wml>

7.3 Serving WML

- You can match the proper mime types to extensions (for static files) in your `web.xml`:

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app
  PUBLIC "/-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2.2.dtd">
  <mime-mapping>
    <extension>
      wml
    </extension>
    <mime-type>
      text/vnd.wap.wml
    </mime-type>
  </mime-mapping>
  <mime-mapping>
    <extension>
      wmls
    </extension>
    <mime-type>
      text/vnd.wap.wmlscript
    </mime-type>
  </mime-mapping>
  <mime-mapping>
    <extension>
      wbmp
    </extension>
    <mime-type>
      image/vnd.wap.wbmp
    </mime-type>
  </mime-mapping>
</web-app>
```

- For dynamic content just generate proper WML and

```java
res.setContentType("text/vnd.wap.wml");
```
• Remember to generate proper set of cards (low bandwidth).

8 Serving Dynamic Images
• Generating images is made easy by the use of Java libraries.

```java
import java.io.*;
import java.awt.*;
import javax.servlet.*;
import javax.servlet.http.*;
import Acme.JPM.Encoders.GifEncoder; //3rd party

public class HelloWorldGraphics extends HttpServlet {

public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
    ServletOutputStream out = res.getOutputStream(); // binary output!
    Graphics g = null;

    try {
        Image image = new BufferedImage(400, 60, TYPE_INT_BGR);
        g = image.getGraphics();

        // Draw "Hello World!" to the off-screen graphics context
        g.setFont(new Font("Serif", Font.ITALIC, 48));
        g.drawString("Hello World!", 10, 50);

        // Encode the off-screen image into a GIF and send it to the client
        res.setContentType("image/gif");
        GifEncoder encoder = new GifEncoder(image, out);
        encoder.encode();
    }
    finally {
        // Clean up resources
        if (g != null) g.dispose();
    }
}
```

8.1 About Images
• GIF is lossless, max 256. Encumbered by patents.
• JPEG is lossy, high photo compression.
• PNG is the new GIF. Its smaller, millions of colors, lossless, alpha channel, no patents.
• The jdk has a jpeg encoder in com.sun.image.codec.jpeg.
• Sun’s [Java Advanced Imaging API](https://java.sun.com/javee) has a PNG encoder.
• You can also combine images and modify existing images dynamically.
9 Serving Compressed Content

- This is transparent to the user and saves bandwidth.
- All you have to do is
  1. Set the `Content-Encoding` header to the appropriate of `gzip` (best), `compress`, or `deflate`.
  2. Wrap the output in a `GZIPOutputStream` or `ZipOutputStream`, make sure to `close()`.

- Client sends an `Accept-Encoding` header that specifies the acceptable encodings.

10 Session Tracking

Dory suffers from short-term memory loss.

- HTTP is `stateless`.
- Every statement is completely new to the server.
- Hard to implement shopping cart, customized homepages, etc.
- We need a way to know who we are talking to!

10.1 User Authentication

- A simple way is to use server security (later) and force the user to log in.
- You can then do a

  ```java
  String name = req.getRemoteUser();
  ```

to get the name of the user.

- It is easy to implement but has some problems:
  1. It requires everyone to create account and log in.
  2. There is no logout mechanism, just kill browser.
  3. Cannot have more than one session at a time.
10.2 Hidden Form Fields

- You can use hidden form fields like

```html
<body>
<form method="get" action="/servlet/hello">
  If you don’t mind me asking, what is your name?
  <input type="text" name="name"/>
  <input type="hidden" name="id" value="1234"/>
  <input type="hidden" name="degree" value="bs"/>
  <input type="submit"/>
</form>
<p/>
</body>
```

- Every html page you generate must be a form and include the hidden fields.

```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ShoppingCartViewerHidden extends HttpServlet {

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

        // Cart items are passed in as the item parameter.
        String[] items = req.getParameterValues("item");
        // Ask if the user wants to add more items or check out.
        // Include the current items as hidden fields so they’ll be passed on.
        out.println("<form action="/servlet/ShoppingCart" method="POST">
            <input type="hidden" name="item" value="" + items[i] + ">
            <input type="hidden" name="item\" value="" + 
            <input type="submit"/>
        </form>

        // Cart items are passed in as the item parameter.
        String[] items = req.getParameterValues("item");
        // Ask if the user wants to add more items or check out.
        // Include the current items as hidden fields so they’ll be passed on.
        out.println("<form action="/servlet/ShoppingCart" method="POST">
            <input type="hidden" name="item" value="" + items[i] + ">
            <input type="hidden" name="item\" value="" + 
            <input type="submit"/>
        </form>
    }
}
```

10.3 URL Rewriting

- Another way to achieve the same thing as with URL rewriting.

- This has the advantage that it works for all URLs (GET)

- But, it makes long URLs.

- Both methods require us to dynamically generate all pages in website in order to keep session.
10.4 Cookies

- Netscape introduced cookies, now defined in [RFC 2109].
- You can create one with

  ```java
  public Cookie(String name, String value);
  ```

  then add it to the response with

  ```java
  public void HttpServletResponse.addCookie(Cookie cookie);
  ```

- The client will store 20 cookies per site, each one no more than 4Kb. You can retrieve the cookies with

  ```java
  public Cookie[] HttpServletRequest.getCookies();
  ```

- You restrict the domain to which the client will send the cookie with

  ```java
  public void Cookie.setDomain(String pattern);
  ```

  and the path with

  ```java
  public void Cookie.setPath(String uri);
  ```

- The expiration date is set with

  ```java
  public void Cookie.setMaxAge(int expiry-second);
  ```

- Set the comment associated with it:

  ```java
  public void Cookie.setComment(String comment);
  ```

- Set a new value with

  ```java
  public void Cookie.setValue(String newValue);
  ```

10.4.1 Cookies Example

```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ShoppingCartViewerCookie extends HttpServlet {
  public void doGet(HttpServletRequest req, HttpServletResponse res)
      throws ServletException, IOException {
    ...
  }
}
```
res.setContentType("text/html");
PrintWriter out = res.getWriter();

// Get the current session ID by searching the received cookies.
String sessionid = null;
Cookie[] cookies = req.getCookies();
if (cookies != null) {
    for (int i = 0; i < cookies.length; i++) {
        if (cookies[i].getName().equals("sessionid")) {
            sessionid = cookies[i].getValue();
            break;
        }
    }
}

// If the session ID wasn’t sent, generate one.
// Then be sure to send it to the client with the response.
if (sessionid == null) {
    sessionid = generateSessionId();
    Cookie c = new Cookie("sessionid", sessionid);
    res.addCookie(c);
}

private static String generateSessionId() {
    String uid = new java.rmi.server.UID().toString(); // guaranteed unique
    return java.net.URLEncoder.encode(uid); // encode any special chars
}

• Some consider cookies a privacy threat and turn them off. We can achieve the same effect
  with the previous two methods.

10.5 Session Tracking API

• Every user is associated with a javax.servlet.http.HttpSession object were you can store
  any info about him.

• You get it with

    public HttpSession HttpServletRequest.getSession();

    if now session then one is created.

• You can add data to it with

    public void HttpSession.setAttribute(String name, Object value);

    and get its value with

    public Object HttpSession.getAttribute(String name);
public class SessionTracker extends HttpServlet {

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

        // Get the current session object, create one if necessary
        HttpSession session = req.getSession();

        // Increment the hit count for this page. The value is saved
        // in this client’s session under the name "tracker.count".
        Integer count = (Integer) session.getAttribute("tracker.count");
        if (count == null)
            count = new Integer(1);
        else
            count = new Integer(count.intValue() + 1);
        session.setAttribute("tracker.count", count);

        out.println("<html><head><title>SessionTracker</title></head>");
        out.println("<body><h1>Session Tracking Demo</h1>");

        // Display the hit count for this page
        out.println("You’ve visited this page "+ count +
                "\time.\times.");

        out.println("</p>/") ;

        out.println("<h2>Here is your session data:</h2>");
        Enumeration enum = session.getAttributeNames();
        while (enum.hasMoreElements()) {
            String name = (String) enum.nextElement();
            out.println(name + ": "+ session.getAttribute(name) + "<br/>");
        }
        out.println("</body></html>");
    }
}

10.5.1 Session Tracking Customizations

• The servlet engine automatically chooses the method to use (first cookies, then re-writing).

• Sessions expire if user does not click on anything. Default is 30min. Change it in web.xml or with

public void HttpSession.setMaxInactiveInterval(int secs);

set with care (security vs resource use).
• Other helpful methods are

```java
// Has this session just been created?
public boolean HttpSession.isNew();

// Invalidate session
public void HttpSession.invalidate();

// When was it created?
public long HttpSession.getCreationTime();
```

10.5.2 Session Tracking: Behind the Scenes

• If cookies fail, url rewriting is handled by

```java
// return the new url
public String HttpServletResponse.encodeURL(String url);
```

• If you feel nosy, you can get the actual is string with

```java
public String HttpSession.getId();
```

• You can build an object that gets a callback whenever it is bound or unbound by having it implementing the `javax.servlet.http.HttpSessionBindingListener` interface. The callbacks you need to implement are

```java
public void HttpSessionBindingListener.valueBound(HttpSessionBindingEvent event);
public void HttpSessionBindingListener.valueUnbound(HttpSessionBindingEvent event);
```

11 Security

• Four aspects:
  1. Authentication
  2. Authorization
  3. Confidentiality
  4. Integrity

11.1 HTTP Authentication

• HTTP protocol has **basic authentication** were server maintains database of usernames and passwords.

• Server asks for username/pass when client tries to access secure pages.

• Client sends user/pass encoded using Base64 (basically, plain text).

• In **digest authentication** they are encrypted using MD5 (secure encryption), but server must still maintain list of user/pass (honey-pot).
• You can configure these into servlet server, by giving each user one or more roles in the tomcat-users.xml file:

```
<tomcat-users>
  <user name="Dilbert" password="dnrc" roles="engineer" />
  <user name="Wally" password="iluvalice" roles="engineer,slacker" />
  <user name="MrPointyHair" password="MrPointyHair" roles="manager,slacker" />
</tomcat-users>
```

then modify web.xml to protect the needed directories

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
  PUBLIC "+//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd"
>
<web-app>
  <servlet>
    <servlet-name>secret</servlet-name>
    <servlet-class>SalaryServer</servlet-class>
  </servlet>

  <security-constraint>
    <web-resource-collection>
      <web-resource-name>SecretProtection</web-resource-name>
      <url-pattern>/servlet/SalaryServer</url-pattern>
      <url-pattern>/servlet/secret</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>manager</role-name>
    </auth-constraint>
    <login-config>
      <auth-method>
```
BASIC <!-- BASIC, DIGEST, FORM, CLIENT-CERT -->
</auth-method>
<realm-name>
  Default <!-- optional, only useful for BASIC -->
</realm-name>
</login-config>

<security-role>
  <role-name>
    manager
  </role-name>
</security-role>
</web-app>

- Once the user is logged in you can get his principal–entity being authenticated–with:

  public java.security.Principal HttpServletRequest.getUserPrincipal();
  principal.getName();

  or just check for the correct role with

  public boolean HttpServletRequest.isUserRole(String role);

11.2 Form-Based Authentication

- You can instruct the servlet server to use a form, but still do the same basic authentication.
- Friendly logins. Very popular.
- Change web.xml to

  <?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE web-app
   PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
   "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
  <web-app>
    <!-- ...-->

    <login-config>
      <auth-method>
        FORM <!-- BASIC, DIGEST, FORM, CLIENT-CERT -->
      </auth-method>
      <form-login-config>
        <form-login-page>
          /loginpage.html
        </form-login-page>
        <form-error-page>
          /errorpage.html
        </form-error-page>
      </form-login-config>
    </login-config>
no the server will show `loginpage.html` whenever someone who has not logged in tries to access a secure page. `loginpage.html` should look something like

```html
<!--The j_* names are important-->  
<form method="POST" action="j_security_check">  
Name: <input type="text" name="j_username" value="" size="15">  
Password: <input type="password" name="j_password" value="" size="15">  
<input type="submit" value=" OK ">
</form>
```

- Password is transmitted in plain text.
- Neither form or standard support logout.

11.3 Custom Authentication

- If you want to store usernames in a database. Must write servlet to do authorization.
- The client sets the **Authorization:** header to:

  ```
  Authorization: BASIC base64(username:password)
  ```

```java
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.oreilly.servlet.Base64Decoder;

public class CustomAuth extends HttpServlet {

    Hashtable users = new Hashtable();

    public void init(ServletConfig config) throws ServletException {
        super.init(config);

        // Names and passwords are case sensitive!
        users.put("Wallace:cheese", "allowed");
        users.put("Gromit:sheepnapper", "allowed");
        users.put("Penguin:evil", "allowed");
    }

    public void doGet(HttpServletRequest req, HttpServletResponse res)  
        throws ServletException, IOException {
        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();

        // Get Authorization header
        String auth = req.getHeader("Authorization");

        // Do we allow that user?
```
if (!allowUser(auth)) {
   // Not allowed, so report he's unauthorized
   res.setHeader("WWW-Authenticate", "BASIC realm=" + "users\" ");
   res.sendError(res.SC_UNAUTHORIZED);
   // Could offer to add him to the allowed user list
}
else {
   // Allowed, so show him the secret stuff
   out.println("Top-secret stuff");
}
}

// This method checks the user information sent in the Authorization
// header against the database of users maintained in the users Hashtable.
protected boolean allowUser(String auth) throws IOException {
   if (auth == null) return false; // no auth
   if (!auth.toUpperCase().startsWith("BASIC "))
      return false; // we only do BASIC
   // Get encoded user and password, comes after "BASIC ">
   String userpassEncoded = auth.substring(6);
   // Decode it, using any base 64 decoder (we use com.oreilly.servlet)
   String userpassDecoded = Base64Decoder.decode(userpassEncoded);
   // Check our user list to see if that user and password are "allowed"
   if ("allowed").equals(users.get(userpassDecoded)))
      return true;
   else
      return false;
}

11.4 Custom Form-Based Authorization

- You can use a form to request password and then handle the authorization with your own
  servlet. Now you can request any number of things from user.
- When user asks for secure document redirect to login screen. After login go back to originally
  requested page.
- All protected resources must check to make sure user is logged in.
- The login handler is

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

   public class LoginHandler extends HttpServlet {

      public void doPost(HttpServletRequest req, HttpServletResponse res)
         throws ServletException, IOException {

         // Your code here
      }
   }
res.setContentType("text/html");
PrintWriter out = res.getWriter();

// Get the user’s account number, password, and pin
String account = req.getParameter("account");
String password = req.getParameter("password");
String pin = req.getParameter("pin");

// Check the name and password for validity
if (!allowUser(account, password, pin)) {
    out.println("...Access Denied...");
} else {
    // Valid login. Make a note in the session object.
    HttpSession session = req.getSession();
    session.setAttribute("logon.isDone", account); // just a marker object

    // Try redirecting the client to the page he first tried to access
    try {
        String target = (String) session.getAttribute("login.target");
        if (target != null) {
            res.sendRedirect(target);
            return;
        }
    } catch (Exception ignored) { }

    // Couldn’t redirect to the target. Redirect to the site’s home page.
    res.sendRedirect("/");
}

protected boolean allowUser(String account, String password, String pin) {
    return true; // trust everyone
}

11.5 Digital Certificates

• We use public key encryption.

• Public key encrypts, private key signs.

• Trusted third parties (Verisign) sign our public key.

• The Secure Sockets Layer (SSL) protocol forms the basis of the Transport Layer Security
  (TLS) protocol, as detailed in RFC 2246. Its the standard.

• It works by:
  1. User connects to server using https.
  2. Server signs its own public key with its own private key and send it back to browser.
  3. Browser uses the server’s public key to verify that the same person who signed the key also
     owns it.
4. Browser checks if authority (Verisign) signed the public key (avoid man-in-middle attack). Otherwise, asks user if key can be trusted.

5. Client generates symmetric key for session, encrypts it with server's public key, and sends to server.
   • Similarly (inversely), a server can request an authentication certificate from the client.

11.6 Configuring SSL Security
   • An servlet that requires https can say so in the web.xml file

   ```xml
   <security-constraint>
   <!-- .... -->
   <user-data-constraint>
   <transport-guarantee>
   CONFIDENTIAL
   </transport-guarantee>
   </user-data-constraint>
   </security-constraint>
   ```

   • The servlet can determine if it was called under https with

     ```java
     public boolean ServletRequest.isSecure();
     ```

11.7 Security Summary
   • If data is sensitive, use https.
   • Use your own authentication if you want, but be mindful of how passwords are stored.
   • Never implement your own encryption.

12 Enterprise Servlets
   • Enterprise servlets are designed to support large-scale websites.
     – Reliability.
     – Load balancing.
     – Failover.
     – Integration with other J2EE technologies.

12.1 Load Distribution
   • Getting a faster machine and connection is almost always enough.
   • [Enterprise Java Beans](EJB) are designed to be distributable objects. As such, an EJB must follow certain rules.
   • Servlets have no such rules, but a good approximation is
     1. Instance and static variables should not store state, since different instances of the servlet might exist.
2. `ServletContext` cannot be used to store state, for the same reason.

3. Any object placed in `HttpSession` should implement `java.io.Serializable`.

4. Use the `getServletContext().getResource()` to read files.

- If you follow these rules you can mark servlet as `<distributable/>` in `web.xml`

- There are several distribution (clustering) styles.
  1. No clustering.
  2. Clustering with no session migration and no session failover. Sessions stick to original server. A crash can result in broken session.
  3. Clustering with session migration and no session failover. All session objects must be serializable.
  4. Clustering with session migration and failover. Crash does not invalidate session.

- These are implemented by servlet server. You must simply follow the rules.

### 12.2 J2EE Integration

- Java 2 Enterprise Edition collects together many APIs: Servlet, JSP, EJB, JavaMail, JMS, JTA, CORBA, JDBC, JAXP, JNDI, and adds some connection glue.

- It also recommends that development be broken down into roles:
  1. J2EE product provider: sells you the server, database system, etc.
  2. Application component provider: You, the one who writes the servlets, xhtml, etc.
  3. Application assembler: gets components in form appropriate for deployment. Sets up JNDI dependencies in `web.xml`.
  5. System administrator: configures and administers the network infrastructure.
  6. Tool provider: sells you the IDE, unit testing, etc.

### 12.3 Referencing External Resources

- The deployer should not change `web.xml`, so init parameters cannot be used for him.

- Instead, use the Java Networking and Directory Interface (JNDI API) to access external resources.

- Use environment entries like

```xml
<env-entry>
  !--
  <description>Send pincode by mail</description>
  !--used as part of the JNDI lookup-->
  <env-entry-name>mainPincode</env-entry-name>
  !--default value-->
  <env-entry-value>false</env-entry-value>
  !--the FQDN of the entry-->
  <env-entry-type>java.lang.Boolean</env-entry-type>
</env-entry>
```

- If the object you are referencing is an EJB component then use
<ejb-ref>
  <description>Cruise ship cabin</description>
  <ejb-ref-name>ejb/CabinHome</ejb-ref-name>
  <!-- Entity or Session: type of EJB component-->
  <ejb-ref-type>Entity</ejb-ref-type>
  <home>com.titan.cabin.CabinHome</home>
  <remote>com.titan.cabin.Cabin</remote>
</ejb-ref>

then the servlet can get a reference to it with:

InitialContext initCtx = new InitialContext();
Object ref = initCtx.lookup("java:comp/env/ejb/CabinHome");
CabinHome home = (CabinHome) PortableRemoteObject.narrow(ref, CabinHome.class);

13 Element Construction Set

- You can also create html by creating java objects using the [Element Construction Set](#).

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.apache.ecs.*;
import org.apache.ecs.html.*;

public class ECSHello extends HttpServlet {

  public void doGet(HttpServletRequest req, HttpServletResponse res)
      throws ServletException, IOException {

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

    Document doc = new Document();
    doc.appendTitle("Testing ECS");
    doc.appendBody(new Big("Hello!"))
      .appendBody(new P())
      .appendBody("The current time is " + new Date());
    doc.output(out);
  }
}

- Templates are better for mostly static content.
- ECS good for very dynamic part of page (extract from dbase).
14 JavaServer Pages

- You can include Java code in static pages. Code gets executed by special servlet and output replaces code.
- Code is between <%% and %>

```html
<HTML>
<HEAD><TITLE>Hello</TITLE></HEAD>
<BODY>
<H1><%if (request.getParameter("name") == null) {
  out.println("Hello World");
} else {
  out.println("Hello, " + request.getParameter("name"));
}%></H1>
</BODY></HTML>
```

- The servlet has access to several local variables: request, response, out, session, application, pageContext.
- The code is compiled when the first person accesses the page. This can be forced.
- You can denote expressions with <%%= %>. These get replaced by their println() value.
- Declarations are denoted with <%%>

```java
<%-- hello2.jsp --%>
<HTML>
<HEAD><TITLE>Hello</TITLE></HEAD>
<BODY>
<H1>Hello, <%= getName(request) %></H1>
</BODY>
</HTML>
<%!
private static final String DEFAULT_NAME = "World";

private String getName(HttpServletRequest req) {
    String name = req.getParameter("name");
    if (name == null)
        return DEFAULT_NAME;
    else
        return name;
}
%>
```

14.1 Directives

- You can denote directives with <@%>. They allow one to control aspects of the controlling servlet.
- They must be of the form

```html
<%@ directiveName attribName="attribValue" %>
```

were the directives are

1. contentType
2. import specifies classes to import.
3. buffer set buffer size.
4. autoFlush should we flush when full or raise exception.
5. session true if page wants access to the user’s session.
6. errorPage specifies error page to display if exception.

14.2 JavaBeans in JSP

- Writing Java code in display html is bad: hard to debug, if non-programmers do it its even worse.
- A JavaBean is just a class with getX()/setX() methods defined for its member variables (AKA properties).
- To embed a bean use <jsp:useBean>, as shown below (assume this page is GET’ed with "..?name=Jeff".

```html
<%@ page import="HelloBean" %>

<jsp:useBean id="hello" class="HelloBean">
  <jsp:setProperty name="hello" property="*" />
</jsp:useBean>

<HTML>
<HEAD><TITLE>Hello</TITLE></HEAD>
<BODY>
<H1>Hello, <jsp:getProperty name="hello" property="name" /></H1>
</BODY>
</HTML>
```

with the bean defined as

```java
public class HelloBean {
    private String name = "World";

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}
```

- The property="*" says to set all properties to values given by parameters.
- You could also use `<.. property="propertyname" param="paramName">` to set propertyname to some param’s value
- or `<.. property="propertyname" value="constant">` to set it to a constant.
14.3 Include and Forward

- You can include static files with

  `<%@ include file="filepath" %>`

- You can include the content generated by a servlet with

  `<jsp:include page="pathToServlet" flush="true"/>`

Notes

2. http://jmvidal.cse.sc.edu/talks/servlets/../internet/cgi.xml
3. http://jmvidal.cse.sc.edu/talks/servlets/gettinginformation.xml
11. http://jakarta.apache.org/ecs/

This talk is available at http://jmvidal.cse.sc.edu/talks/servlets

Copyright © 2004 Jose M Vidal. All rights reserved.