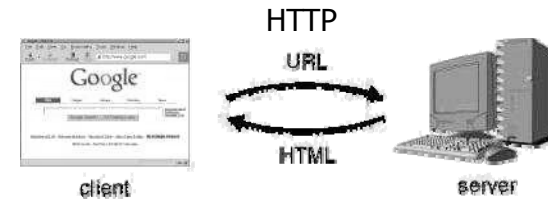


# Web Application Frameworks

Copyright © 2007 Anders Møller <amoeller@brics.dk>

## Components of a classical Web application

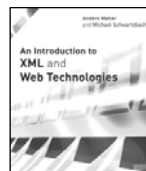


- HTML rendering
- browser scripting (JavaScript)
- database
- business logic
- HTML generation

2

## Overview

- Java Servlets
  - requests and responses
  - deployment
  - state
- JavaServer Pages (JSP)
  - JSP templates
  - the expression language
  - tag files and tag libraries
- JWIG
- Google Web Toolkit (GWT)



3

## Java Servlets and JSP – a brief summary



## Example: Hello World

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

public class HelloWorld extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html><head><title>ServletExample</title></head>"+
            "<body><h1>Hello world!</h1>"+
            "This page was last updated: "+
            new java.util.Date()+
            "</body></html>");
    }
}
```

**Hello World!**

This page was last updated: Fri Dec 24 19:38:23 CET 2004

5

## Deployment

A Web app is structured as a directory:

- *myapp/*
  - contains HTML/CSS/GIF/... files
- *myapp/WEB-INF/*
  - contains the **deployment descriptor** `web.xml`
- *myapp/WEB-INF/classes/*
  - contains servlet class files
- *myapp/WEB-INF/lib/*
  - contains extra jar files

6

## A simple deployment descriptor

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    version="2.4">
    <display-name>A Small Web Application</display-name>
    <servlet>
        <servlet-name>MyFirstServlet</servlet-name>
        <servlet-class>HelloWorld</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>MyFirstServlet</servlet-name>
        <url-pattern>/hello/*</url-pattern>
    </servlet-mapping>
</web-app>
```

7

## State

- One `ServletContext` object for each **Web application**  
(consider using a real database...)
- One `HttpSession` object for each **session**

Access state in these objects with

- `setAttribute("name", value)`
  - `getAttribute("name")`
- **Transient state** as local variables

8

## Example: A Polling Service

A Web application consisting of

- QuickPollQuestion.html
- QuickPollSetup.java
- QuickPollAsk.java
- QuickPollVote.java
- QuickPollResults.java

QuickPoll

What is your question?  
To be or not to be

Register my question

QuickPoll

To be or not to be?

yes  
 no

Vote

QuickPoll

To be or not to be?

Yes: ██████████ 4  
No: ████████ 1

9

## Example: QuickPollSetup.html

```
<html>
<head><title>QuickPoll</title></head>
<body>
<h1>QuickPoll</h1>
<form method=post action=setup>
What is your question?<br>
<input name=question type=text size=40?<br>
<input type=submit name=submit
value="Register my question">
</form>
</body>
</html>
```

QuickPoll

What is your question?  
To be or not to be

Register my question

10

## Example: QuickPollQuestion.java

```
public class QuickPollSetup extends HttpServlet {
    public void doPost(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException {
        String q = request.getParameter("question");
        ServletContext c = getServletContext();
        c.setAttribute("question", q);
        c.setAttribute("yes", 0);
        c.setAttribute("no", 0);
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.print("<html><head><title>QuickPoll</title></head><body>"+
            "<h1>QuickPoll</h1>"+
            "Your question has been registered. "+
            "Let the vote begin!"+
            "</body></html>");
    }
}
```

11

## Example: QuickPollAsk.java

```
public class QuickPollAsk extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.print("<html><head><title>QuickPoll</title></head><body>"+
            "<h1>QuickPoll</h1>"+
            "<form method=post action=vote>");
        String question =
            (String) getServletContext().getAttribute("question");
        out.print("<html><head><title>QuickPoll</title></head><body>"+
            "<h1>QuickPoll</h1>"+
            "Your question has been registered. "+
            "Let the vote begin!"+
            "</body></html>");
    }
}
```

QuickPoll

To be or not to be?

yes  
 no

Vote

12

## Example: QuickPollVote.java (1/2)

```
public class QuickPollVote extends HttpServlet {
    public void doPost(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException {
        String vote = request.getParameter("vote");
        ServletContext c = getServletContext();
        if (vote.equals("yes")) {
            int yes = (Integer)c.getAttribute("yes");
            yes++;
            c.setAttribute("yes", yes);
        } else if (vote.equals("no")) {
            int no = (Integer)c.getAttribute("no");
            no++;
            c.setAttribute("no", no);
        }
    }
}
```

13

## Example: QuickPollVote.java (2/2)

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.print("<html><head><title>QuickPoll</title></head><body>"+
    "<h1>QuickPoll</h1>"+
    "Thank you for your vote!"+
    "</body></html>");
}
}
```

14

## Example: QuickPollResult.java (1/2)

```
public class QuickPollResults extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException {
        ServletContext c = getServletContext();
        String question = (String)c.getAttribute("question");
        int yes = (Integer)c.getAttribute("yes");
        int no = (Integer)c.getAttribute("no");
        int total = yes+no;
        response.setContentType("text/html");
        response.setDateHeader("Expires", 0);
        response.setHeader("Cache-Control",
            "no-store, no-cache, must-revalidate");
        response.setHeader("Pragma", "no-cache");
        PrintWriter out = response.getWriter();
    }
}
```

### QuickPoll

To be or not to be?

Yes:  4  
No:  1

15

## Example: QuickPollResult.java (2/2)

```
out.print("<html><head><title>QuickPoll</title></head><body>"+
    "<h1>QuickPoll</h1>");
if (total==0)
    out.print("No votes yet...");
else {
    out.print(question + "?<p>"+<table border=0>"+
        "<tr><td>Yes:<td>"+drawBar(300*yes/total)+"<td>"+yes+
        "<tr><td>No:<td>"+drawBar(300*no/total)+"<td>"+no+
        "</table>");
}
out.print("</body></html>");
}

String drawBar(int length) {
    return "<table><tr><td bgcolor=black height=20 width="+
        length+"></table>";
}
}
```

### QuickPoll

To be or not to be?

Yes:  4  
No:  1

16

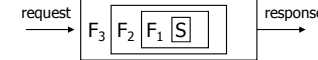
## Problems in QuickPoll

- Need **access control** to QuickPollSetup
- No **escaping of special characters**
- Need to **check right order of execution**
- Need to check that expected **form field data** is present
- No **synchronization** in QuickPollVote
- Should store state in **database**
- **Redundancy** in HTML generation

17

## Filters

- Code being executed before and after the servlet
  - executed in stack-like fashion with e.g. a servlet at the bottom
- Can **intercept** and **redirect** processing
  - security
  - auditing
- Can **modify requests and responses**
  - data conversion (XSLT, gzip, ...)
  - specialized caching



– all without changing the existing servlet code

18

## Overview

- Java Servlets
  - requests and responses
  - deployment
  - state
- JavaServer Pages (JSP)
  - JSP templates
  - the expression language
  - tag files and tag libraries
- JWIG
- Google Web Toolkit (GWT)

19

## Motivation

- Servlets make heavy use of Java
- HTML is generated from string fragments
- JSP templates are **HTML/XML pages with embedded code**
- With the **expression language** and **tag libraries**, JSP templates can be 100% free of Java code!

20

## JSP template example

```
<% response.addDateHeader("Expires", 0); %>
<html>
  <head><title>JSP</title></head>
  <body>
    <h1>Hello world!</h1>
    <%! int hits = 0; %>
    You are visitor number
    <% synchronized(this) { out.println(++hits); } %>
    since the last time the service was restarted.
    <p>
    This page was last updated:
    <%= new java.util.Date().toLocaleString() %>
  </body>
</html>
```

**Hello World!**

You are visitor number 43 since the last time the service was restarted.  
This page was last updated: 28-02-2005 13:56:49

21

## JSP templates, generally

- A text file with snippets of Java code:
  - *expressions* <%= ... %>
  - *statements* <% ... %>
  - *declarations* <%! ... %>
- JSP directives (include, import, errorPage, ...)
- Implicitly declared variables:
  - HttpServletRequest **request**;
  - HttpServletResponse **response**;
  - HttpSession **session**;
  - ServletContext **application**;
  - ServletConfig **config**;
  - JspWriter **out**;
  - PageContext **pageContext**;

JSP templates  
are translated  
into servlets

22

## The "expression language"

- The syntax **`${exp}`** may be used in
  - template text
  - attribute values in markup
- The expression may access
  - variables in the various scopes
  - implicit objects, such as param
- The usual operators on strings, numbers, and booleans are available

```
<html>
  <head><title>Addition</title></head>
  <body bgcolor="${param.color}">
    The sum of ${param.x} and ${param.y} is ${param.x+param.y}
  </body>
</html>
```

**no Java code!**

23

## Tag files

– define **abstractions as new tags**

```
wrap.tag:
<%@ tag %>
<%@ attribute name="title" required="true" %>
<html>
  <head><title>${title}</title></head>
  <body>
    <jsp:doBody/>
  </body>
</html>
```

The sum of `${param.x}` and `${param.y}` is `${param.x+param.y}`

```
<html>
  <head><title>Addition</title></head>
  <body>
    The sum of ${param.x} and ${param.y} is
    ${param.x+param.y}
  </body>
</html>
```



24

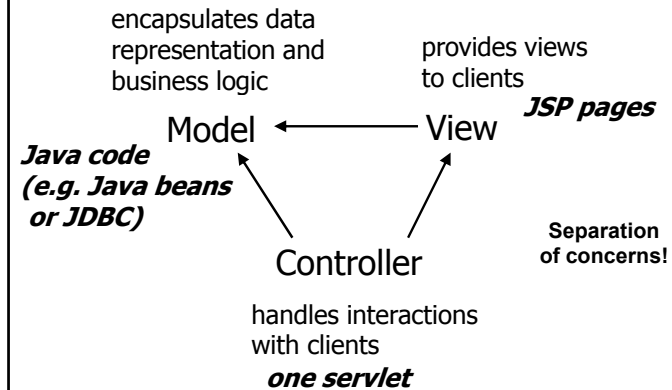
## Tag libraries

Libraries of tags capturing common patterns:

- general, but simple programming tasks (**JSTL**)
  - pagination of large texts
  - date and times
  - database queries
  - regular expressions
  - HTML scraping
  - bar charts
  - cookies
  - e-mail
  - WML
  - ...
- essentially a collection of domain-specific languages

25

## The Model-View-Controller pattern



26

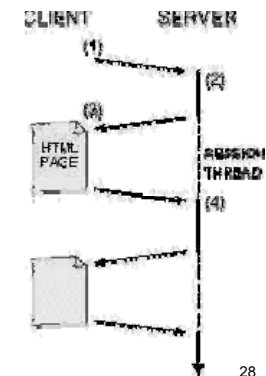
## Summary

- Servlets closely follow the **request-response** pattern from HTTP
- JSP templates are **HTML/XML pages with embedded code**
- The simple **expression language** is often sufficient in place of general Java code
- **Tag files and libraries** allow code to be hidden under a tag-like syntax
- **MVC** provides separation of programming and HTML/XML design tasks with Servlets + JSP

27

## Some limitations of Servlets/JSP

- **No compile-time guarantees of HTML/XML validity**
- **Session management is complicated:**
  - control-flow is often unclear
  - no enforcement of relation between showing a page and receiving form input
  - primitive session state management



28

## Other Web application frameworks

- Struts
- Tapestry
- Cocoon
- Spring
- JSF
- RIFE
- Links
- **JWIG**
- (Ruby on) Rails
- Seaside
- Velocity
- **Google Web Toolkit**
- Wicket
- Stripes
- Seam
- Trails
- Echo2
- Barracuda
- Water
- ... etc.

29

# JWIG



BRICS

Powered by  
BRICS JWIG

## JWIG

### 1. Template-based page construction using XACT

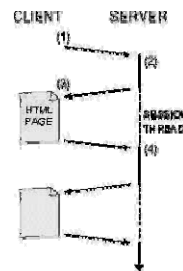
### 2. Session threads

– showing a page and receiving form input modeled as a Remote Procedure Call (RPC)

- explicit control-flow!
- simpler session state management!

### Static checking of

- *output validity*
- *form field consistency*



31

## Example: a guessing game in JWIG

Please guess a number  
between 1 and 100:  
42 continue

You got it, using 428 guesses.  
That makes you the new record holder!  
Please enter your name for the hi-score list:  
John Doe continue

In 2 plays of this game, the record  
holder is **John Doe** with 428 guesses.

32

## GuessingGameWrapper.xml

```
<html>
<head><title>The Guessing Game</title></head>
<body bgcolor="aqua"><[BODY]></body>
</html>
```

33

## GuessingGame (1/5)

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.jwig.*;
import dk.brics.xact.*;

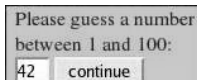
public class GuessingGamePlay extends SessionThread {
    public XML main() throws IOException, ServletException {
        XML wrapper = XML.loadConstant("GuessingGameWrapper.xml");
        XML form = [[
            <form><input name="guess" type="text" size="2" maxlength="2"/>
            <input type="submit" name="continue" value="continue"/></form>
        ]];
```

34

## GuessingGame (2/5)

```
ServletContext c = getServletContext();
Integer plays = (Integer)c.getAttribute("plays");
if (plays==null)
    plays = 0;
else
    plays = plays + 1;
c.setAttribute("plays", plays);
int number = (new Random()).nextInt(100)+1;

show(wrapper.plug("BODY",
    [[Please guess a number between 1 and 100: <form>]]));
```



35

## GuessingGame (3/5)

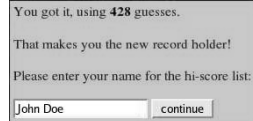
```
int guesses = 1;
boolean done = false;
while (!done) {
    int guess = Integer.parseInt(getParameter("guess"));
    if (guess==number)
        done = true;
    else {
        show(wrapper.plug("BODY", [[
            That is not correct. Try a
            <b><{(guess>number)?"lower":"higher"}</b> number: <form>
        ]]));
        guesses++;
    }
}
```

Note: "show" inside a while loop is difficult with basic Servlets/JSP

36

## GuessingGame (4/5)

```
XML msg = [[You got it, using <b><{guesses}></b> guesses.]];
XML thanks = [[Thank you for playing this exciting game!]];
XML res;
if (guesses.getCurrentRecord() {
  show(wrapper.plugin("BODY", [[
    <{msg}><p/>
    That makes you the new record holder!<p/>
    Please enter your name for the hi-score list:
    <form><input name="name" type="text" size="20"/>
    <input type="submit" name="continue" value="continue"/></form>
  ]]));
  synchronized(c) {
    if (guesses.getCurrentRecord() {
      c.setAttribute("holder", getParameter("name"));
      c.setAttribute("record", new Integer(guesses));
    }
  }
  res = wrapper.plugin("BODY", thanks);
} else
  res = wrapper.plugin("BODY", [[<{msg}><p/><{thanks}>]]);
return res; // terminates the session thread
}
```



37

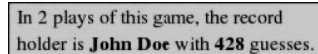
## GuessingGame (5/5)

```
int getCurrentRecord() {
  Integer record = (Integer)c.getAttribute("record");
  if (record!=null)
    return record;
  else
    return Integer.MAX_VALUE; // no players yet
}
}
```

38

## GuessingGameHiScore

```
public class GuessingGameHiScore extends HttpServlet {
  public void doGet() throws IOException, ServletException {
    ServletContext c = getServletContext();
    Integer plays = (Integer)c.getAttribute("plays");
    String holder = (String)c.getAttribute("holder");
    Integer record = (Integer)c.getAttribute("record");
    XML body;
    if (record!=null)
      body = [[In <{plays}> plays of this game,
        the record holder is <b><{holder}></b> with
        <b><{record}></b> guesses.]];
    else
      body = [[No players yet.]];
    XML.loadConstant("GuessingGameWrapper.xml")
      .plugin("BODY", body).write(response.getWriter());
  }
}
```



39

## Catching errors at compile time

```
...
XML ask = [[ <form>Your name? <input name="NAME"/>
  <input type="submit"/></form> ]];
...
*** Field 'NAME' is never available on line 15
*** Invalid XHTML at line 14
--- element 'input': requirement not satisfied:
<or>
  <attribute name="type">
    <union>
      <string value="submit" />
      <string value="reset" />
    </union>
  </attribute>
  <attribute name="name" />
</or>
```

40

## More information

- <http://www.brics.dk/JWIG/>  
(not up-to-date – a new implementation is under development...)
- Aske Simon Christensen, Anders Møller, and Michael I. Schwartzbach, **Extending Java for High-Level Web Service Construction**, *ACM TOPLAS*, 25(6), 2003

41

# Google Web Toolkit



## Motivation

- Traditional browsers just show HTML pages
  - the HTML pages are generated by a server
  - user interaction is based on forms
- Modern browsers support **JavaScript!**
  - dynamically modify the HTML page
  - react to events (keyboard, mouse, back button, ...)
  - communicate with the server anytime!

43

## Main ideas in GWT

- Execute code on the **client side**, to the extent possible
- Program in Java, **compile to JavaScript**
- **Swing-like** GUI programming
  - widgets and panels instead of "raw" HTML
  - events and listeners
  - CSS for styling
- **Asynchronous RPC** mechanism (AJAX) for client-server communication

44

## Advantages

- **"Live" pages** without JavaScript hacking
- The approach works well for other GUIs – why not also for Web browsers?
- The server-side is reduced to being a database frontend

45

## Considerations

- **Security:** we can't always trust the clients to execute the code properly!  
(so be careful what functionality is provided by the server)
- **HTML** and **JavaScript** code is hidden away  
(and making custom widgets is nontrivial)
- The compiler doesn't support full **Java**

46

## Example: the guessing game in GWT

- GuessingGame.gwt.xml
- public/GuessingGame.html  
  GuessingGameHiScore.html } entry points
- client/**GuessingGame.java**  
  GuessingGameHiScore.java } main app code  
  GuessingService.java  
  GuessingServiceAsync.java } RPC
- server/GuessingServiceImpl.java

Please guess a number between 1 and 100:  
42

You got it, using **428** guesses.  
That makes you the new record holder!

Please enter your name for the hi-score list:

John Doe

In 2 plays of this game, the record holder is **John Doe** with **428** guesses.

for security reasons, we must store *session state* and *application state* on the **server!**

47

## GuessingGame (1/6)

```
package dk.brics.client;

import com.google.gwt.core.client.*;
import com.google.gwt.user.client.ui.*;
import com.google.gwt.user.client.rpc.*;

public class GuessingGame implements EntryPoint {
    GuessingServiceAsync server;
    int id; // session ID
    int guesses; // number of guesses
    VerticalPanel vp;
    HorizontalPanel hp;
    TextBox text_field;
    HTML text;
    boolean done;
    final String thanks = "Thank you for playing this exciting game!";
```

48

## GuessingGame (2/6)

```
public void onLoad() {
    server = (GuessingServiceAsync)GWT.create(GuessingService.class);
    String url = GWT.getModuleBaseURL() + "guessing";
    ((ServiceDefTarget)server).setServiceEntryPoint(url);

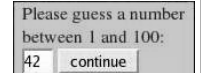
    server.getSessionID(new AsyncCallback() {
        public void onSuccess(Object r) {
            id = ((Integer)r).intValue(); // each session gets a unique ID
            play();
        }
        public void onFailure(Throwable t) { window.alert("RPC error!"); }
    });
}
```

49

## GuessingGame (3/6)

```
void play() {
    guesses = 0;
    vp = new VerticalPanel();
    hp = new HorizontalPanel();
    text_field = new TextBox();
    text = new HTML("Please guess a number between 1 and 100:");
    vp.add(text);
    vp.add(hp);
    hp.setSpacing(5);
    hp.add(text_field);
    hp.add(new Button("continue", new ClickListener() {
        public void onClick(Widget sender) {
            cont();
        }
    }));
    DOM.setStyleAttribute(RootPanel.getBodyElement(),
        "backgroundColor", "aqua");

    done = false;
    RootPanel.get().add(vp);
}
```



50

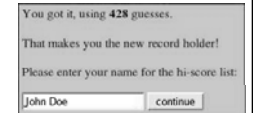
## GuessingGame (4/6)

```
void cont() {
    if (!done) {
        int guess = Integer.parseInt(text_field.getText());
        guesses++;
        server.guess(id, guess, new AsyncCallback() {
            public void onSuccess(Object r) {
                guess(((Integer)r).intValue()); // <0=too low, 0=got it, >0=too high
            }
            public void onFailure(Throwable t) { window.alert("RPC error!"); }
        });
    } else {
        text.setHTML(thanks);
        vp.remove(hp);
        server.storeName(id, text_field.getText(), new AsyncCallback() {
            public void onSuccess(Object r) {}
            public void onFailure(Throwable t) { window.alert("RPC error!"); }
        });
    }
}
```

51

## GuessingGame (5/6)

```
void guess(int status) {
    if (status==0) {
        final String msg = "You got it, using <b>" +
            guesses + "</b> guesses.";
        server.getRecord(new AsyncCallback() {
            public void onSuccess(Object r) {
                int current_record = ((Integer)r).intValue();
                if (guesses<current_record) {
                    text.setHTML(msg + "<p>" +
                        "That makes you the new record holder!<p>" +
                        "Please enter your name for the hi-score list:");
                    done = true;
                } else {
                    text.setHTML(msg + "<p>" + thanks);
                    vp.remove(hp);
                }
            }
            public void onFailure(Throwable t) { window.alert("RPC error!"); }
        });
    }
}
```



52

## GuessingGame (6/6)

```
} else {
    text.setHTML("That is not correct. Try a <b>" +
        (status>0?"lower":"higher") + "</b> number:");
}
text_field.setText("");
}
```

53

## GWT issues, indicated by the example

- **State management** becomes complicated!
  - *Asynchronous RPC* is powerful but often leads to fragmented source code
- **"GUI programming"** is tedious!
  - and HTML knowledge is still required (the HTML widget...)

54

## Conclusion

- **Servlets+JSP** illustrate basic server-oriented Web programming
  - **JWIG** is an ongoing research project that adds **XACT** and session threads
  - **GWT** is an example of a client-oriented framework that turns the browser into "yet another GUI platform"
- ... and there are *many* other approaches to Web application programming!

55