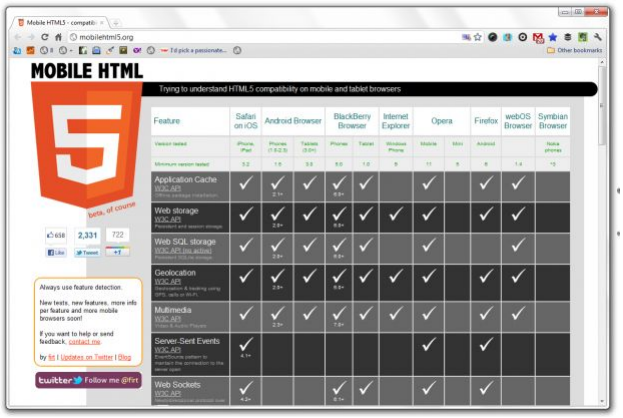
Java Server Pages

BWA Slides #4

Content

- Dynamic content
- Platforms
- Stacks
- Echo systems
- JEE concepts similar in PHP etc.

HTML is Static



Major problem with HTML

- No way to put dynamic data in pages using pure HTML!

Dynamic Content Solutions

PHP

```
<!DOCTYPE html>
<html>
    <body>
        <h1>PHP</h1>
        <?php
        $result = ...;
        echo $result;
        ?>
    </body>
</html>
```

ASP.NET

```
<!DOCTYPE html>
<html>
    <body>
        <h1>ASP.NET</h1>
        <p>
            The time is @DateTime.Now
        </p>
    </body>
</html>
```

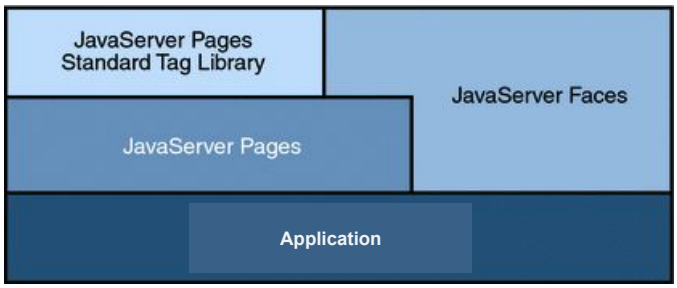
JEE

```
<!DOCTYPE html>
<html>
    <body>
        <h1>JEE</h1>
        <p>
            ${contact.name}
        </p>
    </body>
</html>
```

Similar approach for many platforms (if using a request based approach …)

- Let page composer write HTML with some added tags or expressions
- Server intercept page at request and replace expression with values (toString)
- Server sends processed page to client

JEE View Technologies



NOTE: Servlet is not a view technology any more

Java Server Pages

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
       ...
     </head>
    <body>
        <h1>Detail</h1>
        <table>
            <tr>
                <td>
                    ${contact.id}
                </td>
                 <td>
                    ${contact.name}
                </td>
            </tr>
        </table>
        <button onclick="window.history.back();">Back</button>
    </body>
</html>
```

Could use a Servlets for dynamic content but bad …

- … mixing Java and HTML
- Tedious!
- Need something else (simpler)

JSP is a JEE display technology for dynamic HTML pages.

- Simple for non-programmers to write a "page"
- Page consists of a <%@page..> directive, ordinary HTML and expressions from the Expression Language
- Possibly some special JSP tags like <jsp:include> (not shown, more later)
- Two different JSP syntaxes, we use Standard syntax (non-XML)
- Under the hood: A JSP is a Servlet, … but written in a "tag language"!

Lifecycle

- At first request translated to servlet and compiled (slow first request)
- Then as Servlet

Possible to write Java-code in pages (scriptlets), we never do!
Separations of concerns (unmaintainable)

In NetBeans JSPs are located under the "Web Pages" folder (Maven: src/main/webapp)

- Best practices: Put in src/main/webapp/WEB-INF/jsp directory i..e. application private, can't access using browser address field (that is: must go through application, we have control)

Expression Language

Retrieval

```
${person.name}
   \_______/\____/
       |       |
Key for object     attribute (will call getName on some object)
```

String concatenation

```
"/shop/products/list/${CURRENT_PAGE}"
```

Arithmetic

```
"${CURRENT_PAGE lt COUNT / PAGE_SIZE - 1}"
```

The EL allows page authors to write simple expressions to access Java objects and data from Java objects

- Used in JSP pages and elsewhere, more later …
- Also possible: String manipulation and arithmetic (and more…)
- In EL anything will end up as an expression (a value, … converted to a String)

Immediate evaluation for EL-expressions

- Expression immediately evaluated and result put into page
- Immediate syntax: ${ … *expression* … }
- Immediate syntax is read only (good, this is the view part)

Implicit Objects

```
<%@page contentType="text/html"
pageEncoding="UTF-8"%>
!DOCTYPE html>
<html>
  ${requestScope}


          ${sessionScope}


   ${pageScope}


         ${param}

  ${applicationScope}


</html>
```

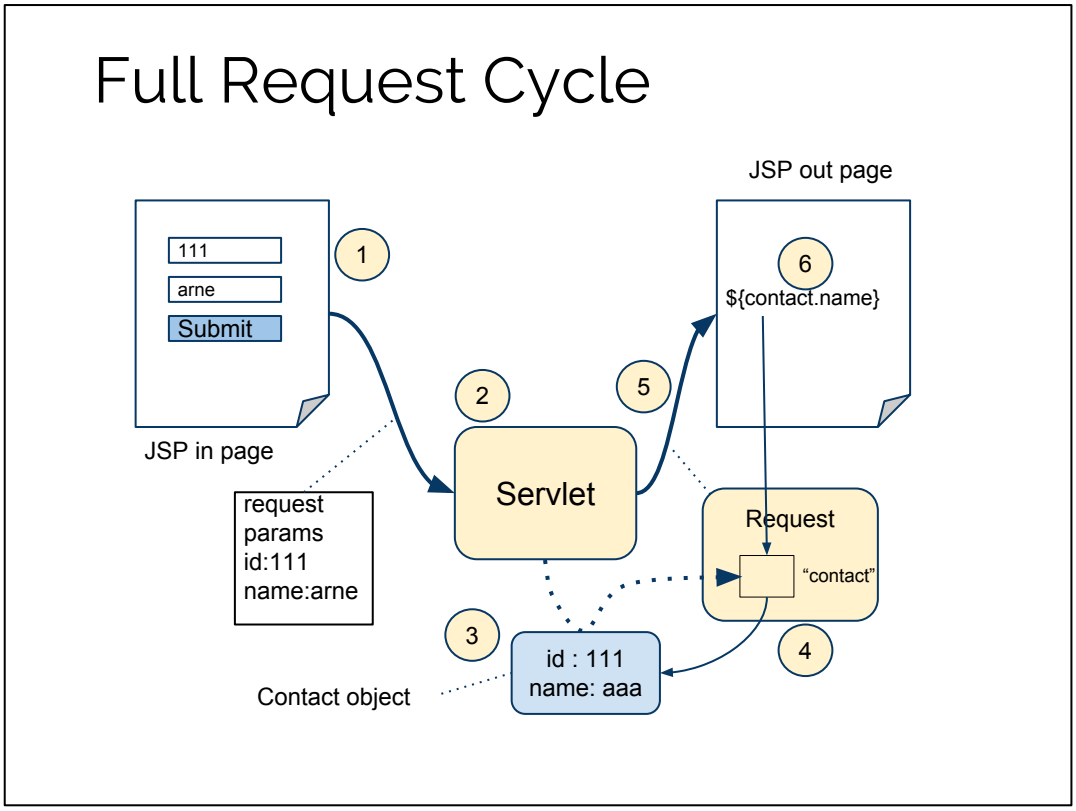
"Implicit objects" are Java objects accessible in JSP page using the expression language

Some implicit objects (there are more)

- pageScope - a Map that maps page-scoped attribute names to their values
- requestScope - a Map that maps request-scoped attribute names to their values
- sessionScope - a Map that maps session-scoped attribute names to their values
- applicationScope - a Map that maps application-scoped attribute names to their values
- param - a Map that maps parameter names to a single String parameter value (obtained by calling ServletRequest.getParameter(String name))
- pageContext, entry to request, response (i.e. the scoped objects used in servlets) and more …
    - ${pageContext.request.contexPath}, will return root of application
    - … useful when creating links (will make it possible to relocate application)

Example:

- Use request.setAttribute("key", …) in Servlet,
- accessing in JSP as ${requestScope.key…}
    - Or ${requestScope[key]…}



Full request cycle example

1. Enter some data into a form in JSP. Post to Servlet
2. Servlet retrieve request parameters and …
3. … creates Contact object, from parameters (normally read/write from/to database)
4. Servlet puts Contact object into request (scoped object in Servlet) using key "contact", (request.setAttribute("contact", object))
5. Servlet forwards request to JSP page (this also sends the request along)
   - Or redirects, but then request won't survive … problems …
6. Page retrieve data using EL

NOTE: Page implicit objects normally automagically searched for key/values (i.e. don't need to specify implicit object, just specify key).

Java Standard Template Library

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
    <head>... </head>
    <body>
        <!-- Generating a list -->
        <table>
            <c:forEach var="contact" items="${contacts}">
                <tr>
                    <td>${contact.id} </td>
                    <td>${contact.name}</td>
                    <td> … </td>
                </tr>
            </c:forEach>
        </table>
    </body>
</html>
```

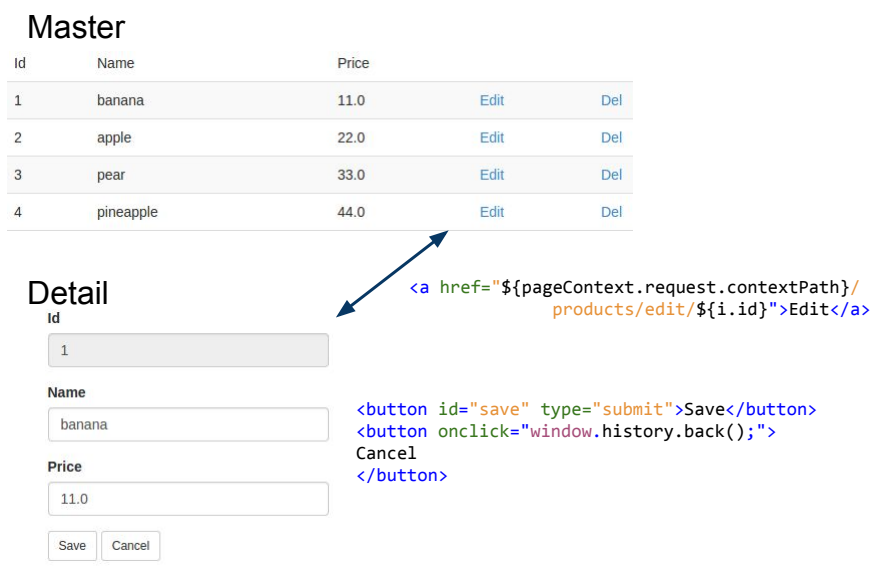
Need statements in pages: assignment, selection, iteration

- Typically a loop to create a table or …
- … a selection to set button/links disables or not
- Use: Of course no application logic, just display logic

Solution: JSTL tags

- Used in conjunction with EL
- Must add directive (<%@taglib,,,>) to page to be able to use JSTL
- We only need the <c: … > tags
- May need Maven dependency on JSTL

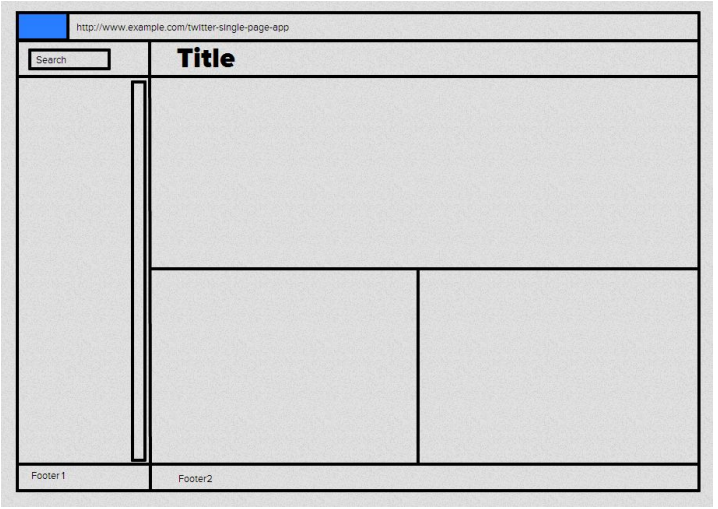
Master Detail Interface



Very common GUI design

- Master consisting of a list.
- When editing we would like to display the details of a single row, open new page (or popup)

HTML is Not Modular



To get a common layout for a site we have common parts for all pages (footer, header)

- To be able to refactor out common parts we need to be able to compose pages
- No way in pure HTML

Composing Pages

Template page (single JSP)

```
<jsp:include page="${param.partial}.jsp" />
```

Partials (many JSPs)

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<section id="pageSection">
  ... content here ...
</section>
```

Use in Servlet

```
case “edit”
req.getRequestDispatcher(“template.jsp?partial=edit”)
                                        .forward(req, resp);
```

Modular JSP Pages consist of composing:

- A single template page for the overall look (common header, menu, footer, etc) using <jsp:include> tag for varying content
  - Using filenames of the varying content (partials)
- Many “partial” JSP-pages containing specific content to display in template (varying part as an HTML <section>),

A servlet informing the template which content to put in the varying part (depending on incoming request)

JSP, CSS and JavaScript

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
...
<head>
<c:set var="res" value="${pageContext.request.contextPath}/resources"/>
      <link rel="stylesheet" type="${res}/css" href="normalize.css" />
      <link rel="stylesheet" type="${res}/css" href="bootstrap.css" />
      <script  type="text/javascript" src="${res}/js/jquery-1.11.3.min.js"></script>
      </head>
<body>
...
</body>
...
```

No problems, just as HTML

- All CSS, JS , images in top level resource folder

URIs In JSPs

Browser URI

```
http://localhost:8084/shop/
```

Relative references inside application

1. `shop/products/list/0`
2. `/shop/products/list/0` (hardcoded)
3. `${pageContext.request.contextPath}/products/list/0` (dynamic)

Resolving

1. `http://localhost:8084/shop/shop/products/list/0` (bad)
2. `http://localhost:8084/shop/products/list/0`
3. `http://localhost:8084/shop/products/list/0`

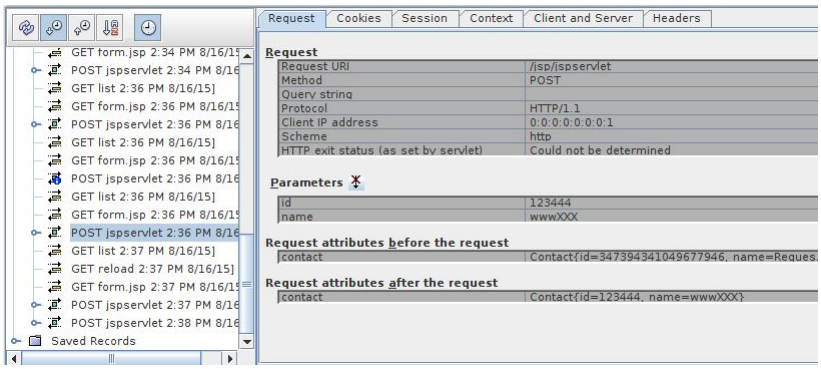
Relative references makes it easy to move resources …

- But inside an application very confusing

If using relative references (without leading /) resolved reference depends on previous URI

- URI in page appended to the URI we used to access the resource…
- Simpler to use /, … whatever URI used will be resolved from server root, but hard to move application
- Best is resolving root dynamically: using request.getContextPath()

HTTP Monitor



To trace the request cycle try HTTP Monitor

- Nice tool in NetBeans to check incoming calls to container
- Select Services > Servers > Tomcat (or other) > Properties > Check Enable HttpMonitor. Restart server