JSP

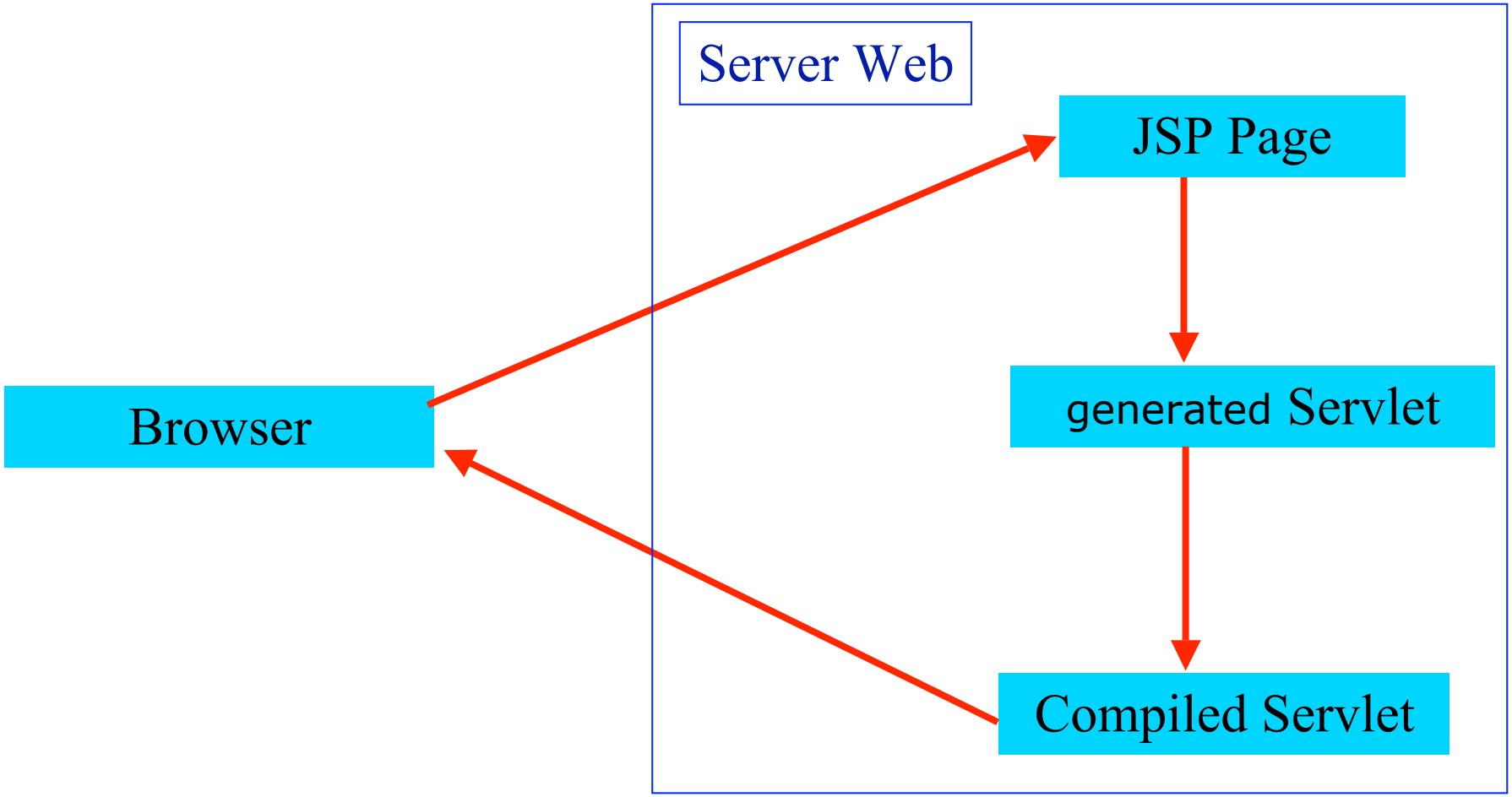
Basic Elements

For a Tutorial, see:
http://java.sun.com/j2ee/1.4/docs/tutorial/doc/JSPIntro.html

Simple.jsp

```
<html>
 <body>
    <% out.println(“Hello World”); %>
  </body>
</html>
```

JSP Lifecycle



JSP nuts and bolts

Syntactic elements:

<%@ directives %>

<%! declarations %>

<% scriptlets %>

<%= expressions %>

<jsp:actions/>

<%-- Comment --%>

Implicit Objects:

- **request**
- **response**
- **pageContext**
- **session**
- **application**
- **out**
- **config**
- **page**

JSP nuts and bolts

Syntactic elements:

<%@ directives %> → **Interaction with the CONTAINER**

<%! declarations %> → **In the initialization of the JSP**

<% scriptlets %> → **In the service method**

<%= expressions %> → **In the service method**

<jsp:actions/>

Scriptlets

A scriptlet is a block of Java code executed during the request-processing time.

In Tomcat all the scriptlets gets put into the service() method of the servlet. They are therefore processed for every request that the servlet receives.

Scriptlet

Examples :

```
<% z=z+1; %>
```

```
<%
      // Get the Employee's Name from the request
      out.println("<b>Employee: </b>" +
      request.getParameter("employee"));
      // Get the Employee's Title from the request
      out.println("<br><b>Title: </b>" +
      request.getParameter("title"));
 %>
```

Expressions

An expression is a shorthand notation that sends the evaluated Java expression back to the client (in the form of a String).

Examples :

<%= getName() %>

<%@ page import=java.util.* %>

Sono le <%= new Date().toString(); %>

Expressions

```
<html><body>
 <%! String nome=“pippo” %>
<%! public String getName() {return nome;} %>
<H1>
Buongiorno
<%= getName() %>
</H1>
 </body></html>
```

Declarations

A declaration is a block of Java code used to:

define class-wide variables and methods in the generated servlet.

They are initialized when the JSP page is initialized.

<%! DECLARATION %>

Examples:

```
<%! String nome="pippo"; %>

<%! public String getName() {return nome;} %>
```

Directives

A directive is used as a message mechanism to:

pass information from the JSP code to the container

Main directives:

page

include (for including other STATIC resources at compilation time)

taglib (for including custom tag libraries)

Directives

<%@ DIRECTIVE {attributo=valore} %>

main attributes:

<%@ page language=java session=true %>

<%@ page import=java.awt.*,java.util.* %>

<%@ page isThreadSafe=false %>

<%@ page errorPage=URL %>

<%@ page isErrorPage=true %>

Standard actions

Standard action are tags that affect the runtime behavior of the JSP and the response sent back to the client.

<jsp:include page="URL" />

For including STATIC or DYNAMIC resources at request time

<jsp:forward page="URL" />

What is a Java bean?

A bean is a Java class that:

- Provides a public no-argument constructor
- Implements java.io.Serializable
- Follows JavaBeans design patterns
  - Has Set/get methods for properties
  - Has Add/remove methods for events
    - Java event model (as introduced by JDK 1.1)
- Is thread safe/security conscious
  - Can run in an applet, application, servlet, ...

```
public class SimpleBean implements Serializable {
    private int counter;
    SimpleBean() {counter=0;}
    int getCounter() {return counter;}
    void setCounter(int c) {counter=c;}
}
```

See
http://java.sun.com/developer/onlineTraining/Beans/JBeansAPI/shortcourse.html

Standard actions involving beans

<jsp:useBean id="name" class="fully_qualified_pathname"
scope="{page|request|session|application}" />

<jsp:setProperty name="bean-name" property="prop-name" value="value" />

<jsp:getProperty name="nome" property="prop-name" />

<%@include@%> or <jsp:include> ?

When should I use a JSP <%@include@%> directive, and when should I use a <jsp:include> action?

A JSP <%@include@%> directive (for example, <%@ include file="myfile.jsp" @%>) includes literal text "as is" in the JSP page and is not intended for use with content that changes at runtime. The include occurs only when the servlet implementing the JSP page is being built and compiled.

The <jsp:include> action allows you to include either static or dynamic content in the JSP page. Static pages are included just as if the <%@include@%> directive had been used. Dynamic included files, though, act on the given request and return results that are included in the JSP page. The include occurs each time the JSP page is served.

See also

http://java.sun.com/blueprints/qanda/web_tier/index.html#directive

<%-- Comment --%> or <!-- Comment --> ?

When should I use JSP-style comments instead of HTML-style comments?

Always use JSP-style comments unless you specifically want the comments to appear in the HTML that results from serving a JSP page.

JSP-style comments are converted by the JSP page engine into Java comments in the source code of the servlet that implements the JSP page. Therefore, JSP-style comments don't appear in the output produced by the JSP page when it runs. HTML-style comments pass through the JSP page engine unmodified. They appear in the HTML source passed to the requesting client.

JSP-style comments do not increase the size of the document that results from the JSP page, but are useful to enhance the readability of the JSP page source, and to simplify debugging the servlet generated from the JSP page.

(taken from:
http://java.sun.com/blueprints/qanda/web_tier/index.html#comments

Predefined Objects

out	Writer
request	HttpServletRequest
response	HttpServletResponse
session	HttpSession
page	this nel Servlet
application	servlet.getServletContext
	area condivisa tra i servlet
config	ServletConfig
exception	solo nella errorPage
pageContext	sorgente degli oggetti, raramente usato

request

```
<%@ page errorPage="errorpage.jsp" %>
<html>
  <head>
    <title>UseRequest</title>
  </head>
  <body>
    <%
        // Get the User's Name from the request
        out.println("<b>Hello: " + request.getParameter("user") + "</b>");
    %>
  </body>
</html>
```

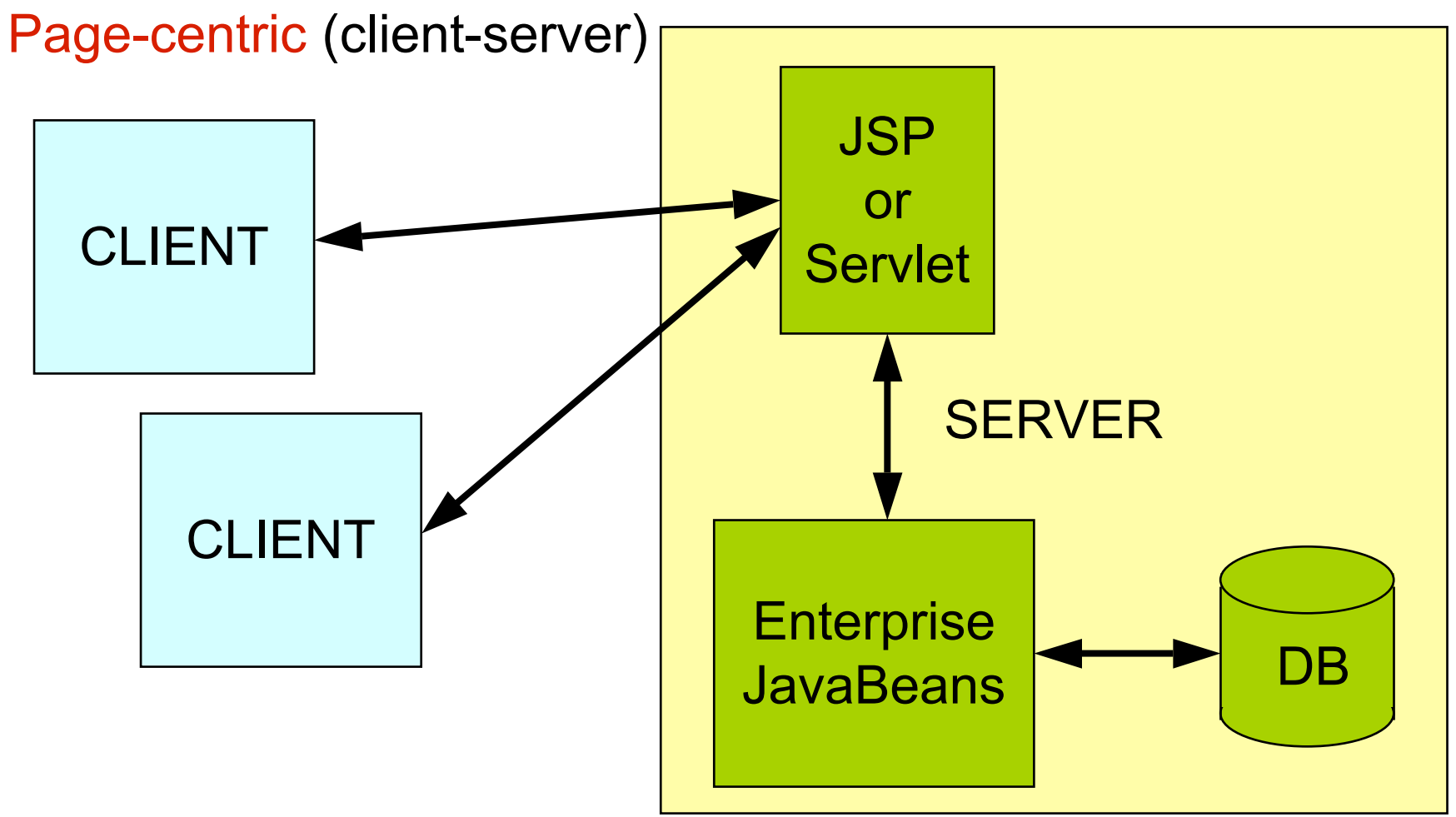
session

```
<%@ page errorPage="errorpage.jsp" %>
<html> <head> <title>UseSession</title> </head> <body>
   <%
    // Try and get the current count from the session
    Integer count = (Integer)session.getAttribute("COUNT");
    // If COUNT is not found, create it and add it to the session
    if ( count == null ) {
      count = new Integer(1);
      session.setAttribute("COUNT", count);
    } else {
      count = new Integer(count.intValue() + 1);
      session.setAttribute("COUNT", count);
    }
    // Get the User's Name from the request
    out.println("<b>Hello you have visited this site: " + count + " times. </b>");
   %>
 </body> </html>
```

JSP

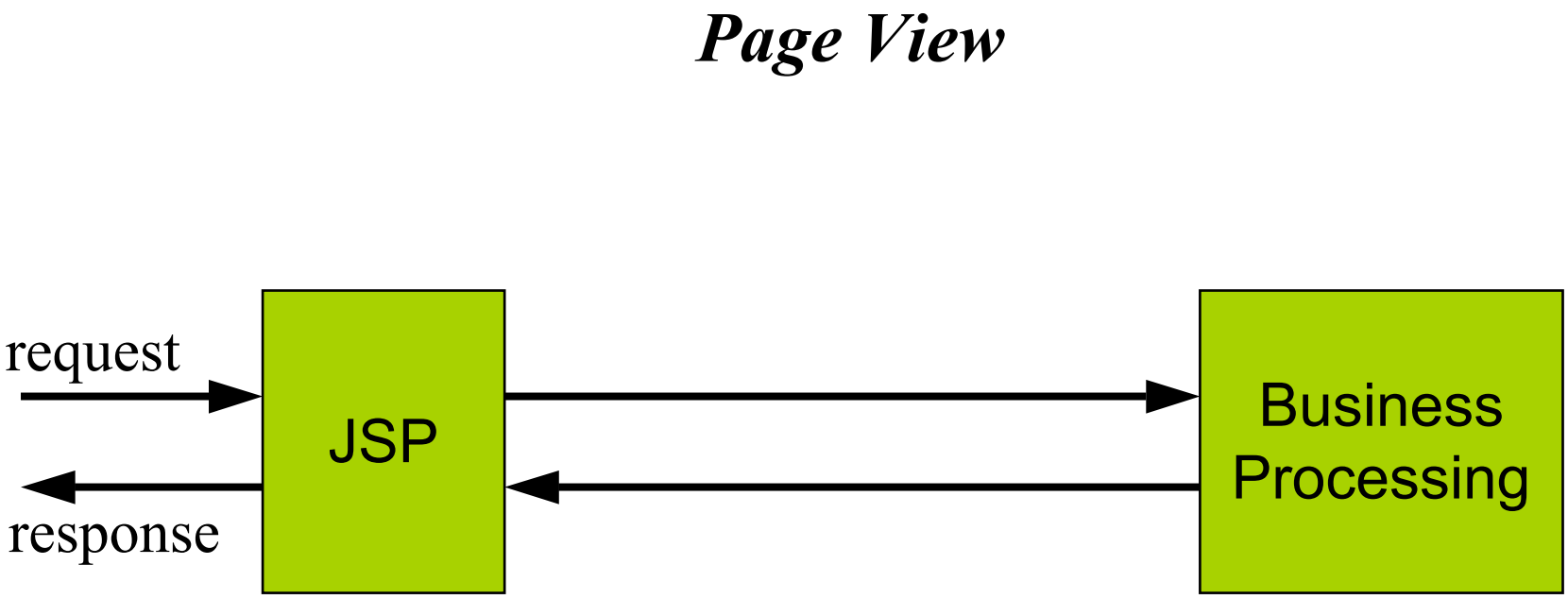
Common patterns

Common JSP patterns



Common JSP patterns

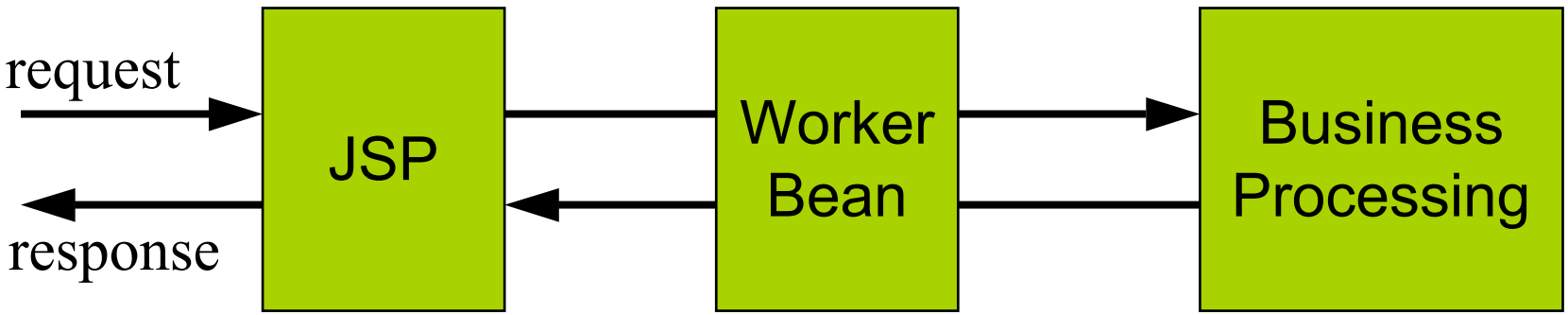
Page-centric 1 (client-server)



Common JSP patterns

Page-centric 2 (client-server)

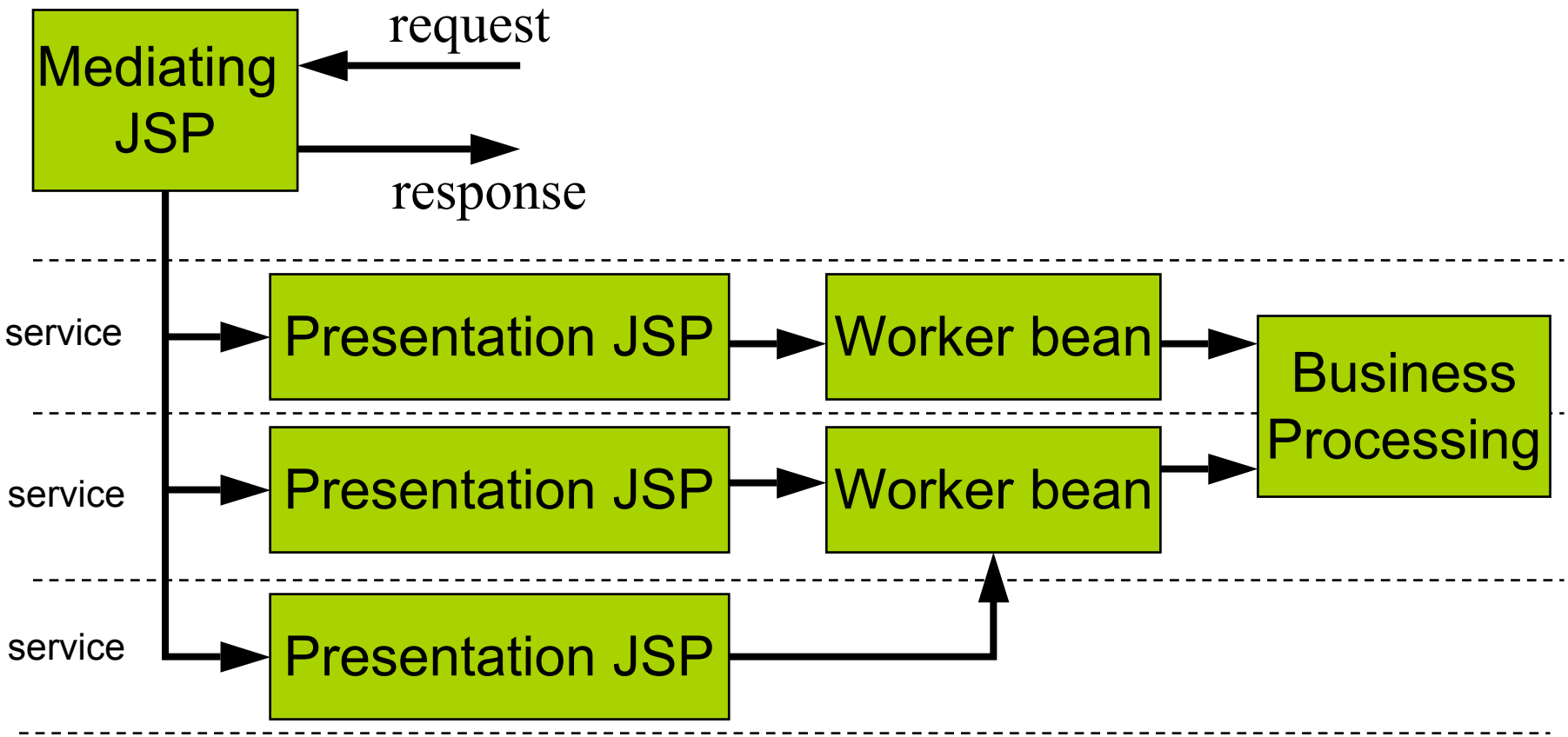
Page View with Bean



Common JSP patterns

Dispatcher (n-tier)

Mediator - View



WebApps
(Tomcat configuration)

JSP pages

To let Tomcat serve JSP pages, we follow the same procedure that we described for static pages.

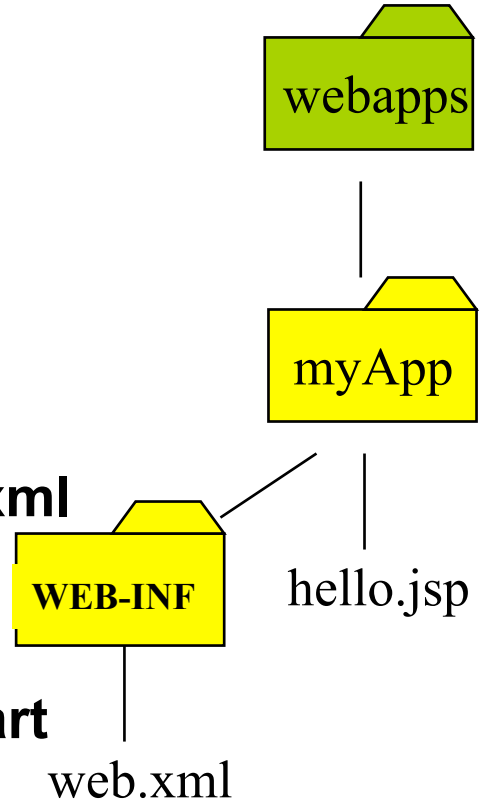
In the myApp folder we can depost the JSP files.

On our Tomcat server, the URL for the hello.jsp file becomes:

http://*machine/port*/myApp/hello.jsp

The WEB-INF directory still contains the same web.xml file as in the static case must be provided.

To actually see the webapp, you might have to restart Tomcat (with older Tomcat versions)



JSP

Tag Extension

http://java.sun.com/products/jsp/tutorial/TagLibrariesTOC.html

JSP custom tag

Ideally, JSP pages should contain no code written in the Java programming language (that is, no expressions or scriptlets). Anything a JSP page needs to do with Java code can be done from a

custom tag

- ***Separation of form and function.***
- ***Separation of developer skill sets and activities.***
- ***Code reusability.***
- ***Clarified system design.***

a JSP custom tag

```
<%@ taglib uri="/hello" prefix="example" %>
<HTML><HEAD><TITLE>First custom tag</TITLE></HEAD>
<BODY>
This is static output
<p />
<i><example:hello>HELLO THERE</example:hello></i>
This is static output
</BODY>
</HTML>
```

hello.doStartTag()

hello.doEndTag()

a JSP custom tag

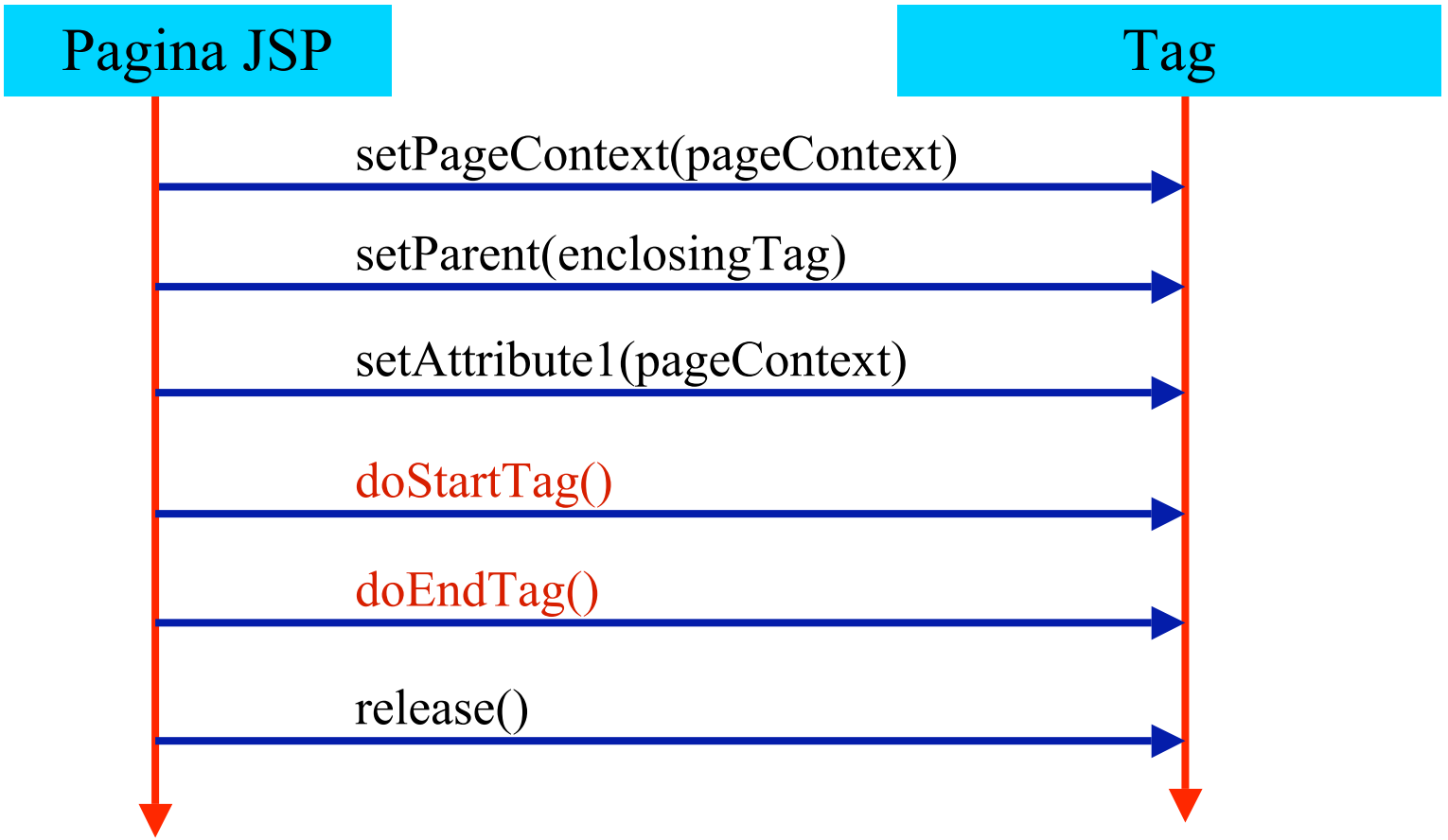
```
package jsptags;
import java.io.IOException;
import java.util.Date;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class HelloTag extends TagSupport {
  public int doStartTag() throws JspTagException {
    try {
      pageContext.getOut().write("Start tag found here<BR>");
    } catch (IOException e) {
      throw new JspTagException("Fatal error: could not write to JSP out");
    }
    return EVAL_BODY_INCLUDE;  // return SKIP_BODY;
  }
```

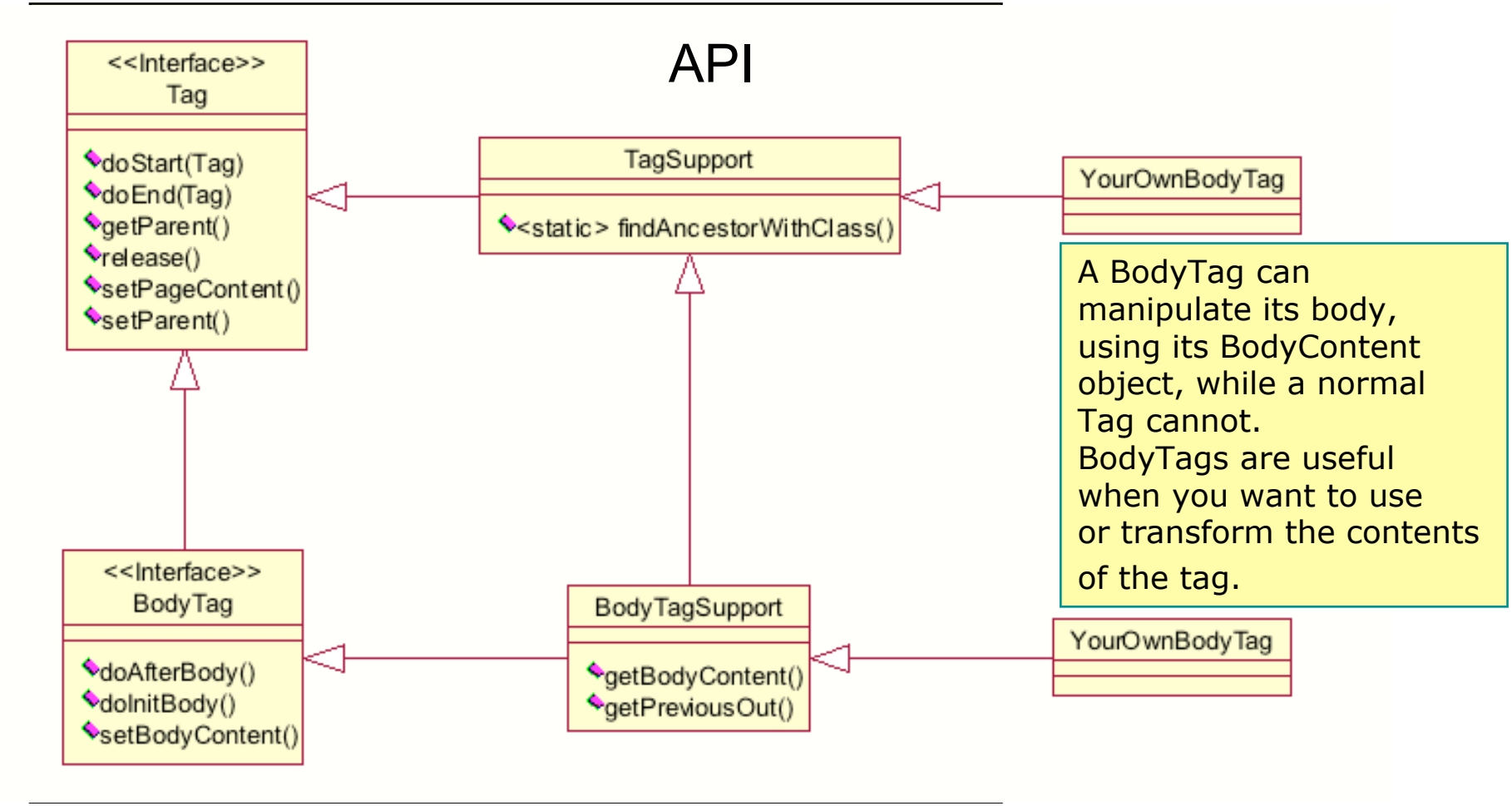
a JSP custom tag

```
…
public class HelloTag extends TagSupport {
…
 public int doEndTag() throws JspTagException {
    try {
      pageContext.getOut().write("End tag found<BR>");
    } catch (IOException e) {
      throw new JspTagException("Fatal error: could not write to JSP out");
    }
    return EVAL_PAGE;      // return SKIP_PAGE;
  }
}
```

Javax.servlet.jsp.tagext.Tag interface



Class Diagram



a JSP custom tag

```
<%@ taglib uri="/hello" prefix="example" %>
<HTML><HEAD><TITLE>First custom tag</TITLE></HEAD>
<BODY>
This is static output
<p />
<i><example:hello>HELLO THERE</example:hello></i>
This is static output
</BODY>
</HTML>
```

hello.doStartTag()

hello.doInitBody()

hello.doAfterBody()

hello.doEndTag()

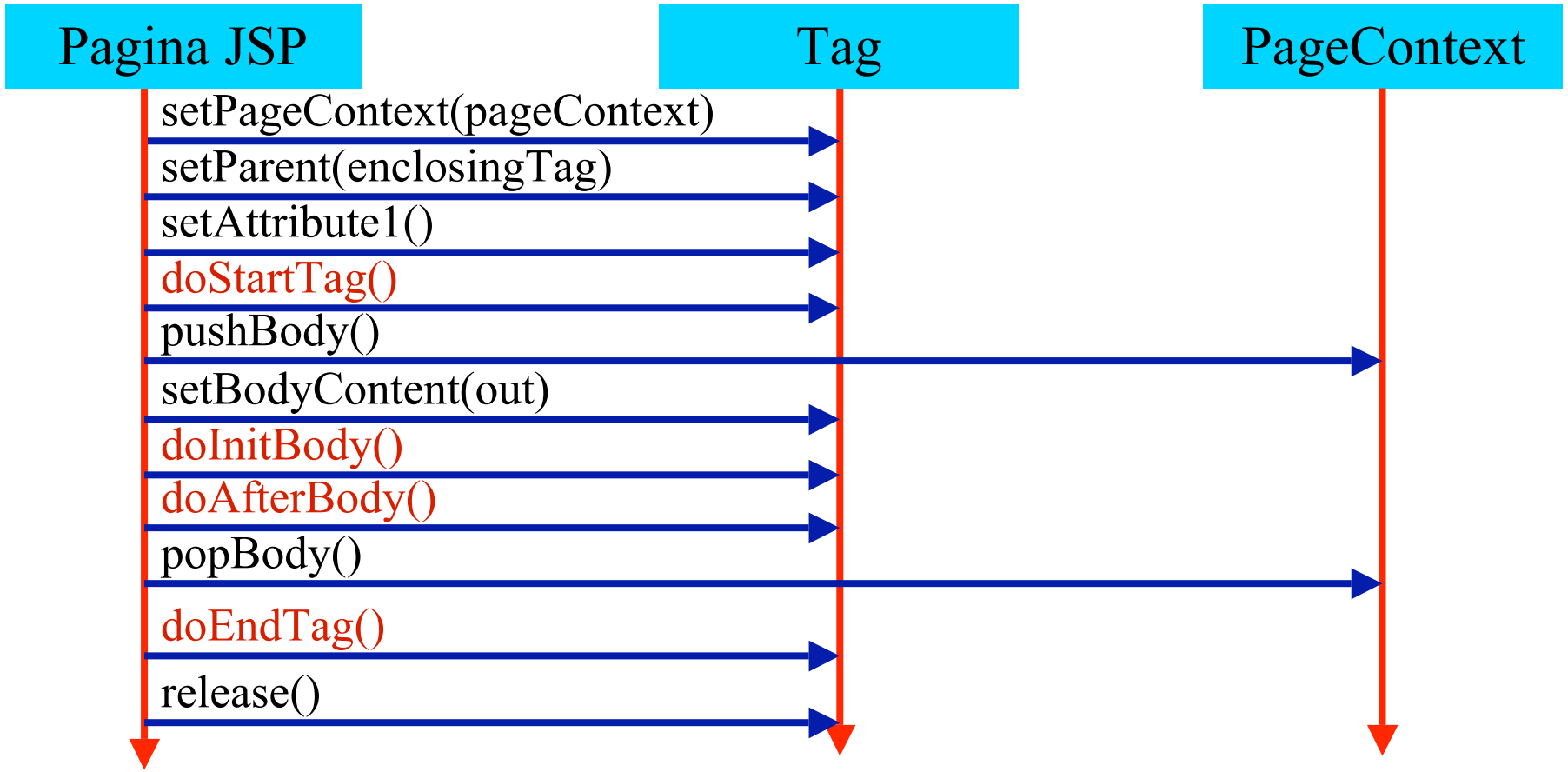
a JSP custom tag

```
package jsptags;
…
public class HelloTag extends BodyTagSupport {
  public int doStartTag() throws JspTagException {
    …
  }
public void doInitBody() throws JspTagException {
    try {
      pageContext.getOut().write("Init Body<BR>");
    } catch (IOException e) {
      throw new JspTagException("Fatal error: could not write to JSP out");
    }
  }
```

a JSP custom tag

```
 public int doAfterBody() throws JspTagException {
   try {
     pageContext.getOut().write("After Body<BR>");
   } catch (IOException e) {
     throw new JspTagException("Fatal error: could not write to JSP out");
   }
   return EVAL_BODY_TAG; // return SKIP_BODY;
 } */
 public int doEndTag() throws JspTagException {
    …
  }
}
```

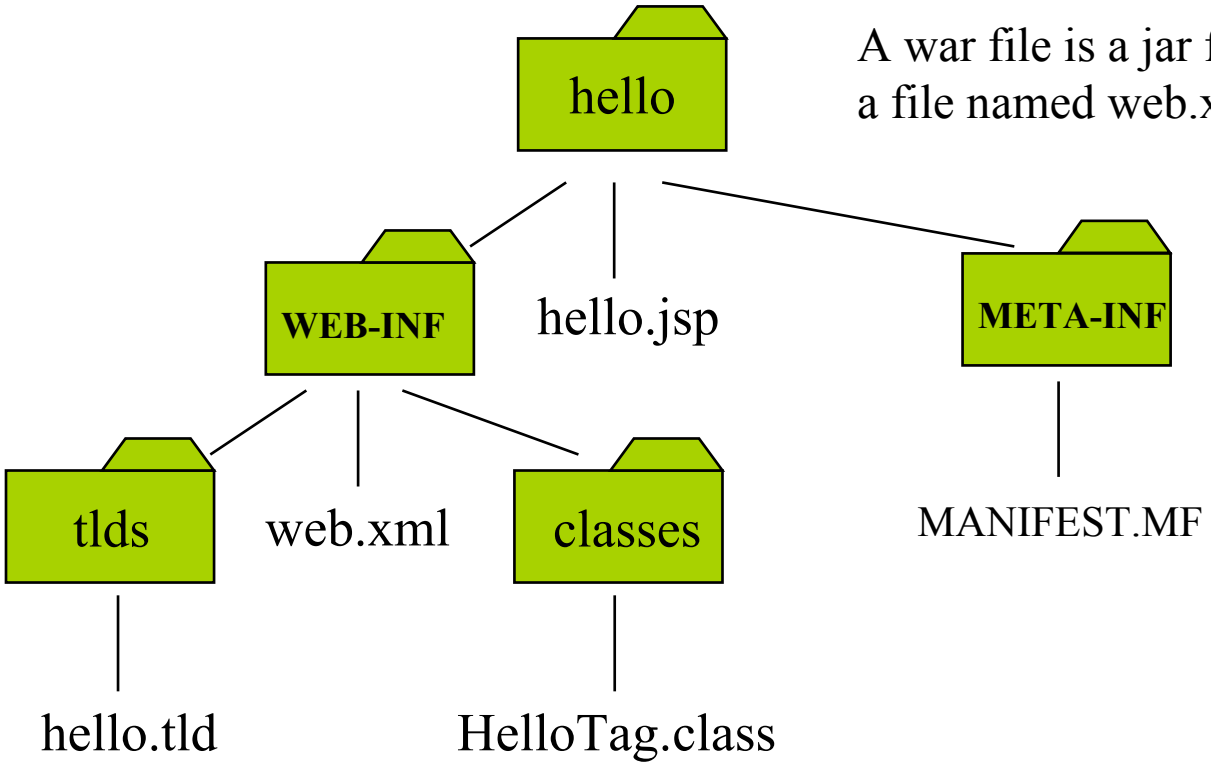
Javax.servlet.jsp.tagext.BodyTag interface



reversing body content

```
import java.io.IOException; import javax.servlet.jsp.*; import javax.servlet.jsp.tagext.*;
public class ReverseTag extends BodyTagSupport {
    public int doEndTag() throws JspTagException {
        BodyContent bodyContent = getBodyContent();
        if (bodyContent != null) {// Do nothing if there was no body content
            StringBuffer output = new StringBuffer(bodyContent.getString());
             output.reverse();
             try {
                bodyContent.getEnclosingWriter().write(output.toString());
            } catch (IOException ex) {
                throw new JspTagException("Fatal IO error");
            }
        }  return EVAL_PAGE;
    }
}
```

structure of the war file



A war file is a jar file with special directories and a file named web.xml in the WEB-INF directory

TLD

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
      PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
      "http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">
<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>examples</shortname>
  <info>Simple example library.</info>
  <tag>
    <name>reverse</name>
    <tagclass>tagext.ReverseTag</tagclass>
    <bodycontent>JSP</bodycontent>
    <info>Simple example</info>
  </tag>
</taglib>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!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>
 <display-name>tagext</display-name>
 <description>Tag extensions examples</description>
 <session-config>
  <session-timeout>0</session-timeout>
 </session-config>

 <taglib>
    <taglib-uri>/hello</taglib-uri>
    <taglib-location>/WEB-INF/tlds/hello.tld</taglib-location>
 </taglib>

</web-app>
```

Public Tag Libraries

See e.g.:

JSTL by Apache

http://jakarta.apache.org/taglibs/doc/standard-doc/intro.html

Open Source JSP Tag Libraries by JavaSource.net

http://java-source.net/open-source/jsp-tag-libraries