Q

How do I configure web.xml ?

more on web.xml

Some of the info expected in the web.xml can be provided via annotation. E.g.

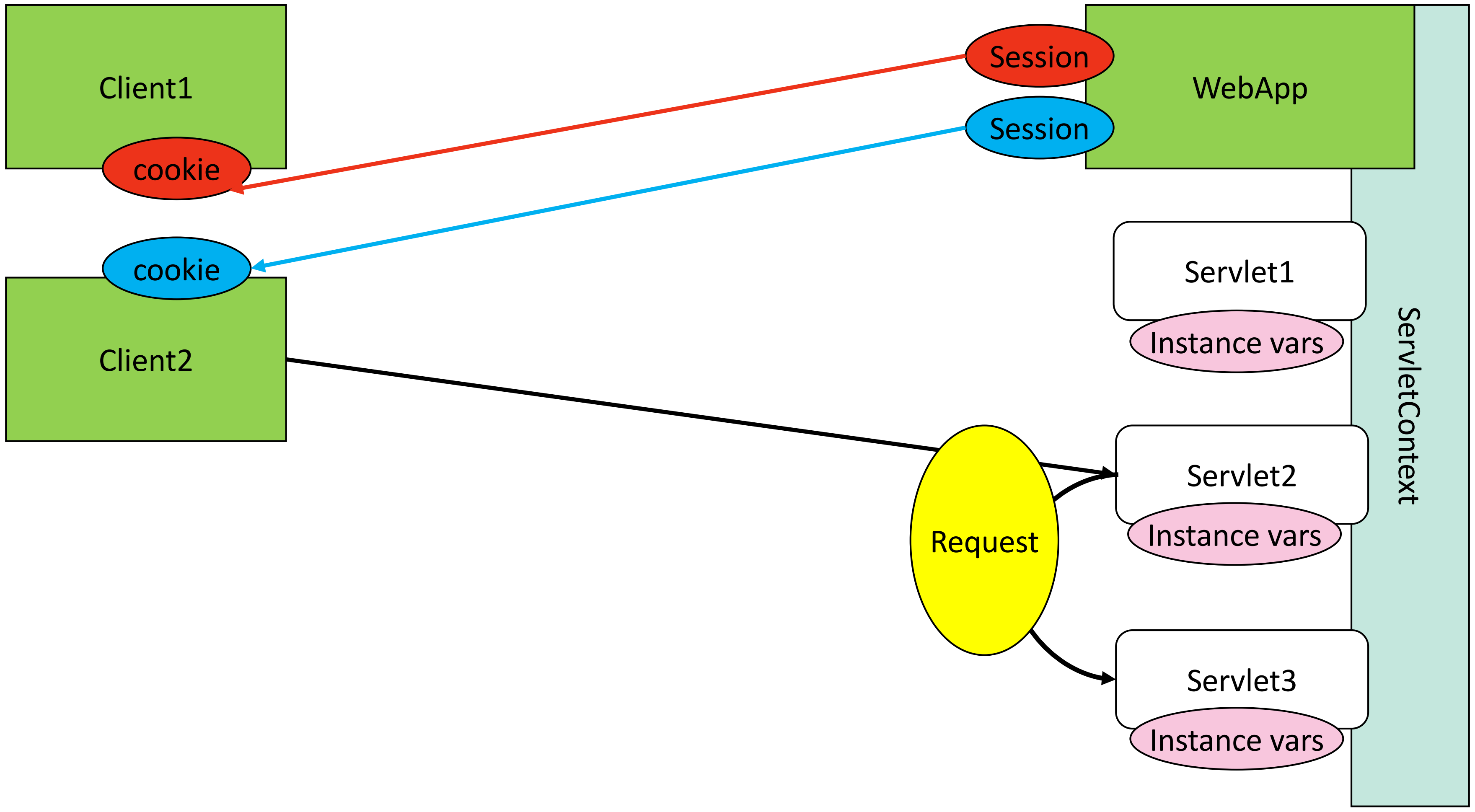
```
package it.unitn.disi.ronchet.myservlets;

@WebServlet(name="myServlet",

        urlPatterns = {"/welcome"})

public class Welcome extends HttpServlet
```

Is equivalent to

```
</web-app>

    <servlet>

        <servlet-name>myServlet</servlet-name>

        <servlet-class>it.unitn.disi.ronchet.myservlets.Welcome

        </servlet-class>
    </servlet>
    <servlet-mapping

         <servlet-name>myServlet</servlet-name>

         <url-pattern>/welcome</url-pattern>

    </servlet-mapping>

</web-app>
```

Q

How can we keep global information in a webApp?


Client1
cookie
cookie
Client2
Session
Session
WebApp
Servlet1
Instance vars
Servlet2
Instance vars
Servlet3
Instance vars
Request
ServletContext

Sharing information

- Within a request:

multiple servlets cooperating through *forward* and *include* mechanisms:

- pass request and response, add information to the **request** (request in jsp)

- Among different requests by the same user:

- use the **Session** object

getSession() in servlets

session in jsp

Sharing information

- Among different invocations of the same servlet:

- use instance variables or static variables

- Among different servlets/jsps of the same WebApp:

- use ServletContext
    - servlets: getServletConfig().getServletContext
    - jsps: application (a special, predefined object)

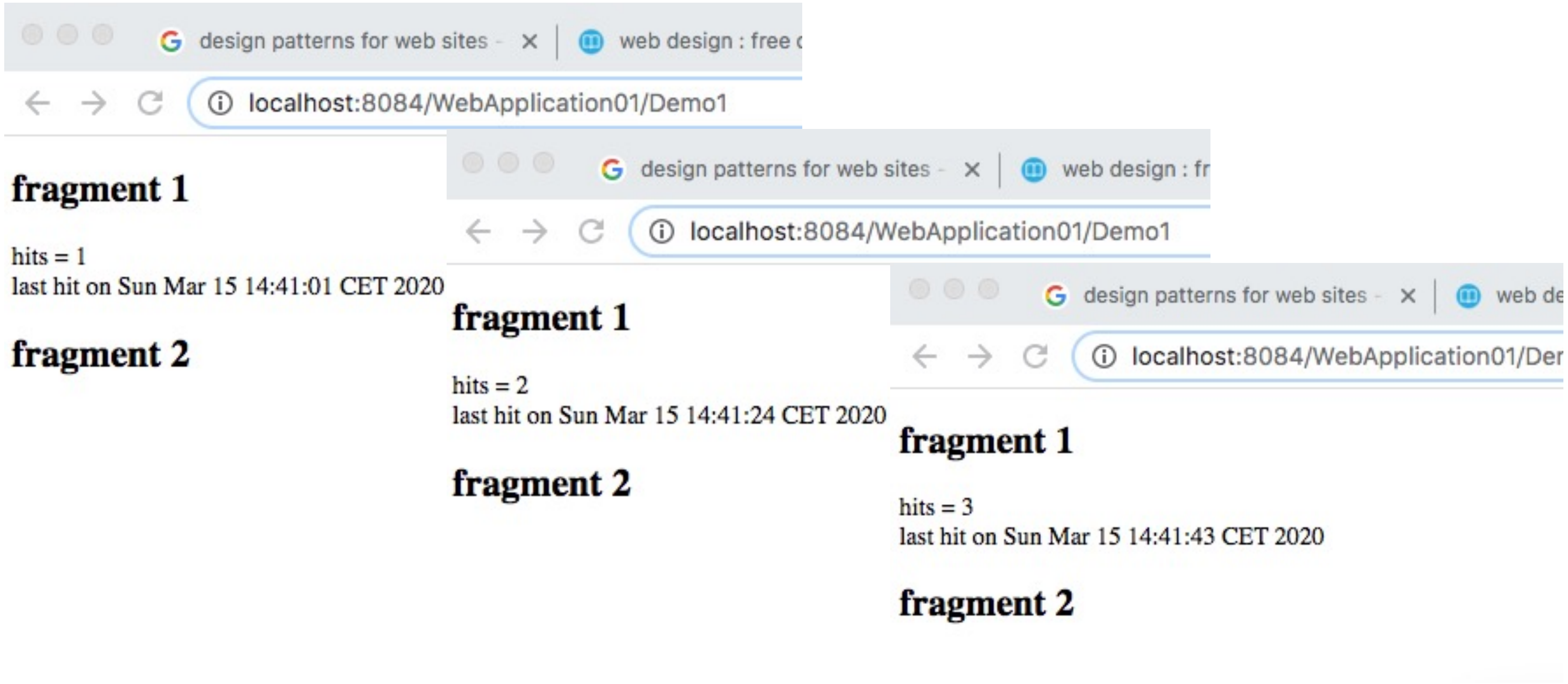
Let us build a hit counter - 1

```
public class Counter {
    int count = 0;
    Calendar timeStamp = Calendar.getInstance();
    public void increase(){
        count++;
        timeStamp = Calendar.getInstance();
    }
    @Override
    public String toString() {
        StringBuffer s = null;
        if (count == 0)
            s = new StringBuffer("<p>no hits yet</p>");
        else {
            s = new StringBuffer("<p>hits = ");
            s.append(count)
                    .append("<br>last hit on ")
                    .append(timeStamp.getTime().toString());
        }
        return s.toString();
    }
}
```

Let us build a hit counter - 2

```
@WebServlet(name = "Demo1", urlPatterns = {"/Demo1"})
public class Demo1 extends HttpServlet {
    Counter counter=new Counter();
    @Override
    protected void doGet(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            request.getRequestDispatcher("/fragment1.html")
                   .include(request, response);
            counter.increase();
            out.println(counter);
            request.getRequestDispatcher("/fragment2.html")
                    .include(request, response);
        }
    }
}
```

Output


design patterns for web sites -
web design : free
localhost:8084/WebApplication01/Demo1

```
fragment 1

hits = 1
last hit on Sun Mar 15 14:41:01 CET 2020

fragment 2
```

design patterns for web sites -
web design : fr
localhost:8084/WebApplication01/Demo1

```
fragment 1

hits = 2
last hit on Sun Mar 15 14:41:24 CET 2020

fragment 2
```

design patterns for web sites -
web de
localhost:8084/WebApplication01/Der

```
fragment 1

hits = 3
last hit on Sun Mar 15 14:41:43 CET 2020

fragment 2
```

Even if anyone accesses from a different client!
But if we restart the server, counter restarts from 1!

Q

How can we persist (global) information in a webApp?

How can we persist the counter?

In a file (named counterData)!

1) In init, let us check if file exists. If yes, let us resume the counter, else, let us create a new one.
2) In destroy, let us save counter in counterData.

Java serialization

```
class A implements Serializable {…}
```

```
A a1=new A();
A a2;
 …
File myFile = new File(filePath);
…
ObjectOutputStream oi = new ObjectOutputStream(new
          FileOutputStream(myFile));
oi.writeObject(a1);
…
ObjectInputStream oi = new ObjectInputStream(new
          FileInputStream(myFile));
a2 = (A) oi.readObject();
```

(throws various exceptions…

(throws various exceptions…

Let us build a hit counter - 1

```
@Override
public void destroy() {
    File f=new File("/Users/ronchet/…");
    try (ObjectOutputStream oi =
            new ObjectOutputStream(
            new FileOutputStream(f))) {
        oi.writeObject(counter);
    } catch (IOException e) {
        e.printStackTrace();
    }}
```

```
public class Counter implements Serializable {
    … // same as before
}
```

In Servlet:

```
Counter counter;
@Override
public void init() {
    File f=new File("/Users/ronchet/Download/counterData"); //update with your path!
    try {
        if (! f.exists()) {
            f.createNewFile();
            counter=new Counter();
    } else {
            ObjectInputStream oi = new ObjectInputStream(new FileInputStream(f));
            counter=(Counter) oi.readObject();
    }
    } catch (IOException | ClassNotFoundException e) { // bad exception catching
        e.printStackTrace();
    }
}
```

!

Wait – our counter solution is not thread-safe!

when sharing info, always think about thread-safety!

Servlets are not thread safe!

unless YOU make them so…

- A *thread* is a lightweight process which has its own call stack and accesses shared data of other threads in the same process (shares heap memory).
- A servlet can be invoked simultaneously by multiple threads (i.e., by multiple requests).
- We can fix this problem dealing with concurrency.

- *You should know about concurrency, threads, semaphores and monitors from your bachelor courses. If you do not, see here:*

  https://docs.oracle.com/javase/tutorial/essential/concurrency/

  (The basics that you need are in the "Concurrency" section)

5 rules to remember

1. Service() , doGet(), doPost() or to be more generic doXXX() methods should not update or modify instance variables as instance variables are shared by all threads of same instance.
2. If you have a requirement which requires modification of instance variable then do it in a synchronized block. (or synchronized method)
3. Above two rules are applicable for static variables also because they are also shared.
4. Local variables are always thread safe (unless they refer to global objects)
5. The request and response objects are thread safe to use because new instance of these are created for every request into your servlet, and thus for every thread executing in your servlet.

Q

Are session objects thread safe?

Sessions and thread safety

- A *session* belongs to a user.
- Hence, when different users activates the same servlet, and this requests a session object, it gets a different object for every user – so no problems with multithreading.

BUT

- if a user opens two windows on the same browser, and accesses the same servlet, then we DO have a thread safety issue!
- that's very unlikely, but yet…

Q

How can we fix our counter making it thread-safe?

Fixing the hit counter - option 1

```
public class Counter {
    int count = 0;
    Calendar timeStamp = Calendar.getInstance();
    public synchronized void increase(){
        count++;
        timeStamp = Calendar.getInstance();
    }
    @Override
    public String toString() {
        StringBuffer s = null;
        if (count == 0)
            s = new StringBuffer("<p>no hits yet</p>");
        else {
            s = new StringBuffer("<p>hits = ");
            s.append(count)
                    .append("<br>last hit on ")
                    .append(timeStamp.getTime().toString());
        }
        return s.toString();
    }
}
```

Fixing the hit counter - option 2

```
@WebServlet(name = "Demo1", urlPatterns = {"/Demo1"})
public class Demo1 extends HttpServlet {
    Counter counter=new Counter();
    @Override
    protected void doGet(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            request.getRequestDispatcher("/fragment1.html")
                   .include(request, response);
            synchronized (this) {
                counter.increase();
            }
            out.println(counter);
            request.getRequestDispatcher("/fragment2.html")
                   .include(request, response);
        }
    }
}
```

Q

What are JSPs?

How are they related to servlets?

JSP Technology

A technology somehow similar to PHP or ASP, ASP.net, but Java-based.

Dual to Servlets

Has been the basis for JSP-CustomTags

Has been the basis for JSF

Tutorial
https://www.tutorialspoint.com/jsp/index.htm

Simple.jsp

```
<%@ page import=java.util.*  %>
<html>
 <body>
    <% int x=Calendar.get(Calendar.HOUR_OF_DAY); %>
    <%= x %>
  </body>
</html>
```

A taste of servlet programming-2

```
import java.util.Calendar;
public class SimpleServlet extends HttpServlet {
    public void doGet (HttpServletRequest request,
            HttpServletResponse response)
                        throws ServletException, IOException {
            PrintWriter out=response.getWriter();
            response.setContentType("text/html");
            out.println("<HTML><BODY>");
            x=Calendar.get(Calendar.HOUR_OF_DAY);
            out.println(x);
            out.println("</BODY></HTML>");
            out.close();
    }
}
```

<%@ directives %>

<% scriptlets %>

<%= expressions %>

Equivalent to:
out.println(expression);

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.

A taste of servlet programming-2

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)

<%@ directives %>

```
import java.util.Calendar;
public class SimpleServlet extends HttpServlet {
   String nome="pippo"; //instance variable
   final float PI=3.1415926535 // constant
   public void getName() {/* this is my function */}
   public void doGet (HttpServletRequest request,
            …
   }
}
```

<%! 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.

Examples:

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

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

Directives

<%@ DIRECTIVE{attributo=valore} %>

main attributes:

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

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

<%@ page errorPage=URL %>

<%@ page isErrorPage=true %>

JSP nuts and bolts

Syntactic elements:

<%@ directives %> → Interaction with the CONTAINER

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

<% scriptlets %> → In the service method

<%= expression %> → (Syntactic sugar)
same as scriptlet: <% out.println(expression %)>

<jsp:actions/>

JSP Standard actions

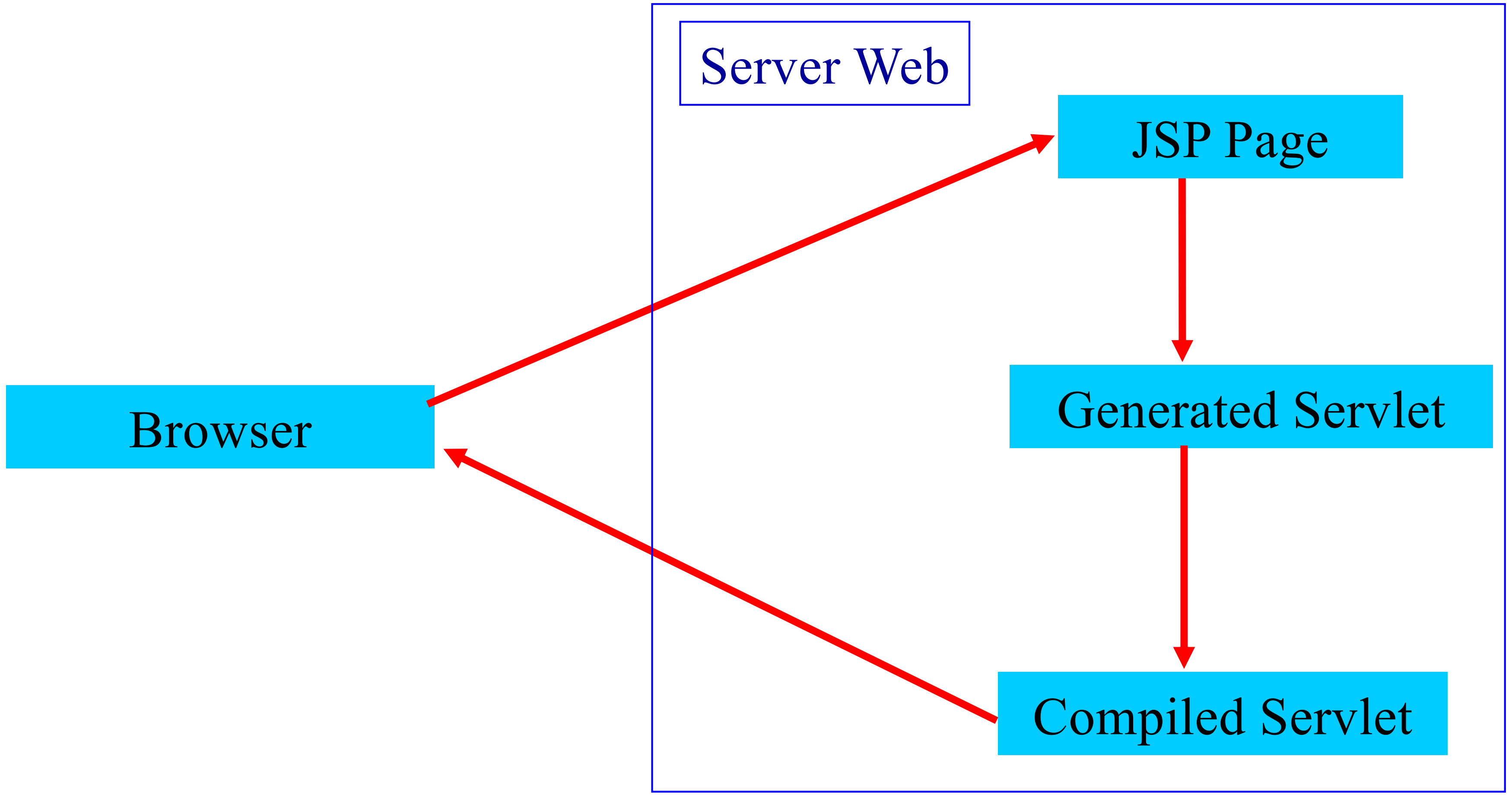
```
<jsp:include page=“URL” />
```

For including STATIC or DYNAMIC resources at request time

```
<jsp:forward page=“URL” />
```

```
<jsp:useBean id= "instanceName"
  scope= "page | request | session | application"
  class= "packageName.className" type= "packageName.className"
  beanName="packageName.className | <%= expression >" >
</jsp:useBean>
```

JSP Lifecycle



Q

How should I configure Tomcat to use JSPs?

JSP pages

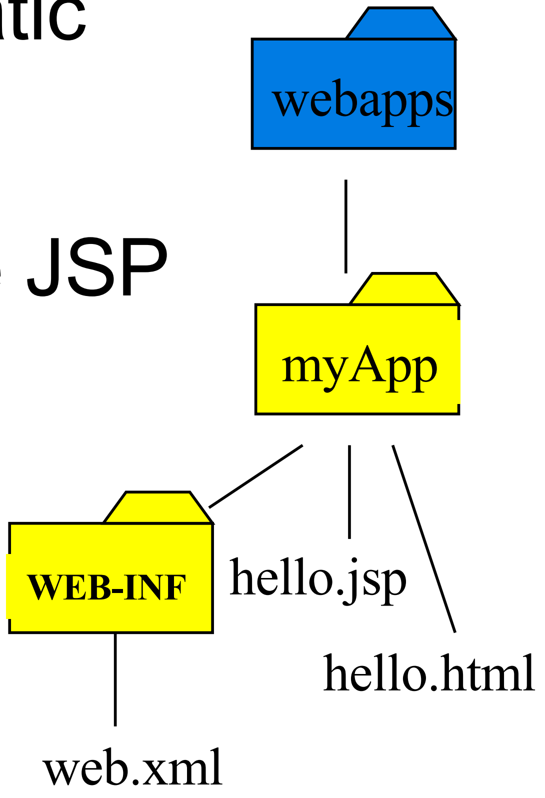
To let Tomcat serve JSP pages, we follow the same procedure that we use for static pages.

In the myApp folder we can deposit 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 can be empty.



Q

How can I access request and response in JSPs?

How can I use sessions with JSPs?

Predefined Objects

out	Writer
request	HttpServletRequest
response	HttpServletResponse
session	HttpSession
page	this in the Servlet
application	servlet.getServletContext area shared among all servlets within the same webapp
config	ServletConfig
exception	only in a errorPage
pageContext	

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>
```

JSP in action – Example part 1

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="java.util.Date"%>
<%@page language="java"  session="true" %>

<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type"
             content="text/html; charset=UTF-8">
        <title>Session Test JSP</title>
    </head>
    <body>
      <%!  Integer accessCount; %>
      <%
        accessCount=(Integer)session.getAttribute("accessCount");
         if (accessCount == null) {
            accessCount = 0;   // autobox int to Integer
         } else {
            accessCount = new Integer(accessCount + 1);
         }
         session.setAttribute("accessCount", accessCount);
      %>
```

JSP in action – Example part 2

```
Session is new? <% out.println(session.isNew()); %>
      <h2>You accessed this site " <%= accessCount %>
            times in this session.</h2>
      <ul><li>Your session ID is " <%= session.getId() %></li>
          <li>Session creation time is
               <%= new Date(session.getCreationTime()) %> </li>
          <li>Session last access time is  <%=
               new Date(session.getLastAccessedTime()) %> </li>
          <li>Session max inactive interval  is <%=
               session.getMaxInactiveInterval() %> seconds</li>
       </ul>

       <p><a  href='<%= request.getRequestURI() %>'>Refresh</a>
       <p><a  href='
            <%= response.encodeURL(request.getRequestURI())%>'>
               Refresh with  URL rewriting</a>
       <form method="GET" action="endSession.jsp">
           <input type="submit" value="End Session">
       </form>
    </body>
</html>
```

Q

How can we make good use of JSPs?

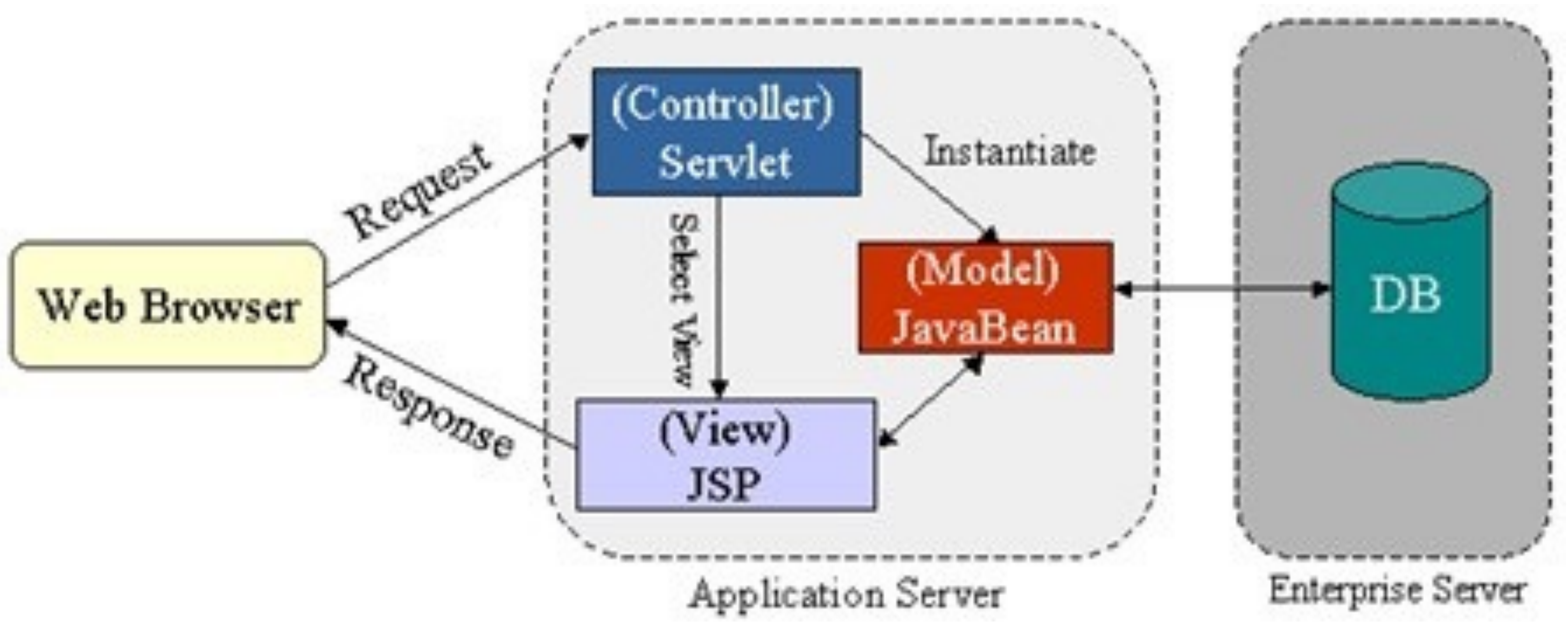
Best practices

- **Don't overuse Java code in HTML pages**
- **Choose the right include mechanism:**
    - *Static data* such as headers, footers, and navigation bar content is best kept in separate files and not regenerated dynamically.
    - Once such content is in separate files, they can be included in all pages using one of the following include mechanisms:
    - Include directive: <%@ include file="filename" %>

      Include action: <jsp:include page="page.jsp" />
- **Don't mix business logic with presentation**
    - JSP code should be limited to front-end presentation.
- **Use filters if necessary** (See next lecture)
- **Use a database for persistent information** (See next lectures)
    - Use connection pooling

JSP usage: MVC pattern

MVC



Web Browser
Request
Response
(Controller) Servlet
Instantiate
Select View
(Model) JavaBean
(View) JSP
DB
Application Server
Enterprise Server

What is a Javabean?

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)
- Is thread safe/security conscious
  - Can run in an applet, application, servlet, ...

Example:

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

Standard actions involving beans

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

```
<jsp:setProperty name="nome" property="value" />
<jsp:getProperty name="nome" property="value" />
```

See: https://www.tutorialspoint.com/jsp/jsp_java_beans.htm

Example - BeanOne.java

define the bean

```
package beans;
import java.io.Serializable;
public class BeanOne implements Serializable {
    String name;
    String surname;
    public BeanOne() {}
    public String getName() { return name; }
    public String getSurname() {return  surname;}
    public void setName(String name) { this.name =  name;}
    public void setSurname(String surname) {this.surname = surname;}
    @Override
    public String toString() {
        return "BeanOne{" + "name=" + name + ", surname=" + surname +  '}';
    }
}
```

Example - index.html

main page

```
<!DOCTYPE html>
<html>
<head>
   <title>JSP - Hello World</title>
</head>
<body>
<br/>
<a href="jspOne.jsp">jspOne</a><br>
<a href="jspTwo.jsp">jspTwo</a><br>
<a href="addZ">Add Z</a><br>
<a href="InvalidateServlet">invalidate session</a>
</body>
</html>
```

Example - jspOne.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
     <title>JSP Page</title>
  </head>
  <body>
    <h1>Setting the property...</h1>
     <jsp:useBean id="myBean1" class="beans.BeanOne" scope="session"/>
     <jsp:setProperty name="myBean1" property="surname" value="de pippis"/>
     <jsp:setProperty name="myBean1" property="name" value="pippo"/>
      <p><%=myBean1.toString()%></p>
      <hr>
      <jsp:include page="index.html"></jsp:include>
</html>
```

set value
in the bean

What happens If
we change the scope?

Example - jspTwo.jsp

show value
in the bean

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Getting the property...</h1>
        <jsp:useBean id="myBean1" class="beans.BeanOne" scope="session"/>
        <jsp:getProperty name="myBean1" property="surname" />
        <jsp:getProperty name="myBean1" property="name" />
        <p><%=myBean1.toString()%></p>
    <hr>
    <jsp:include page="index.html"></jsp:include>
</body>
</html>
```

Example - BeanAccessServlet.java

modify value
in the bean

```
@WebServlet(name = "addZServlet", value = "/addZ")
public class HelloServlet extends HttpServlet {
  protected void doGet(HttpServletRequest request,  HttpServletResponse response)
              throws ServletException, IOException  {
      HttpSession session=request.getSession();
      if (session.getAttribute("myBean1")==null) {
         session.setAttribute("myBean1", new BeanOne());
      }
      BeanOne aBean=(BeanOne)(session.getAttribute("myBean1"));
      aBean.setName(aBean.getName()+"z");
      request.getRequestDispatcher("jspTwo.jsp").forward(request, response);
  }
}
```

InvalidateSessionServlet.java

invalidate session

```
@WebServlet(name = "InvalidateServlet", value = "/InvalidateServlet")
public class InvalidateServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
                response) throws ServletException,IOException {
        HttpSession session=request.getSession();
        session.invalidate();
        try {
            request.getRequestDispatcher("jspTwo.jsp").forward(request, response);
        } catch (ServletException | IOException e) {
            e.printStackTrace();
        }
    }
}
```

Pay attention to the scope!

JSP

Servlet

```
<jsp:useBean id="myBean1"
class="beans.BeanOne"
scope="session"/>
```

```
Session session=request.getSession();
BeanOne x= (BeanOne )session.getAttribute("myBean1");
```

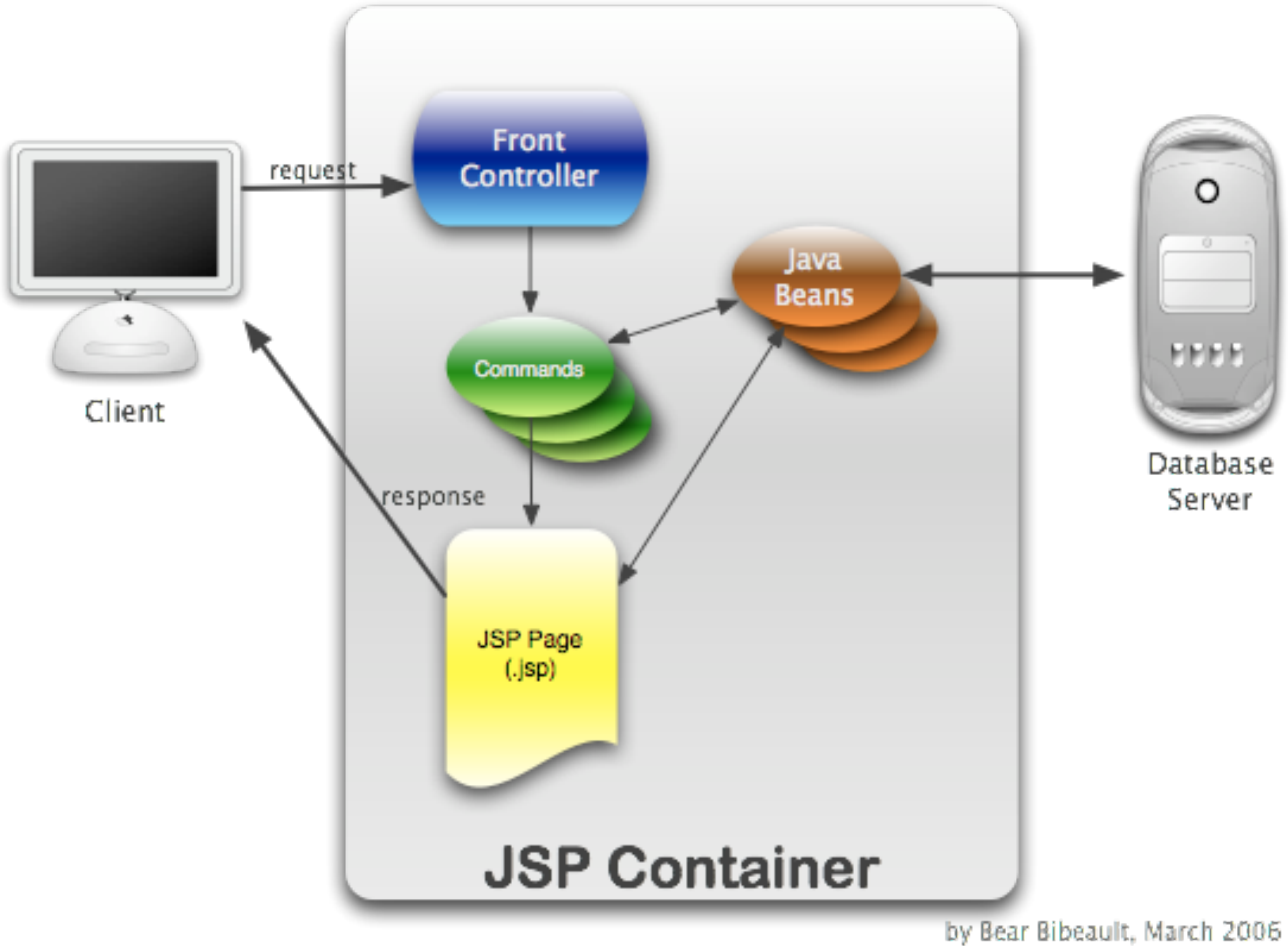
```
<jsp:useBean id="myBean1"
class="beans.BeanOne"
scope="application"/>
```

```
ServletContext context=request.getServletContext();
BeanOne x= (BeanOne )context.getAttribute("myBean1");
```

```
<jsp:useBean id="myBean1"
class="beans.BeanOne"
scope="request"/>
```

```
BeanOne x= (BeanOne )request.getAttribute("myBean1");
```

Front controller pattern



Examples

from:
https://javaranch.com/journal/200603/frontman.html

The Front Controller employed by the Struts package typically uses a servlet mapping of ***.do** where whatever appears as the prefix of the ".do" is used to lookup the actual class path of the Command (called "actions" in Struts) in an internal configuration map.

Another example, the pattern that I usually use, is to employ
a servlet mapping such as **/command/***
where the prefix "command" triggers the front controller,
and the rest of the path info is used to lookup the Command class I
n an internal map.
It would be typical to see URLs along the lines of:

http://some.server.com/webapp/command/deleteItem
http://some.server.com/webapp/command/insertItem
http://some.server.com/webapp/command/doSomethingWonderful