Java Servlet Pages Technology

Most of this material is taken from Prof. M. Ronchetti' s slides

JSP Technology

- A technology somehow similar to PHP, but Java based: a server-side scripting Java-like langage HTML-embedded
- Dual to Servlets
- Has been the basis for JSP-CustomTags
- Classes and interfaces specific to JSP in the packages:
  - **`javax.servlet.jsp`**
  - **`javax.servlet.jsp.tagext`**

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

From Simple.jsp…

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

…to SimpleServlet

```
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();
   }
}
```

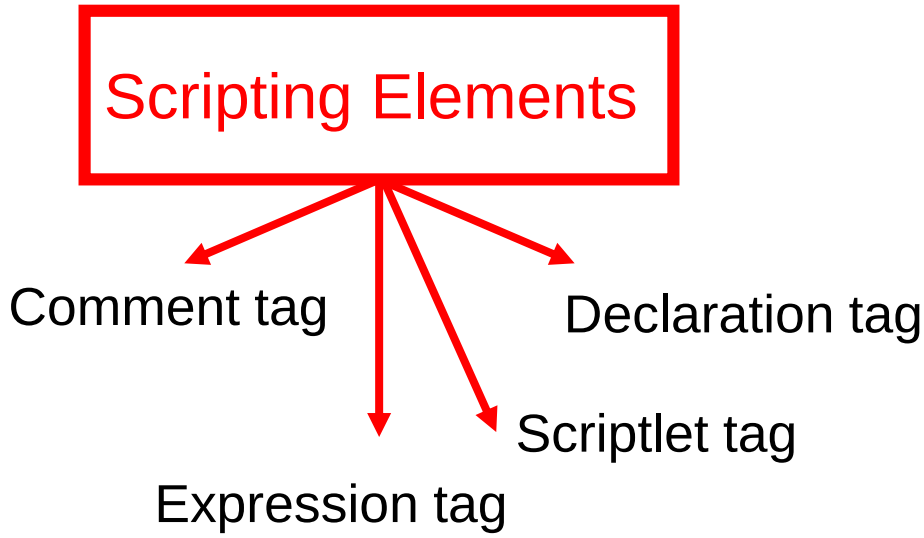
<%@ directives %>

<% scriptlets %>

<%= expressions %>

JSP Elements

- JSP elements help the developers to write the Java code within the tags
- They include:



Directives

<%@ DIRECTIVE TYPE attribute="value" %>

- Directives are used to provide messages to the container about how to manage the client requests while converting a JSP page into a servlet

- Three kinds of directives:
  - page
  - Include
  - taglib

Directives: page

- A page directive defines global information about the whole JSP page that contains it
- The settings it changes direcly affect the page's compilation
- Main attributes:

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

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

<%@ page errorPage=URL %>

<%@ page isErrorPage=true %>

Directives: include

- Include is used to include one file in a JSP page at the translation step. Generally files are static pages such as HTML, text, other files
- Taglib declares that the JSP page uses personalized tags and reports the URL of where they are defined

Scripting elements

Comment tag: <% -- JSP Comment -- %>

Declaration tag: <%! Declaration; [declaration;] +…%>

Expression tag: <%= expression %>

Scriptlet tag: <% code snipplet %>

JSP Standard actions

<jsp:action_name attribute=" attribute_value" >

<jsp:include page="URL" />

For including STATIC or DYNAMIC resources at request time

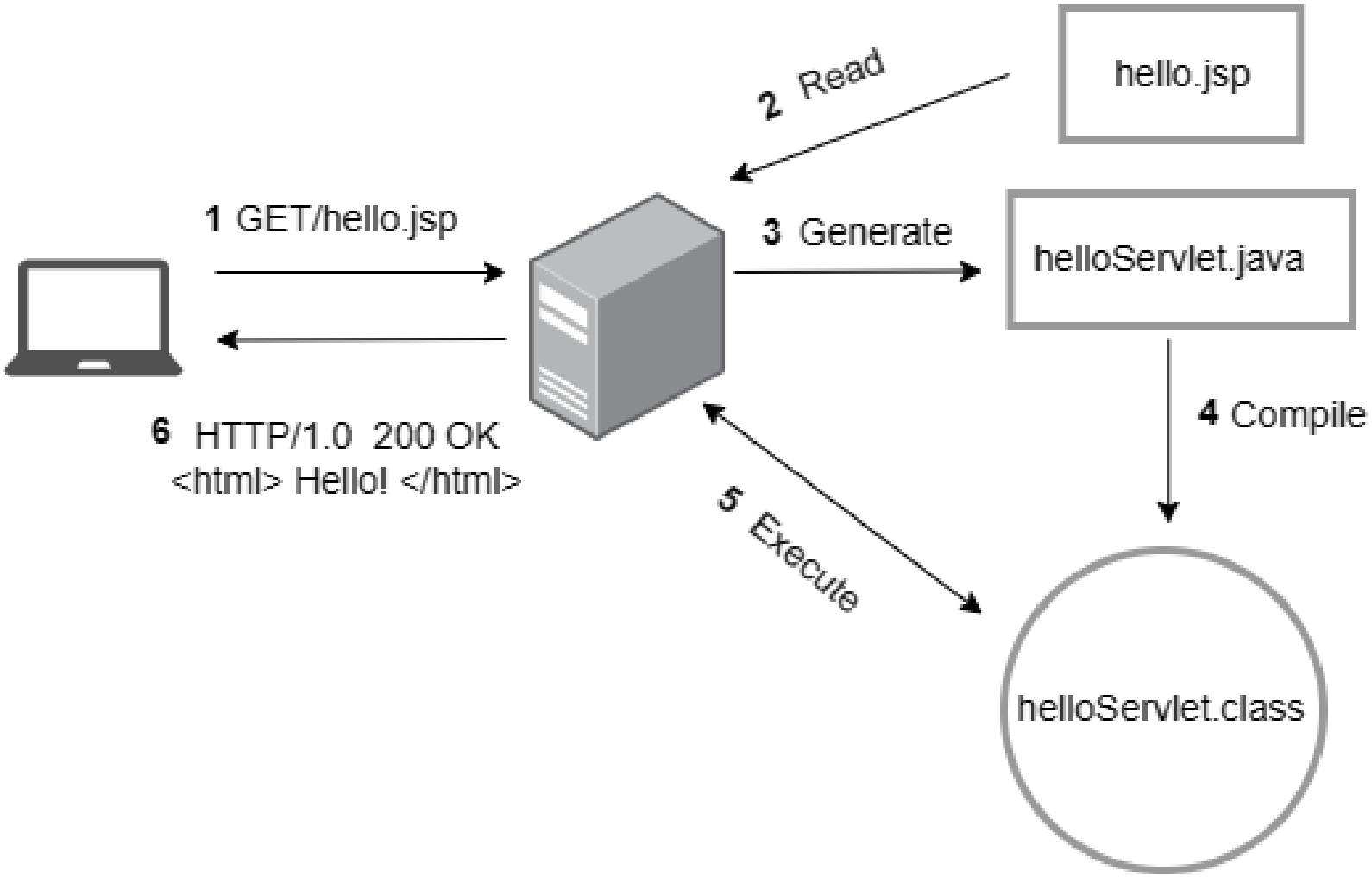
<jsp:forward page="URL" />

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

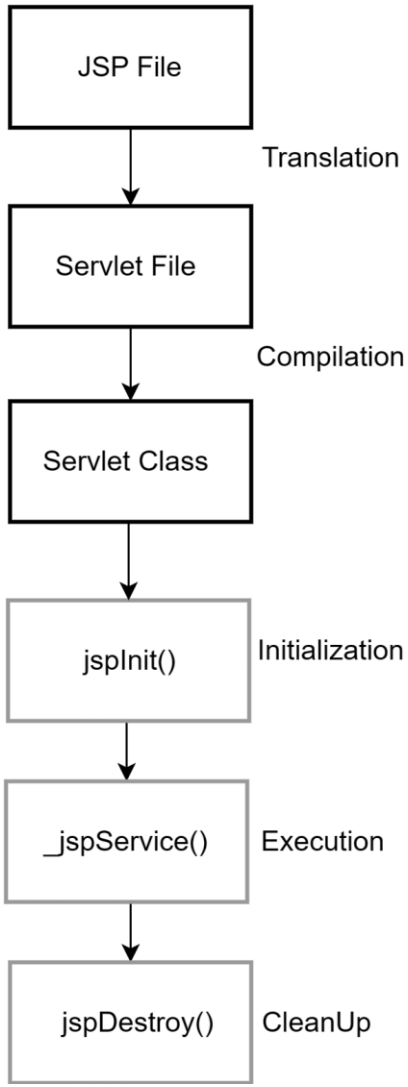
Our date and time

A quick look at date_and_time_jsp

JSP processing



JSP Lifecycle



Where is the generated code?

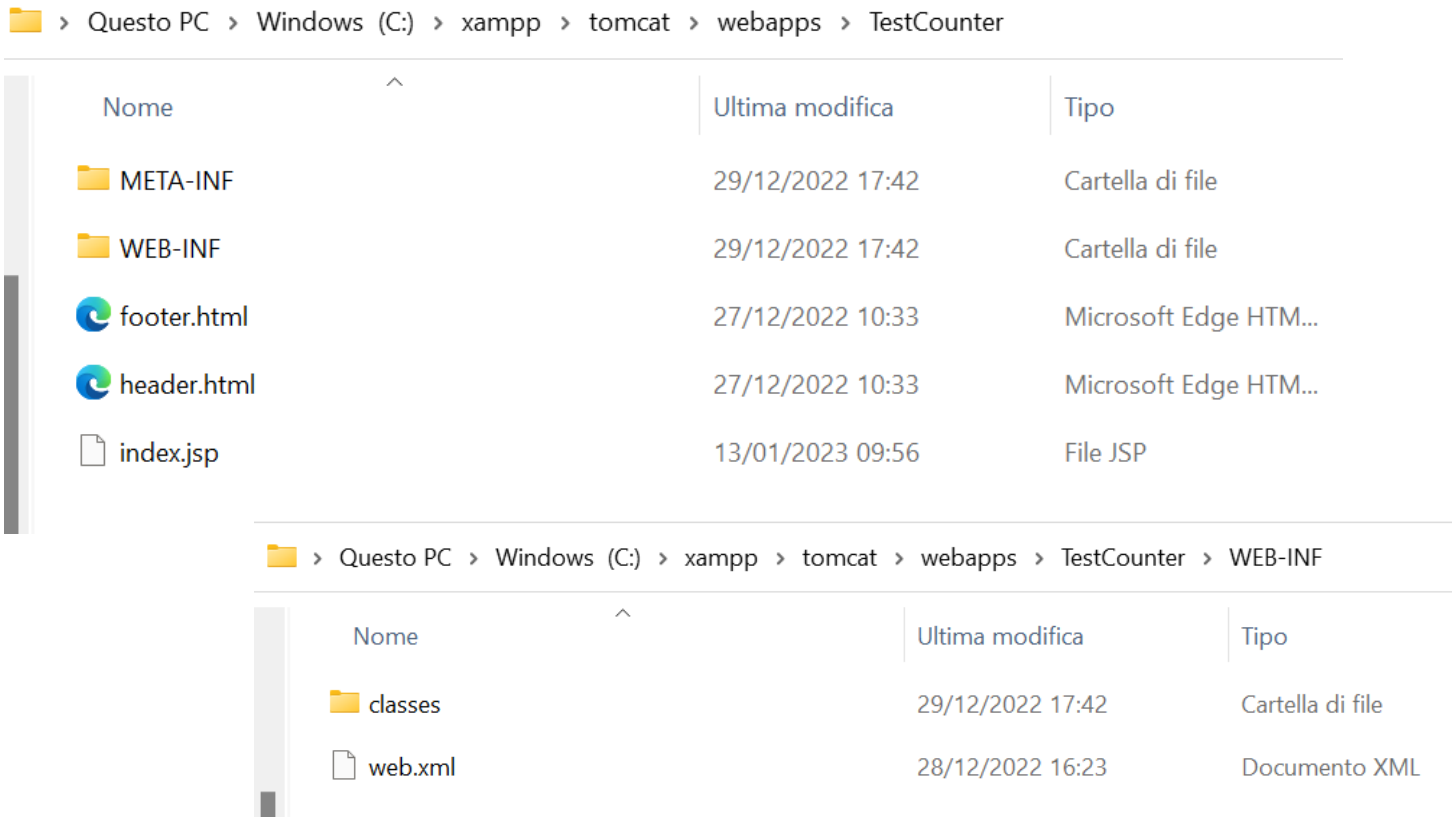
- Some IDE's allow you to see the Servlet generated from the JSP (e.g. Netbeans)
- Unfortunately, IntelliJ does not.

https://intellij-support.jetbrains.com/hc/en-us/community/posts/360003783520-Debugging-JSP-in-Intellij-2019-Ultimate

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



Q

How can I access request and response in JSPs?

How can I use sessions with JSPs?

Implicit Objects

	Represents
out	Writer
request	HttpServletRequest
response	HttpServletResponse
session	HttpSession
page	this in the Servlet
application	servlet.getServletContext
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>
```

A jsp-based calculator

A quick look at a jsp-based calculator

Q

How can we make good use of JSPs?

Best practices

1. **Don't overuse Java code in HTML pages**
2. **Choose the right include mechanism**:
   - *Static data* such as headers, footers, is best kept in separate files and not regenerated dynamically.
   - Once such content is in separate files, they can be included using one of the following include mechanisms:

```
<%@include file = "filename" %>
<jsp:include page = "page.jsp" />
```

3. **Don't mix business logic with presentation**
   - JSP code should be limited to front-end presentation.
4. **Use filters if necessary** (See next lectures)
5. **Use a database for persistent information** (See next lectures)

What is a Javabean?

- A bean is a Java class that:
  - Provides a public no-argument constructor
  - Implements java.io.Serializable
  - Has set/get methods for properties
  - Is thread safe/security conscious

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

- There exist 3 standard actions used for referencing JavaBeans:

```
<jsp:useBean id = "id" class = "bean's class" scope = "bean's scope">

<jsp:setProperty name = "bean's id" property = "property name"
value = "value"/>

 <jsp:getProperty name = "bean's id" property = "property name"/>
```

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

Example BeanOne.java


```
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 +  '}';
    }
}
```

index.html

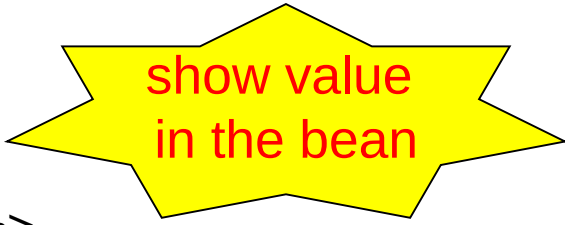

```
<!DOCTYPE html>
<html>
<head>
    <title>JSP - Hello World</title>
</head>
<body>
<br/>
<a href="jspOne.jsp">jspOne</a><br>
<a href="jspTwo.jsp">jspTwo</a><br>
<a href="addZ">Add Z</a><br>
<a href="InvalidateServlet">invalidate session</a>
</body>
</html>
```

jspOne.jsp

set 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>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>
    <a href="index.html"> Home </a></body>
</html>
```

jspTwo.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>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>
    <a href="index.html"> Home </a></body>
</html>
```

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

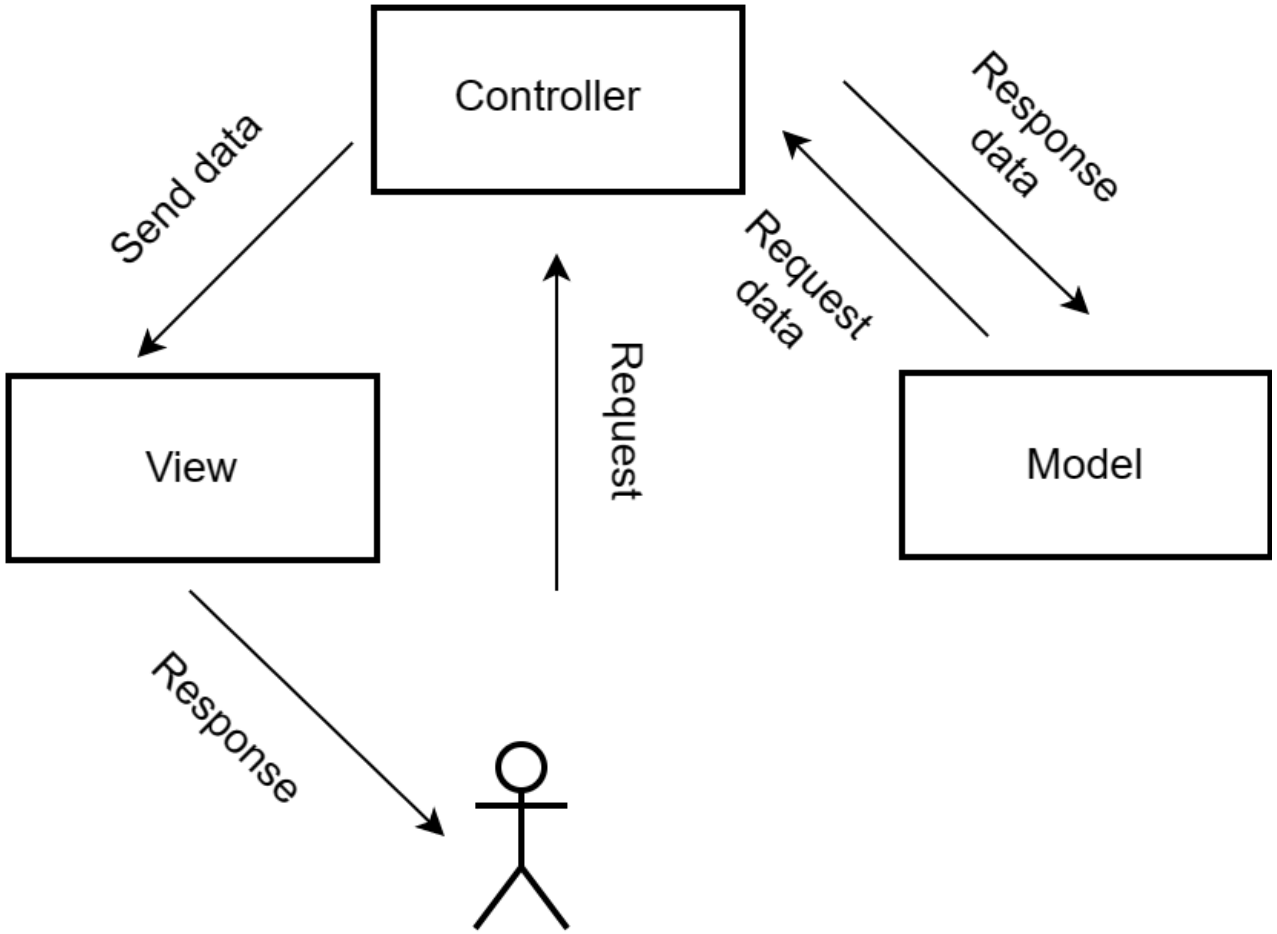

```
@WebServlet(name = "InvalidateServlet", value = "/InvalidateServlet")
public class InvalidateServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse  response) throws
ServletException,IOException {
        HttpSession session=request.getSession(false);
        if (session != null)
            session.invalidate();
}
```

Pay attention to the scope!

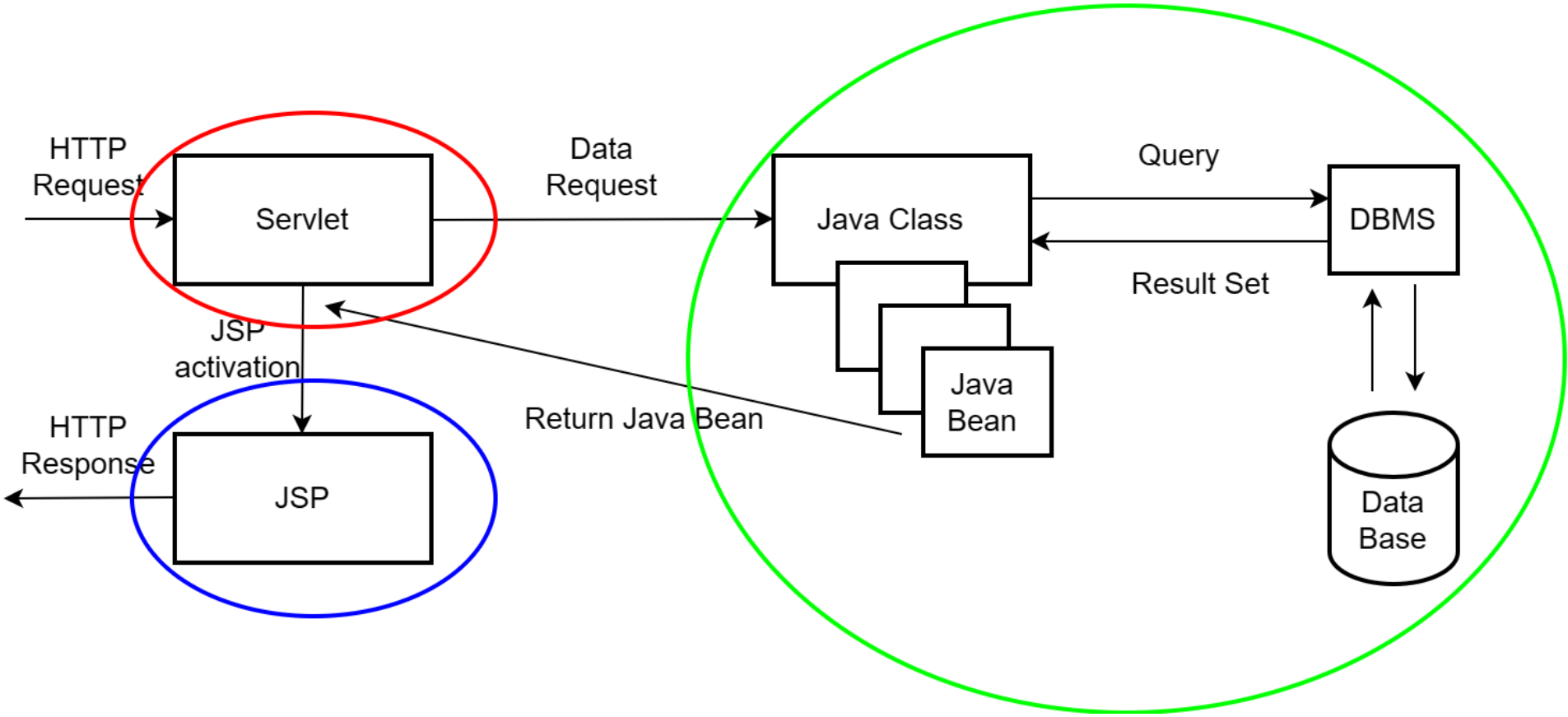
JSP	Servlet
`<jsp:useBean id="myBean1" class="beans.BeanOne" scope="session"/>`	`Session session=request.getSession();` `BeanOne x= (BeanOne )session.getAttribute("myBean1");`
`<jsp:useBean id="myBean1" class="beans.BeanOne" scope="application"/>`	`ServletContext context=request.getServletContext();` `BeanOne x= (BeanOne )context.getAttribute("myBean1");`
`<jsp:useBean id="myBean1" class="beans.BeanOne" scope="request"/>`	`BeanOne x= (BeanOne )request.getAttribute("myBean1");`

JSP usage: MVC pattern

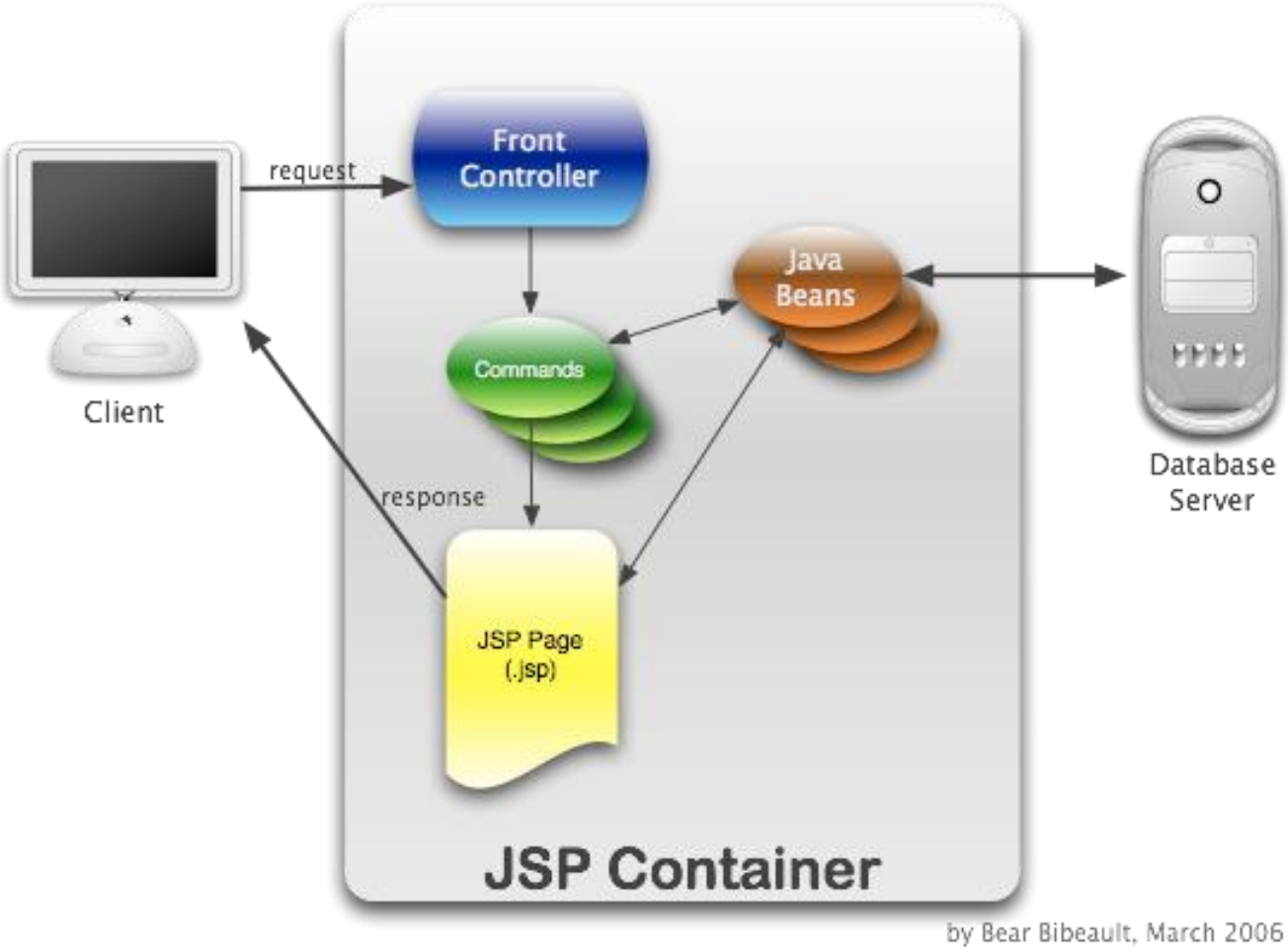
Model View Controller



Model View Controller



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 In 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