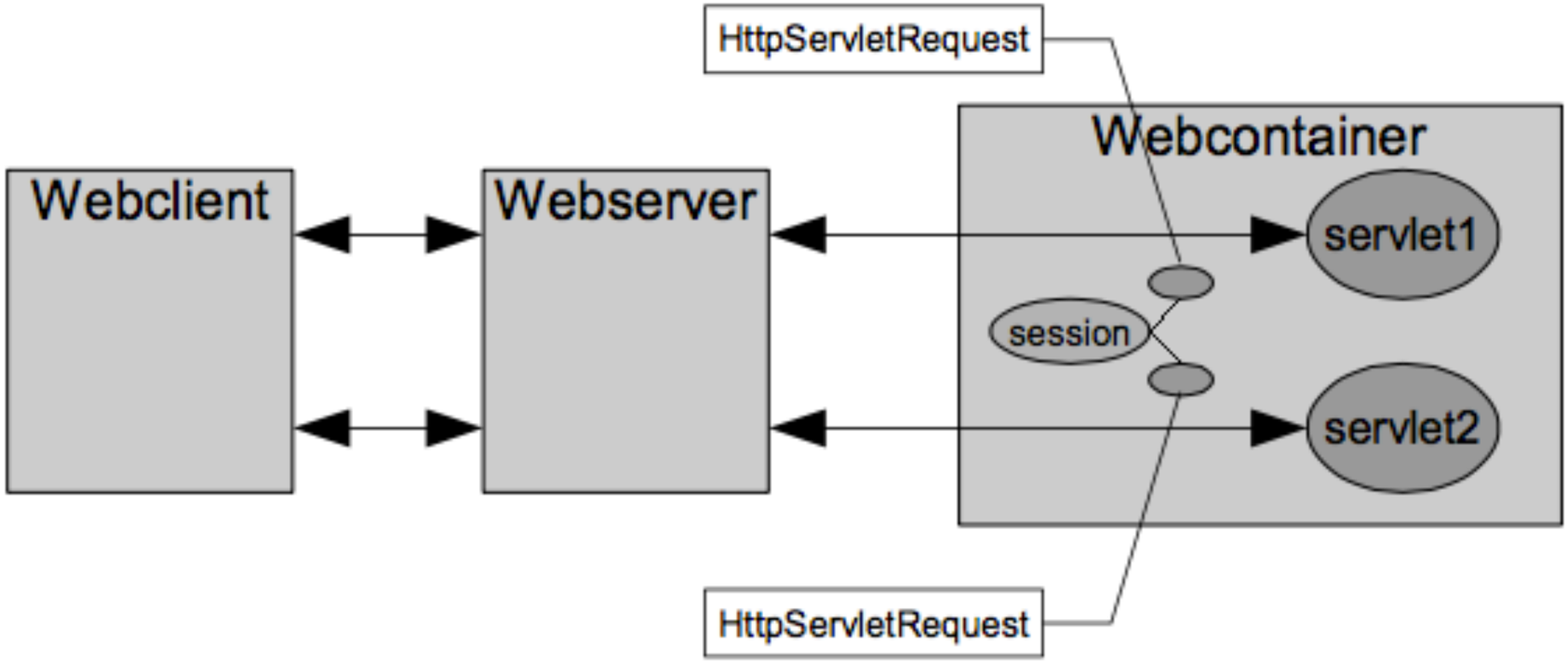
WEB-COMPONENTEN

SESSIES, SCOPE & JSP

SESSIES - STATE BIJHOUDEN OVER VERSCHILLENDE HTTP REQUESTS

- HTTP is van nature een stateless protocol
- informatie bijhouden over 1 of meerdere requests (bvb. shopping cart)



SESSIES - STATE BIJHOUDEN OVER VERSCHILLENDE HTTP REQUESTS

Het sessie-object is een object van de interface `HttpSession`. Deze interface beschikt o.a. volgende methoden:

Methode	***Omschrijving***
`setAttribute()`	Hiermee kunnen we eender welk object als attribuut aan het sessie-object toevoegen. Ieder attribuut heeft een naam die als parameter meegegeven wordt.
`getAttribute()`	Hiermee kunnen we een attribuut terug opvragen.
`getAttributeNames()`	Geeft een lijst van alle attributen die toegekend zijn aan het sessie-object.
`isNew()`	Geeft aan of deze sessie nieuw is en dus nog niet bevestigd is door de gebruiker.
`removeAttribute()`	Hiermee kan een attribuut terug verwijderd worden.

SESSIES - IMPLEMENTATIE

- cookies
- URL-encoding
- SSL-sessies

Men kan de gebruikte technieken eventueel instellen met de methode `setSessionTrackingModes()` van de *servlet context* of via volgende *tags* in ***web.xml***:

```
...
<session-config>
   <tracking-mode>COOKIE</tracking-mode>
</session-config>
...
```

In ons voorbeeld kunnen we gebruik maken van *URL rewriting*

```
out.print("<form method='get' action='" +
          response.encodeURL(request.getRequestURI()) + "'>");
```

Dit resulteert in volgende URL:

`Calculator;`**`jsessionid=6CE6EDD8B3B359269A34CD0594733F59`**

- timeout #setMaxInactiveInterval
- #invalidate

De levensduur kan in de *deployment descriptor* ***web.xml*** als volgt geconfigureerd worden:

```
<web-app>
   ...
   <session-config>
      <session-timeout>20</session-timeout>
   </session-config>
   ...
</web-app>
```

Tag	***Omschrijving***
`<session-config>`	Sessie configuratie.
`<session-timeout>`	De levensduur van sessie in minuten. Indien deze waarde kleiner of gelijk is aan 0, zullen sessies nooit verlopen.

- @WebListener

Voor het afhandelen van sessie-*events* zijn er volgende interfaces:

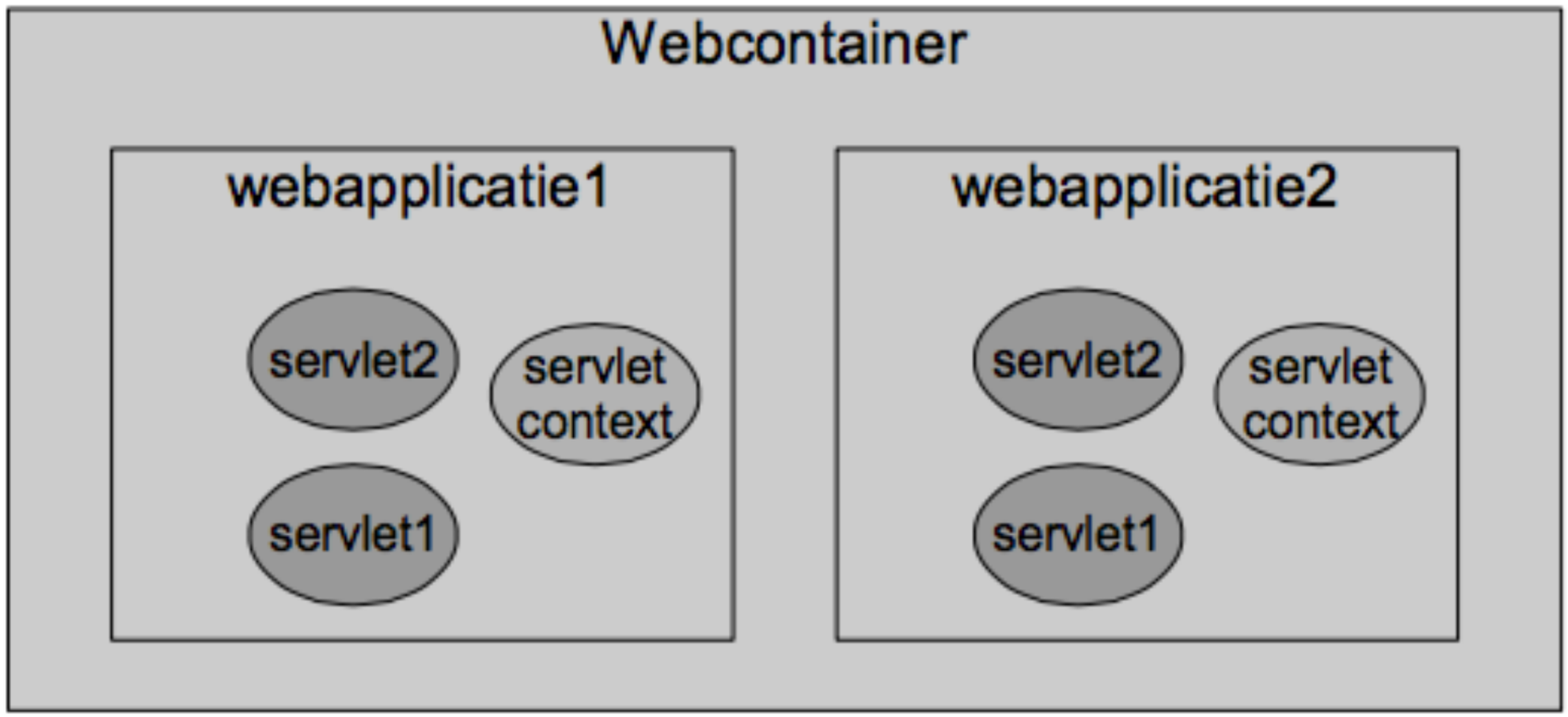
Interface	***Omschrijving***
`HttpSessionListener`	Luistert naar gebeurtenissen waarbij sessies gecreëerd en beëindigd worden.
`HttpSessionActivationListener`	Luistert naar gebeurtenissen waarbij sessies geactiveerd of gepassiveerd worden.
`HttpSessionAttributeListener`	Luistert naar gebeurtenissen waarbij attributen van een sessie toegevoegd, vervangen of verwijderd worden.
`HttpSessionBindingListener`	Luistert naar gebeurtenissen waarbij het object aan een sessie-object gebonden wordt.
`HttpSessionIdListener`	Luistert naar gebeurtenissen waarbij het identificatienummer van de sessie gewijzigd wordt.

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

@WebListener
public class VisitorSessionListener implements HttpSessionListener{
   private int visitorsTotal = 0;
   private int visitorsActive = 0;

   public void sessionCreated(HttpSessionEvent se) {
      visitorsTotal++;
```

- gedeeld door alle servlets in dezelfde container(web-applicatie) (en op dezelfde machine… dus niet in een cluster)



- parameters die gelden voor de gehele web-applicatie

```
        <param-name>name</param-name>
        <param-value>value</param-value>
    </context-param>
    ...
</web-app>
```

SERVLET CONTEXT

Methode	***Omschrijving***

`setAttribute()`
`getAttribute()`
`removeAttribute()`
`getAttributeNames()`
	Met deze methoden kunnen we attributen toevoegen, opvragen en verwijderen.

`getInitParameter()` `getInitParameterNames()`	Met deze methoden kunnen we de initialisatieparameters van de servlet context opvragen.

`getResource()`
`getResourceAsStream()`
`getResourcePaths()`
	Met deze methoden kunnen we bestanden opvragen die zich binnen de web-applicatie (WAR) bevinden.

`getContext()`	Met deze methode kunnen we het *servlet context*-object van een andere web-applicatie opvragen.
`getRealPath()`	Met deze methode kunnen we het absolute pad opvragen van een bestand dat zich binnen de web-applicatie bevindt.
`getContextPath()`	Geeft het contextpad van de huidige web-applicatie.
`getSessionCookieConfig()`	Geeft de configuratie van de *session tracking cookie*.

Methode	***Omschrijving***
`setSessionTrakingModes()`	Hiermee kan men de wijze van sessietracering instellen.
`addListener()`	Hiermee kan men programmatorisch een *listener* instellen.
`addFilter()`	Hiermee kan men programmatorisch een filter instellen.
`log()`	Hiermee kan men boodschappen naar het logbestand schrijven.

SERVLET CONTEXT - VOORBEELD

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

@WebListener
public class VisitorSessionListener implements HttpSessionListener{
   public static final String TOTAL = "visitorsTotal";
   public static final String ACTIVE = "visitorsActive";

   public void sessionCreated(HttpSessionEvent se) {
      ServletContext sc = se.getSession().getServletContext();
      int visitorsTotal = 1;
      int visitorsActive = 1;

      Integer total = (Integer) sc.getAttribute(TOTAL);
      if (total != null) {
         visitorsTotal = total + 1;
      }
      sc.setAttribute(TOTAL,visitorsTotal);

      Integer active = (Integer) sc.getAttribute(ACTIVE);
      if (active != null) {
         visitorsActive = active + 1;
      }
      sc.setAttribute(ACTIVE,visitorsActive);
   }

   public void sessionDestroyed
```

```
      ServletContext sc = se.getSession().getServletContext();
      Integer active = (Integer) sc.getAttribute(ACTIVE);

      int visitorsActive = 0;
      if (active != null) {
         visitorsActive = active - 1;
      }
      sc.setAttribute(ACTIVE,visitorsActive);
   }
}
```

SERVLET CONTEXT - VOORBEELD

We kunnen nu een *servlet* maken die de tellers ophaalt uit de *servlet context* en deze op de pagina toont:

```
@WebServlet("/Visitors")
public class VisitorServlet extends HttpServlet {
   public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
                                          throws IOException {
      request.getSession();
      int total = (Integer) getServletContext().getAttribute(
            VisitorSessionListener.TOTAL);
      int active = (Integer) getServletContext().getAttribute(
            VisitorSessionListener.ACTIVE);
      response.setContentType("text/html");
      response.setCharacterEncoding("UTF-8");
      try (PrintWriter out = response.getWriter()) {
         out.println("<!DOCTYPE html>");
         out.println("<html><head><title>");
         out.println("Visitors");
         out.println("</title></head>");
         out.println("<body>");
         out.println("Total visitors: " + total + "<br />");
         out.println("Current visitors: " + active + "<br/>");
         out.println("</body></html>");
      }
   }
}
```

SERVLET CONTEXT - EVENTS

implementeren:

Interface	***Omschrijving***
`ServletContextListener`	Luistert naar gebeurtenissen waarbij de context creëerd en vernietigd wordt.
`ServletContextAttributeListener`	Luistert naar gebeurtenissen waarbij attributen aan een context-object toegevoegd, vervangen of verwijderd worden. Deze interface wordt geïmplementeerd door de objecten die aan de webcontext als attribuut toegevoegd worden.

Methode	***Omschrijving***
`contextDestroyed()`	Wordt opgeroepen zodra het context-object vernietigd wordt.
`contextInitialized()`	Wordt opgeroepen zodra het context-object geïnitialiseerd wordt.

- Include(dynamisch insluiten)
  - ander servlet
  - JSP
  - HTML
- Forward (doorsturen naar een andere URL)

- via RequestDispatcher
  - request#getRequestDispatcher(path)#include()
    - relatief of tov root
  - context#getRequestDispatcher(path)#include()
    - steeds tov root
- context#getNamedDispatcher(name)#include()
  - servlet naam geconfigureerd in deployment descriptor

SERVLET - INCLUDE (VOORBEELD)

We illustreren dit met een voorbeeld. Stel dat we op verschillende pagina's van onze website het aantal bezoekers willen laten zien. We kunnen dan een afzonderlijke *servlet* maken die enkel deze gegevens weergeeft. We kunnen vervolgens deze *servlet* insluiten in andere *servlets*.

De in te sluiten *servlet* ziet er als volgt uit:

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

@WebServlet(name = "VisitorIncludeServlet",
            value="/VisitorInclude")
public class VisitorIncludeServlet extends HttpServlet {
   public void doGet(HttpServletRequest request,
         HttpServletResponse response) throws IOException {
      int total = (Integer) getServletContext().getAttribute(
            visitorsTotal);
      int active = (Integer) getServletContext().getAttribute(
            visitorsActive);
      @SuppressWarnings("resource")
      PrintWriter out = response.getWriter();
      out.println("Total visitors: " + total + "<br />");
      out.println("Current visitors: " + active + "<br/>");
   }
}
```

SERVLET - INCLUDE (VOORBEELD)

In dergelijke *servlet* is het niet toegelaten *header*-waarden van de *response* in te stellen. Dit wordt trouwens gewoon genegeerd. Voorts mag de *outputstream* of *writer* niet afgesloten worden; de insluitende *servlet* zal meestal immers nog gegevens willen toevoegen. We gebruiken daarom hier ook geen *try-with-resources* want deze zou de *writer* wel afsluiten. Om te voorkomen dat de IDE een waarschuwing geeft voegen we de extra annotatie `@SuppressWarnings("resource")` toe.

De insluitende *servlet* ziet er als volgt uit:

```
@WebServlet("/Including")
public class IncludingServlet extends HttpServlet {
   public void doGet(HttpServletRequest request,
         HttpServletResponse response) throws ServletException,
         IOException {
      request.getSession();
      response.setContentType("text/html");
      try (PrintWriter out = response.getWriter()) {
         out.println("<html>");
         out.println("<head>");
         out.println("<title>Including Servlet</title>");
```

```
         out.println("</head>");
         out.println("<body>");
         out.println("<div>Including another servlet</div>");
         RequestDispatcher disp = getServletContext()
            .getNamedDispatcher("VisitorIncludeServlet");
         disp.include(request, response);
         out.println("</body>");
         out.println("</html>");
      }
```

SERVLET - FORWARD

- via RequestDispatcher
  - ophalen cfr 'include'
  - #forward()
- typisch voor MVC (Model-View-Controller architectuur)

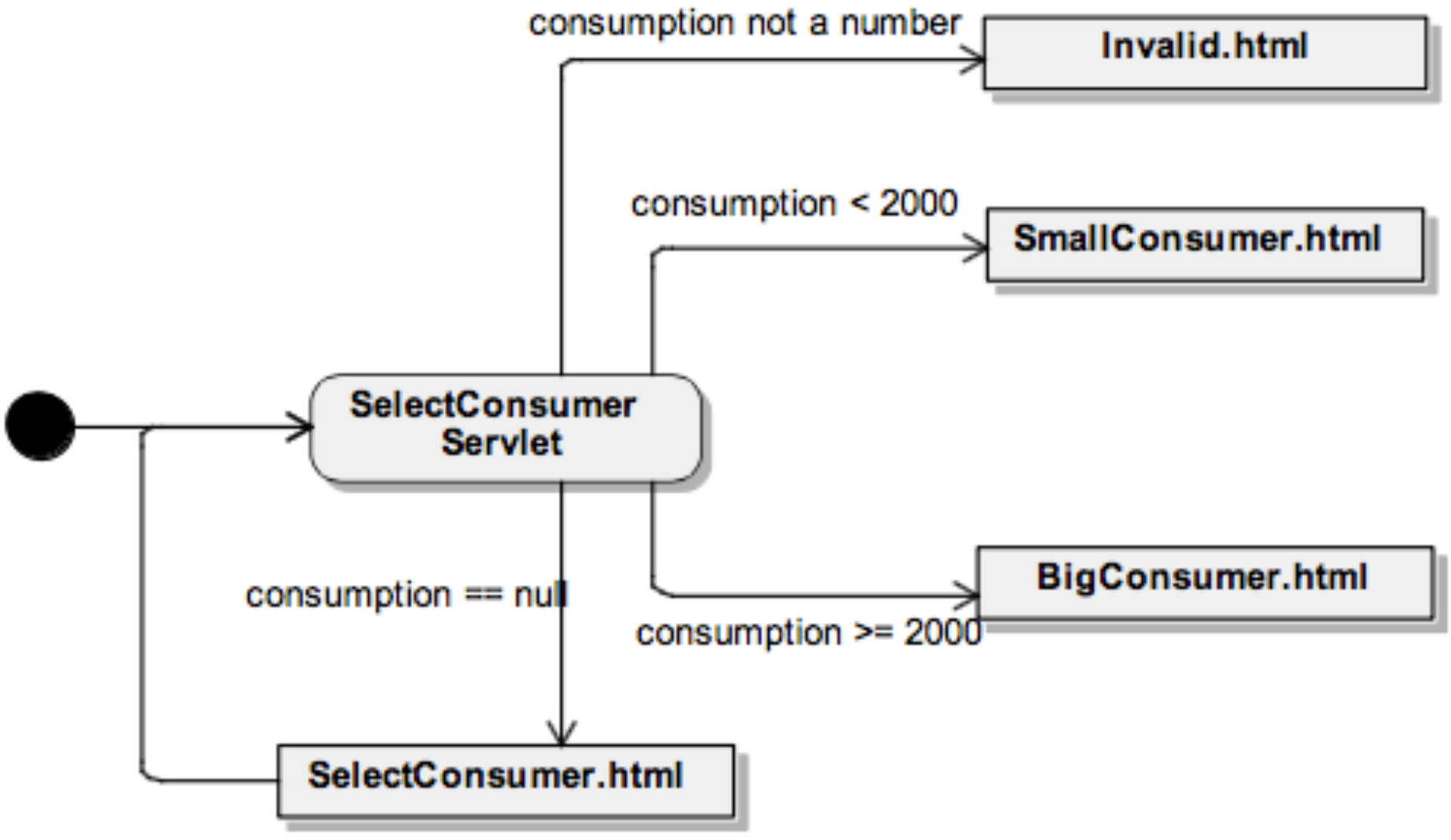
- extra gegevens meegeven via toevoegen attributen aan het request object

SERVLET - FORWARD (VOORBEELD)

In het volgende voorbeeld maken we een servlet ***SelectCustomerServlet*** die op basis van het energieverbruik de bezoeker doorstuurt naar de site voor kleinverbruikers of voor grootverbruikers. Bij ongeldige invoer wordt het verzoek doorgestuurd naar een foutpagina.

```
@WebServlet("/SelectCustomer")
public class SelectCustomerServlet extends HttpServlet {
   public void doGet(HttpServletRequest req,
         HttpServletResponse resp) throws ServletException,
         IOException {
      String page = "SelectConsumer.html";
      String consumption = req.getParameter("consumption");
      if (consumption != null) {
         int cons;
         try {
            cons = Integer.parseInt(consumption);
            if (cons < 2000) {
               page = "SmallConsumer.html";
            } else {
               page = "BigConsumer.html";
            }
         }
         catch (NumberFormatException ex) {
            page = "Invalid.html";
         }
      }
      RequestDispatcher disp = req.getRequestDispatcher(page);
      disp.forward(req,resp);
   }
}
```

SERVLET - FORWARD (VOORBEELD)



SERVLET - REDIRECT

- stuurt een commando naar de browser
  - via HTTP statuscode 3xx met URL
  - de browser gaat zelf de andere pagina ophalen
  - bvb verwijzen naar URL op een andere server

```
@WebServlet(value = "/Redirect",
     initParams = @WebInitParam(name = "url",
                                value = "http://www.noelvaes.eu"))
public class RedirectServlet extends HttpServlet {
   private String url;

   public void init() throws ServletException {
      url = getInitParameter("url");
      if(url == null) {
         throw new ServletException("url null");
      }
   }

   protected void doGet(HttpServletRequest req,
                        HttpServletResponse resp)
         throws ServletException, IOException {
      resp.sendRedirect(url);
   }
```

SERVLET - COOKIES

- kleine tekstbestanden
- worden weggeschreven in browser
- wordt bij elke request terug meegegeven aan de server
- response#addCookie

SERVLET - COOKIES

Eigenschap	***Accessors***	***Omschrijving***
Name	`getName()` `setName()`	Iedere *cookie* heeft een naam.
Value	`getValue()` `setValue()`	De inhoud van de *cookie*. Dit is steeds in de vorm van een string.
Comment	`getComment()` `setComment()`	Commentaar m.b.t. de *cookie*.
Domain	`getDomain()` `setDomain()`	*Cookies* worden door de browser enkel verzonden naar het domein waartoe ze behoren. Daarom bezit iedere *cookie* de naam van het domein.
Path	`getPath()` `setPath()`	*Cookies* worden enkel verzonden naar een bepaald pad van een bepaald domein.
MaxAge	`getMaxAge()` `setMaxAge()`	Een *cookie* heeft een bepaalde levensduur. Indien een *cookie* deze levensduur overschrijdt, wordt hij door de *client* verwijderd. De levensduur wordt opgegeven in seconden. Een negatieve levensduur betekent dat de *cookie* niet wordt opgeslagen en verwijderd wordt zodra het browservenster sluit. Een levensduur van 0 seconden verwijdert de *cookie* onmiddellijk.

Secure	`getSecure()`
	
	`setSecure()`
		Beveiligde *cookies* kunnen vertrouwelijke informatie bevatten en mogen enkel over een beveiligde verbinding naar de server gestuurd worden (HTTPS).

SERVLET - COOKIES (VOORBEELD)

```
@WebServlet("/SelectLanguage")
public class LanguageServlet extends HttpServlet {
   public static final String LANGUAGE = "language";

   public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
         throws IOException {
      response.setContentType("text/html");
      response.setCharacterEncoding("UTF-8");
      String language = null;
      // Get language from cookie
      Cookie cookies[] = request.getCookies();
      if (cookies != null) {
         for (Cookie cookie : cookies) {
            if (cookie.getName().equals(LANGUAGE)) {
               language = cookie.getValue();
               break;
            }
         }
      }
```

SERVLET - COOKIES (VOORBEELD)

```
// Get language from request parameter
if (language == null) {
   language = request.getParameter(LANGUAGE);
   if (language != null) {
      // Create new cookie
      Cookie cookie = new Cookie(LANGUAGE, language);
      cookie.setMaxAge(60);
      response.addCookie(cookie);
   }
}
try(PrintWriter out = response.getWriter()) {
   out.println("<html><header>");
   out.println("<title>Language Selection</title>");
   out.println("</header><body>");
   if (language == null) {
      // Print choices
      out.print("<a href='SelectLanguage?language=en'>");
      out.print("English</a><br/>");
      out.print("<a href='SelectLanguage?language=nl'>");
```

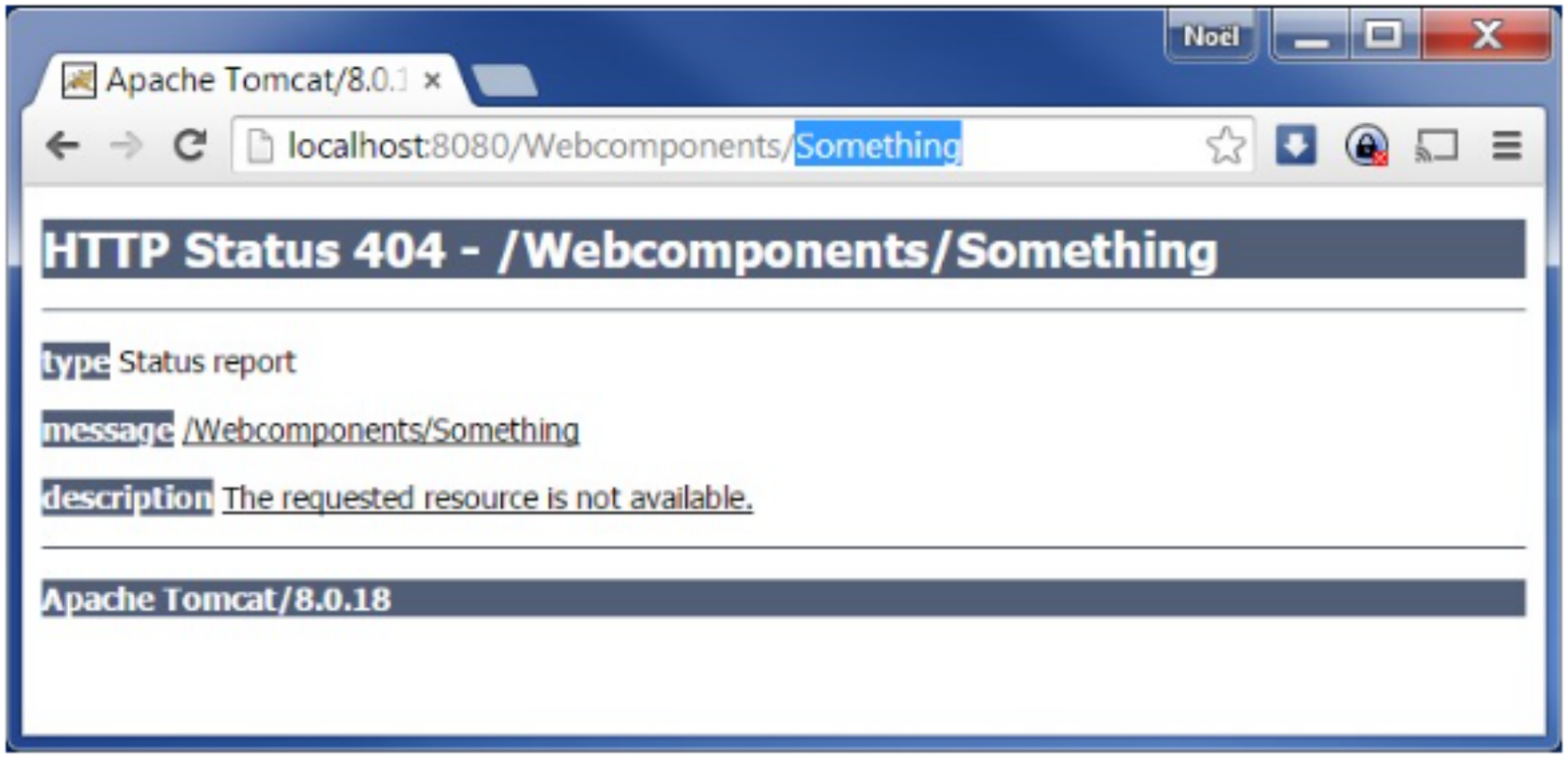
- ServletException

```
        } catch (SQLException e) {
            throw new ServletException(e);
        }
    }
```

```
    <error-page>
        <exception-type>java.sql.SQLException</exception-type>
        <location>/ErrorDatabase.html</location>
    </error-page>
```

Voor elk type *exception* kunnen we eventueel een eigen foutpagina gebruiken. Indien we

- web container error



In dit voorbeeld is het de foutcode 404 die erop wijst dat de informatie niet beschikbaar is. Ook dit is geen aangename foutpagina. We kunnen nu onze eigen pagina ervoor in de plaats stellen. De configuratie in ***web.xml*** ziet er als volgt uit:

```
<error-page>
   <error-code>404</error-code>
   <location>/Error404.html</location>
</error-page>
```

airbnb

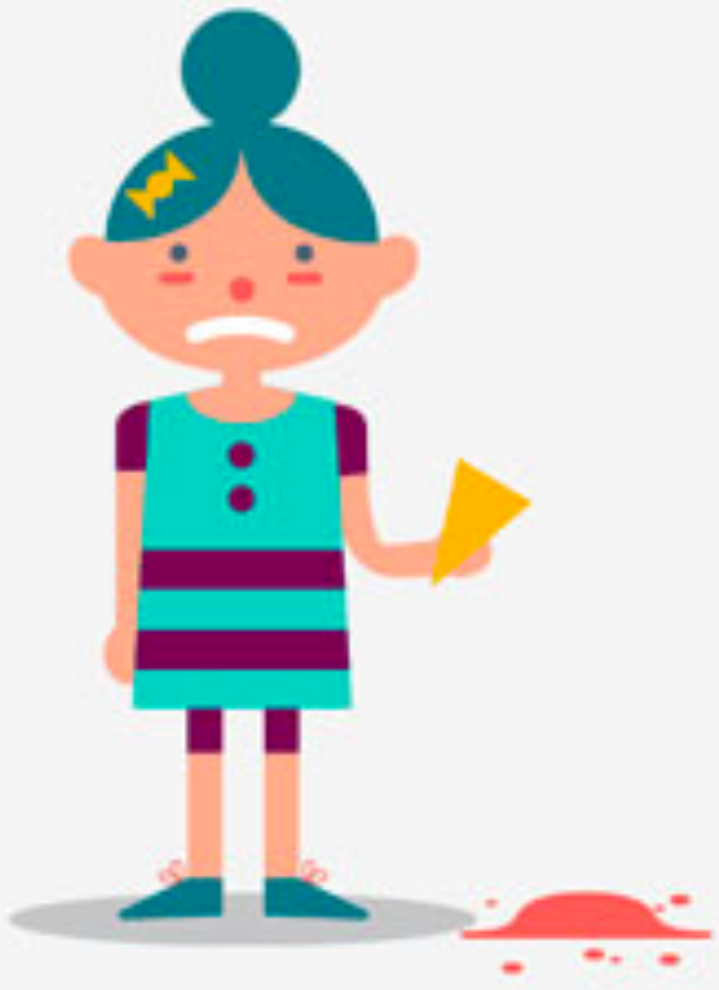
Oops!

We can't seem to find the page you're looking for.

Error code: 404

Here are some helpful links instead:

Home
Search
Help
Traveling on Airbnb
Hosting on Airbnb
Trust & Safety
Sitemap

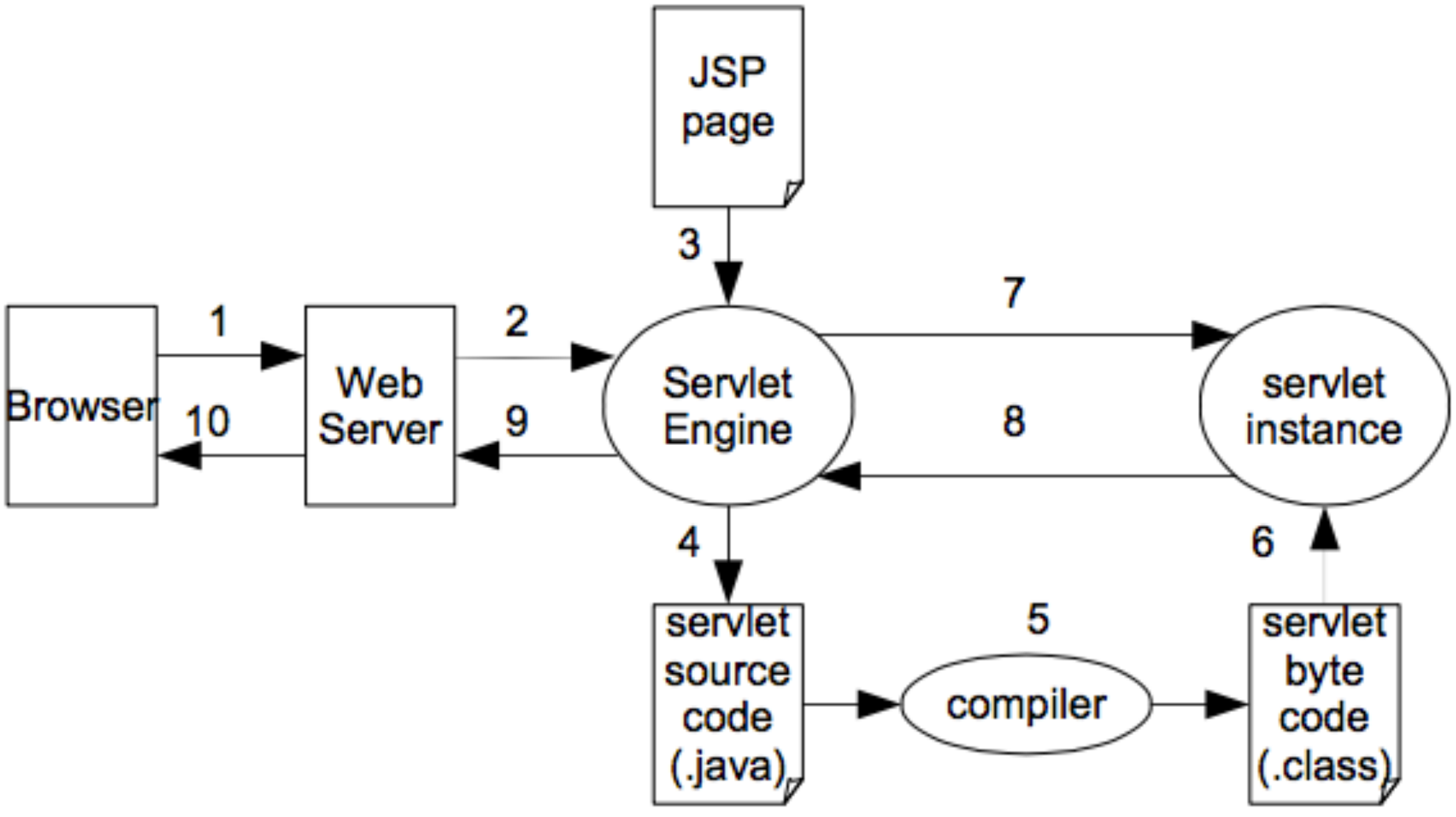


WEB-COMPONENTEN

JAVA SERVER PAGES

- dynamische web-pagina's
- gewone HTML-pagina met specifieke JSP tags
- achter de schermen gecompileerd naar een servlet
  - de 1e keer dat een JSP opgeroepen wordt, wordt de time-hit genomen

JSP - LIFECYCLE



JSP - "HELLO WORLD" SERVLET VS JSP

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

@WebServlet(value="/HelloWorld")
public class HelloWorldServlet extends HttpServlet {
   protected void doGet(HttpServletRequest request,
         HttpServletResponse response) throws IOException {
      response.setContentType("text/html");
      response.setCharacterEncoding("UTF-8");
      try (PrintWriter out = response.getWriter()) {
         out.println("<!DOCTYPE html>");
         out.println("<html>");
         out.println("<head>");
         out.println("<title>Hello World Servlet</title>");
         out.println("</head>");
         out.println("<body>");
         out.println("Hello World");
         out.println("</body>");
         out.println("</html>");
      }
   }
}
```

HelloWorld.jsp

```
<!DOCTYPE html>
<html>
<head>
<title>Hello World Page</title>
</head>
<body>
Hello World
</body>
</html>
```

JSP - DYNAMISCHE ELEMENTEN

- scriptlets
- expressions
- declarations

Scriptlets worden met volgende notatie in de JSP-pagina opgenomen:

```
<% Java-code %>
```

Binnen deze Java-code kan er gebruik gemaakt worden van een aantal objecten:

Object	***Klasse***	***Omschrijving***
`application`	`ServletContext`	Bevat de gegevens van de *web context*.
`config`	`ServletConfig`	Bevat de gegevens van de *servlet*-configuratie.
`exception`	`Throwable`	Het *exception*-object dat door een andere servlet gegenereerd werd. Dit is enkel beschikbaar in foutpagina's (zie later).
`out`	`JspWriter`	*Writer* om het resultaat naar toe te schrijven.
`page`	`Object`	Instantie van de servlet die de JSP-pagina verwerkt.
`pageContext`	`PageContext`	*Scope* object verbonden aan de pagina.
`request`	`HttpServletRequest`	Het *request*-object van het verzoek.
`response`	`HttpServletResponse`	Het *response*-object van het verzoek.
`session`	`HttpSession`	Het *session*-object van het verzoek.

```
<body>
<% out.println("Hello World"); %>
</body>
```

JSP - EXPRESSIONS

<%= *expression* %>

Dit is equivalent aan:

```
<% out.print(expression); %>
```

Stel dat we de tekst hebben opgeslagen in een string-object, dan kunnen we die als volgt afdrukken:

```
<html>
<head>
<title>Hello World Servlet</title>
</head>
<body>
<% String text = "Hello World"; %>
<%=text%>
</body>
</html>
```

JSP - DECLARATIONS

De variabele `text` die we in voorgaand voorbeeld gebruikt hebben, is een lokale variabele van de service-methode. We kunnen ook *member*-variabelen en *member*-methoden aan de gegenereerde *servlet* toevoegen. Hiervoor gebruiken we de volgende syntax:

```
<%! declarations %>
```

Stel dat we in ons voorbeeld een methode willen toevoegen die een string omkeert.

```
<html>
<head>
<title>Reverse</title>
</head>
<body>
<%!
// Declaration of a member variabele
private String text = "Hello World";

// Declaration of a member method
private String reverse(String t){
   StringBuilder sb = new StringBuilder(t);
   return sb.reverse().toString();
}
%>

<%=reverse(text)%>
<br />
</body>
</html>
```

JSP - PAGE DIRECTIVES

Met *page directives* kan men instellingen doen voor een JSP-pagina. De algemene syntax ziet er als volgt uit:

```
<%@ page
[ language="java" ]
[ extends="package.class" ]
[ import="{package.class | package.*}
[ session="true|false" ]
[ buffer="none|8kb|sizekb" ]
[ autoFlush="true|false" ]
[ info="text" ]
[ isELIgnored="true|false" ]
[ errorPage="relativeURL" ]
[ contentType="mimeType [ ; charset=c
"text/html ; charset=ISO-8859-1" ]
[ isErrorPage="true|false" ]
```

Attribuut	***Standaard***	***Omschrijving***
`language`	`java`	Geeft de taal aan die in de *scriptlets* gebruikt wordt. De enige toegestane taal is momenteel ***java***.
`extends`	-	Geeft aan dat de servlet afgeleid is van een andere klasse. Deze optie is enkel voor zeer geavanceerd gebruik.
`import`	-	Geeft dat aan dat volgende pakketten en/of klassen geïmporteerd dienen worden. De pakketten of klassen worden gescheiden door een komma.
`session`	`true`	Geeft aan of er een sessie-object gecreëerd wordt of niet. Het *session*-object is enkel beschikbaar indien dit attribuut *true* is.
`buffer`	`8kb`	De grootte van de buffer van de *outputstream*. De standaard grootte is 8kb.
`autoFlush`	`true`	Geeft aan of de buffer geledigd moet worden zodra hij vol is. Indien dit op *false* staat, wordt een *exception* gegenereerd zodra de buffer vol is.
`info`	-	Tekst die wordt teruggegeven door de methode `servlet.getServletInfo()`.
`isELIgnored`	`false`	Geeft aan of *Expression Language* uitgeschakeld is.
`errorPage`	-	De URL van de JSP-pagina die wordt opgeroepen zodra de servlet een *exception* genereert.
`isErrorPage`	`false`	Geeft aan dat deze pagina een foutpagina is die door een ander servlet werd opgeroepen. In dit geval is het *exception*-object beschikbaar.
`contentType`	`text/html`	MIME type voor het antwoord.
`pageEncoding`	`ISO-8859-1`	Karaktercodering voor het antwoord.
`trimDirectiveWhitespaces`	`false`	Verwijdert overtollige lege ruimte tussen script-elementen (sinds JSP 2.1).

JSP - JSP TO SERVLET

```
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import java.util.*;

public final class HelloWorld2_jsp extends
org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {

  String text ="Hello World";
  private String sayHello() {
     return text;
  }

  private static java.util.List

  public Object getDependants()
    return _jspx_dependants;
  }

  public void _jspService(HttpS
HttpServletResponse response)
        throws java.io.IOExcept

     ...

     out.write(" \r\n");
     out.write("\r\n");
     out.write("\r\n");
     out.write("<html>\r\n");
     out.write("<head>\r\n");
     out.write("<title>Hello Wo
     out.write("</head>\r\n");
     out.write("<body>\r\n");
     out.write("<table>\r\n");
    for(int i = 0; i < 10; i++)
      out.write("\r\n");
      out.write("<tr>\r\n");
      out.write("<td>");
      out.print(i);
      out.write("</td>\r\n");
      out.write("<td>");
      out.write("</tr>\r\n");
      out.print(sayHello());
      out.write("\r\n");
    }
```

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

<%!
String text ="Hello World";
private String sayHello() {
      return text;
}
%>

<html><head>
<title>Hello World</title>
</head>
<body>
<table>
<% for(int i = 0; i < 10; i++)
{ %>
   <tr>
      <td><%=i%></td>
      <td><%=sayHello()%></td>
   </tr>
<% } %>
</table></body></html>
```

JSP - INCLUDE @ FORWARD/REDIRECT

```
<%@ include file="filename" %>
```

Hiermee kan men de inhoud van een bestand in de JSP-pagina statisch insluiten. Dit gebeurt voordat de JSP-pagina vertaald en gecompileerd wordt. Deze techniek wordt vaak gebruikt om bijvoorbeeld een voettekst aan een pagina toe te voegen.

Net als bij de *servlets* is het ook mogelijk het antwoord van een andere *servlet* (of een statische webpagina) dynamisch in te sluiten op het moment dat de *servlet* wordt uitgevoerd. In dit geval gebruiken we de volgende syntax:

```
<jsp:include page="includedPage" />
```

Hierbij zal het insluiten dus gebeuren nadat de JSP-pagina vertaald en gecompileerd wordt. Voordeel hiervan is dat bij wijziging van de ingesloten pagina, de insluitende pagina niet gehercompileerd moet worden.

```
<jsp:forward page="forwardPage" />
```

JSP - INCLUDE @ FORWARD/REDIRECT

Zowel bij het insluiten als doorsturen wordt het originele verzoek, inclusief alle parameters doorgestuurd. Men kan evenwel aan een verzoek een parameter toevoegen met de volgende syntax:

```
<jsp:include page="..." >
      <jsp:param name="param1" value="value1"/>
```

Om een gemeenschappelijke hoofding en voettekst toe te voegen aan een verzameling pagina's, kan men ook gebruik maken van volgende configuratie in ***web.xml***:

```
<web-app>
   ...
   <jsp-config>
      <jsp-property-group>
         <url-pattern>/*</url-pattern>
         <include-prelude>/Header.html</include-prelude>
         <include-coda>/Footer.html</include-coda>
      </jsp-property-group>
   </jsp-config>
   ...
</web-app>
```

Met de *tag* `<jsp-property-group>` definiëren we een verzameling pagina's die beantwoorden aan het *url pattern*. Voor deze pagina's voorzien we een hoofding (*prelude*) en voettekst (*coda*).

WEB-COMPONENTEN

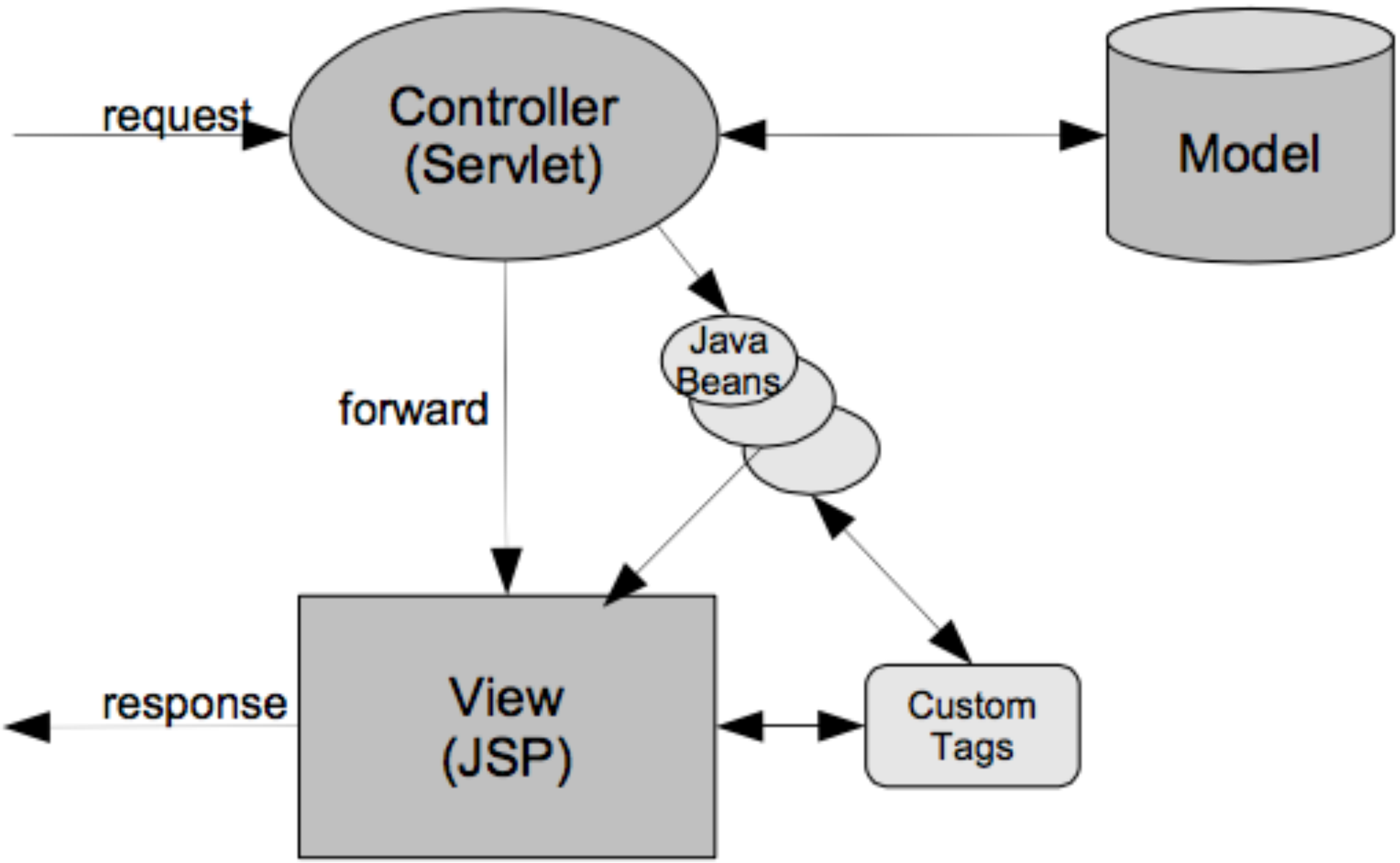
MODEL-VIEW-CONTROLLER

MVC

- Scriptlets = layout(html) en programmatie(java) verweven
  - minder gestructureerd
  - moeilijker te onderhouden
- MVC(design pattern)
  - *Separate the user interface into three interconnected components: the model, the view and the controller. Let the model manage the data, the view display the data and the controller mediate updating the data and redrawing the display.*

    https://github.com/iluwatar/java-design-patterns/tree/master/model-view-controller

MVC


request
Controller
(Servlet)
Model
Java
Beans
forward
response
View
(JSP)
Custom
Tags