JavaServer Pages

devAcademy

JSP - Concepto

- Normalmente, esperamos que un Servlet nos responda con una página HTML

- Generar páginas web dinámicamente desde servlets html, resulta un poco tedioso e inapropiado

JSP - Concepto

- Las JSP, son una alternativa en la generación dinámica de páginas HTML como respuesta del servidor
- La ventaja: En un mismo documento, defino **partes estáticas** (HTML generalmente) y lo mezclo con **partes dinámicas** (Código JAVA)

JSP - Ejemplo

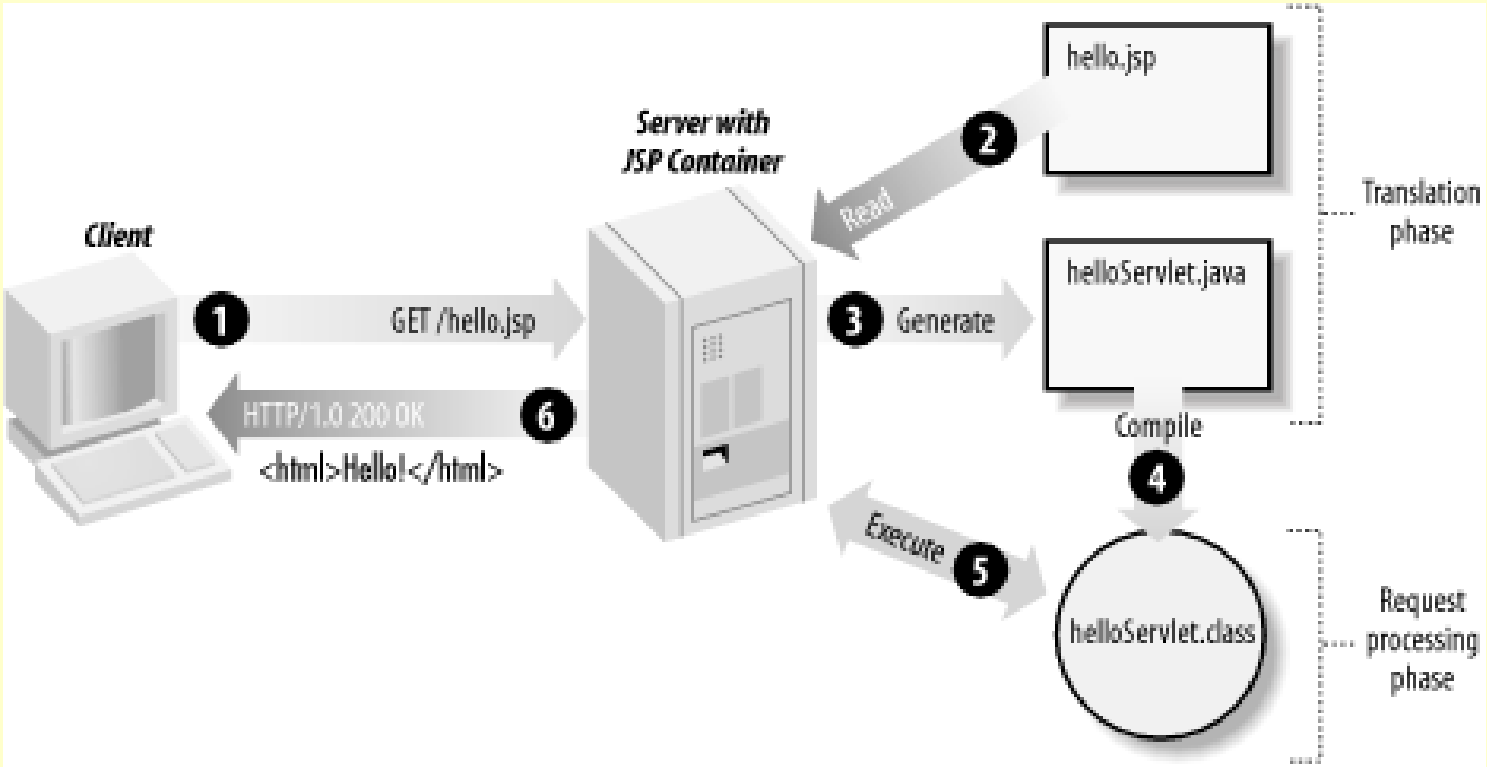
```
<!DOCTYPE html>
<html>
<head><title>SaludoJSP</title></head>
<body>
<h1>Hola <%if (request.getParameter("cliente")!=null)
                out.println(request.getParameter("cliente"));
          %>
</h1>
</body>
```

JSP - Concepto

- En realidad, un archivo JSP, es tratado por el Servidor como un SERVLET

- Pero tanto el uso como el ciclo de vida de un JSP, es implícito para el servidor y más transparente para el usuario

JSP - Arquitectura



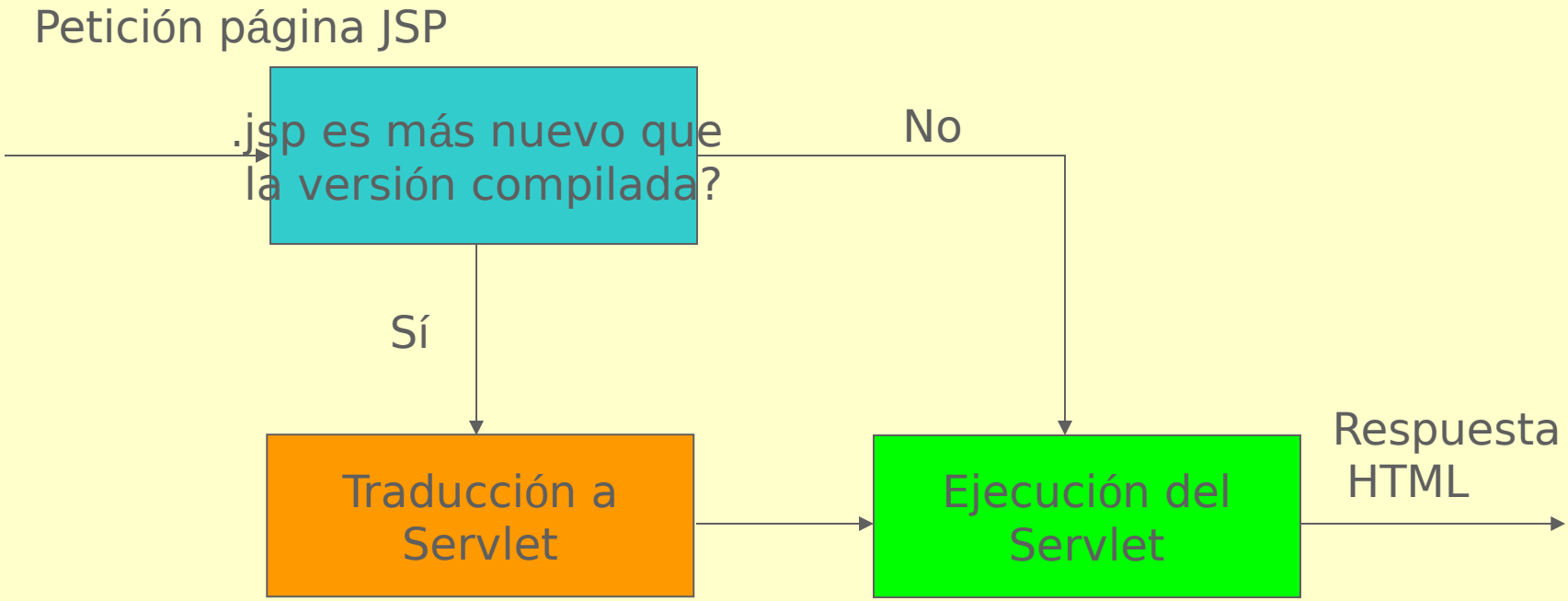
Traducción y proceso

Hay 2 fases claramente diferenciadas:

TRADUCCIÓN → Parseo, Generación .java, Generación .class

PROCESO → Ejecución del servlet generado y generación de la respuesta

Traducción y proceso



Métodos de un JSP

- jspInit () → Se ejecuta cuando se instancia por el Servidor
- _jspService () → Se ejecuta cada vez que se procesa un servlet (service Servlets)
- jspDestroy() → Se ejecuta cuando el servidor decide eliminarlo

JSP – Comentarios

<%-- comentario JSP --> No se envía al cliente (ignorado en la fase de traducción)

<!-- comentario HTML --> Sí se envía al cliente en el documento HTML

Ejemplo JSP

```
<%@ page language='java'
    contentType='text/html;charset=iso-8859-1'%>
<%@ page import='java.util.Date' %>
    <html> <head> <title>Hola Mundo</title> </head>
    <body> <p>Hola, esto es una página JSP.</p>
    <p>La hora del servidor es <%= new Date() %></p>
    </body>
</html>
```

De JSP a Servlet

```
 _jspService{
out.write("<html>\n ");
out.write(" <head>\n ");
out.write(" <title>Hola Mundo</title>\n ");
out.write(" </head>\n ");
out.write(" <body>\n <p>Hola, esto es una página JSP.</p>\n ");
out.write(" <p>La hora del servidor es ");
out.print( new Date() );
out.write("</p>\n ");
out.write(" </body>\n");
 out.write("</html>\n");  …}
```

JSP en el Cliente : HTML

```
<html> <head>
    <title>Hola Mundo</title> </head>
    <body> <p>Hola, esto es una página JSP.</p>
    <p>La hora del servidor es Tue Jan 26 13:25:34
    CET 2015</p>
</body> </html>
```

JSP – Sintaxis

Elementos de una JSP:

- SCRIPT
  - Código Java: Expresiones, Declaraciones y Scriptlets
- DIRECTIVAS
  - Page, include, bibliotecas de etiquetas
- ACCIONES
  - Uso de JavaBeans, transitar entre páginas

JSP – Elementos Script

Código JAVA

- Expresiones <%= %>
- Scriptlets <% %>
- Declaraciones <%! %>

JSP – Expresiones

<%= Expresión %> Se traduce y su resultado, se convierte a un String y será mostrado en la página, como cualquier otro elemento estático

En el servidor, ser hará un out.println (expresión); Es información que quiero Mostrar al cliente

JSP – Expresiones

Ejemplo <%= new java.util.Date() %>

En el servlet out.println (new java.util.Date())

También puedo definir un elemento de tipo expresión, usando **sintaxis XML**, de la forma:

<jsp:expression> Expresion **</jsp:expression>**

Variables predefinidas

request → El objeto HttpServletRequest
response → El objeto HttpServletReponse
session → El objeto HttpServletSession asociado
out → El fichero de salida JspWriter
application → El objeto ServletContext
config→ El objeto ServletConfig
pageContext → PageContext
page → Acceso directo al Servlet generado*
exception → Información sobre excepciones

PageContext

La variable pageContext la podemos usar Para setear y obtener atributos de diferentes ámbitos y además, nos da acceso al resto de variables predefinidas y a otros elementos

PageContext

Ejemplo de uso:

pageContext.setAttribute("UName", id, PageContext.SESSION_SCOPE); //guarda el atributo “UName” con valor Id en la sesión

pageContext.setAttribute("UName", id); //guarda el atributo “UName” con valor Id en la página

pageContext.getRequest();//obtiene una referencia al objeto HttpServletRequest

JSP- Scriptlets

<% Código Java %> El código que contenga, pasa a formar parte del método jspService()

Ejemplo

```
<% String urlGet = request.getQueryString();
    out.println("URL GET : " + urlGet);
    response.setContentType("text/plain"); %>
```

JSP- Scriptlets

<% Scritplets %> También tengo opción de definirlo con etiquetas **XML**:

```
<jsp:scriptlet> String urlGet =
 request.getQueryString();
 out.println("URL GET : " + urlGet);
  </jsp:scriptlet>
```

Expresiones + Scriptlets

```
<head>
<%String usuario=
    session.getAttribute("Username");%>
<body>
Nombre = <%=usuario>
```

Scriptlets + HTML

```
<% if (hello) { %>
        <P>Hello, world
    <% } else { %>
        <P>Goodbye, world
    <% } %>
```

Declaraciones

<%! Declaración %> El código, formará parte de la sección de declaración del Servlet generado, fuera de cualquier método

Ideal para declarar métodos privados o variables a nivel de clase

Declaraciones

<%! Declaración %> Por tanto, podré referenciar a los métodos o atributos definidos en declaraciones, desde cualquier otra parte de la página

Declaraciones

Ejemplo

```
<%! private Usuario usuario = null;
      public void jspInit()
    {
    usuario = new Usuario ();
    }
 %>
```

Nota: Usando declaraciones, puedo sobreescribir los métodos jspInit() y jspDestroy()

JSP – Directivas

- Qué son: Aportan información de alto nivel al Servlet en que se traducirá nuestro jsp, como:

  -¿Qué clases son importadas?

  - Sesiones, páginas de errores

  - Tipos MIME

  - ¿Qué contenidos incluyo?

JSP – Directivas

- Las hay de 3 tipos:
  - Page
  - Include
  - Taglib

Veremos las dos primeras, y dejamos para más tarde Taglib, por ser más avanzado su uso

JSP – Page Import

```
<%@ page import="java.util.*"%>
…
<%
List lista_nombres = new ArrayList();
…
%>
```

JSP – Page Type

<%@ page contentType="text/html"%>

Si estoy creando una página web, es decir, mi código de la plantilla es HTML, indicaré ese tipo MIME

Si estuviera gereando un xml desde mi JSP, debería indicar application/*xml*

JSP – Page Session

<%@ page session="true"%>

Valor por defecto→ El jsp, irá asociado a una sesión. La variable session, será inicializada

<%@ page session="false"%>

Si indico false→ El jsp, no irá asociado a una sesión. Dentro, el servidor no creará una sesión

JSP – errorPage

<%@ page errorPage=“error.jsp”%>

Indico dónde debe dirigirse la excepción, al no ser capturada en esta página**

<%@ page isErrorPage =“true”%>

Hacemos que la variable exception, haya sido instanciada y contenga información útil relativa a la excepción generada

JSP – INCLUDE

- Con el uso de esta directiva, podemos integrar en nuestro jsp, trozos de otras páginas que queremos incluir

- Es parecido al método include que usábamos en Servlets, pero más fácil y potente de usar

JSP – INCLUDE

Hay dos formas de usar esta característica:

<%@ include file="..”%> - Tiempo de traducción (como directiva – dentro del mismo Servlet)

<jsp:include page="..”/> -Tiempo de procesamiento (como acción – se genera otro Servlet)

JSP – Acciones

Las acciones permiten integrar código Java que es ejecutado en la fase de procesamiento (al contrario que ocurre con las Directivas, usadas en la fase de traducción)

<jsp:action_name attribute="value" />

Las acciones, se expresan en XML y es una forma abreviada, y estándar de definir código Java en nuestro JSP

JSP – Acciones

Hay 2 tipos de acciones, predefinidas y personalizadas. Las predefinidas, son:

Jsp:forward

Jsp:param

Jsp:include

Jsp:useBean / jsp:getProperty / jsp:setProperty

Para usar las personalizadas*, deberemos especificar una directiva taglib

JSP:Forward

Jsp:forward Redirijo una petición

```
<html>
<head> <title>Redirección Action Tag</title>
</head>
<body>
<jsp:forward page="second.jsp" />
</body>
</html>
```

JSP:Param

Jsp:param Paso de parámetros a otro JSP

```
<jsp:forward page="second.jsp">
    <jsp:param name ="date" value="20-05-2012" />
     <jsp:param name ="time" value="10:15AM" />
     <jsp:param name ="data" value="ABC" />
</jsp:forward>
```

JSP:Include

Jsp:include Incluyo una página en la petición actual

```
<html>
<head> <title>Redirección Action Tag</title>
</head>
<body>
<jsp:include page=“ejemplo.jsp"  />
</body>
</html>
```

JSP – BEANS – Rdo.

- Un bean, es una clase de Java normal, que:
  - Tiene un constructor sin parámetros
  - Atributos privados, accesibles por getNombreAtributo y setNombreAtributo
  - Si un atributo es boolean, debe accederse por el método isNombreAtributo

JSP – Bean Persona

public Persona{
private int edad; private boolean mayorEdad;
private String nombre;
Persona () { }
public String getNombre()
public String getEdad()
public String setNombre(String nombre)
private void setEdad(int edad)
private boolean isMayorEdad()

JSP – Action useBean

- Desde un jsp, podemos acceder directamente a una clase Java (Bean) que siga el estándar, de la forma usando la acción :useBean
  - <jsp:useBean id="nombreBean" class="paquete.Clase"/>
  - <jsp:setProperty name="nombreBean" property="propertyName" value="propertyValue"/>
  - <jsp:getProperty name="nombreBean" property="propertyName" />

JSP:useBean

<jsp:useBean id="juan" class="curso.Persona"/>

Esto es equivalente a

<% curso.Persona juan = new curso.Persona() %>

Ventajas :

-Toma de parámetros automática

-Es más fácil compartir objetos en distintas páginas y/o servlets

JSP:setProperty

<jsp:setProperty name="juan" property="edad"
value="28"/>

Esto es equivalente a

<% juan.setEdad(28)%>

JSP:getProperty

<jsp:getProperty name="juan" property="edad"/>

Esto es equivalente a

<%=juan.getEdad()%>

Uso clases de Java, sin emplear código Java explícitamente.

JSP – BEANS-FORMS

Asocio directamente el parámetro que viene en el formulario

```
<jsp:useBean id="juan" class="curso.Persona"/>
<jsp:setProperty
    name="juan"
    property="edad"
    param="edad"/>
```

JSP – BEANS-FORMS

Asocio directamente TODOS parámetros que viene en el formulario

```
<jsp:useBean id="juan" class="curso.Persona">
<jsp:setProperty
    name="juan"
    property="*">
```

JSP – BEANS - SCOPE

- Puedo emplear el atributo scope, para indicar el ámbito en que nuestro objeto Bean es instanciado

<jsp:useBean id=“juan” class=“curso.Persona”
scope=“application/”>

JSP – BEANS - SCOPE

-ÁMBITOS

PAGE – PageContext (defecto)

REQUEST - HttpRequest

SESSION - HttpSession

APPLICATION – ServletContext

Indicando el scope, mi clase bean, se almacena como atributo del ámbito indicado

JSP – Setting condicional

<jsp:useBean id=“paco”
class=“curso.Persona”
scope=“application”> <jsp:setProperty name=“paco” edad=“36”>**</jsp:useBean>**

Indicando setProperty así, la instrucción, sólo se ejecutará una vez (cuando el objeto paco se crea)

MVC

El Modelo Vista Controlador, es un patrón de diseño muy común en el desarrollo de aplicaciones web. Permite estructurar el código en 3 capas o paquetes:

- Modelo: El “negocio”, la funcionalidad de mi progrma en si
- Vista: Lo que el usuario verá en su navegador
- Controlador: La parte encargada de recibir y gestionar las peticiones del usuario

MVC

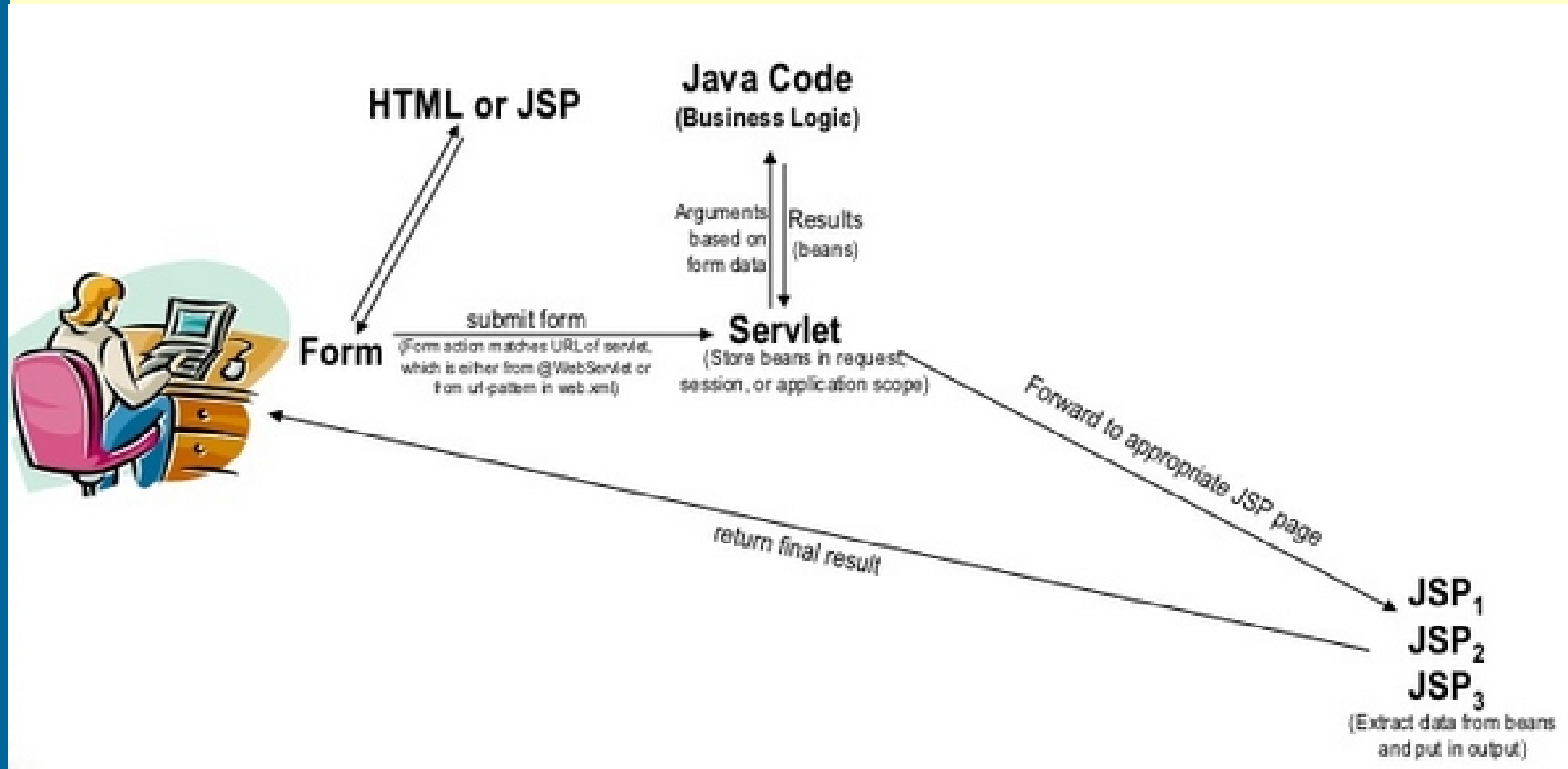
Con este modelo de abstracción funcional, consigo separar la definición de mis servicios (Controlador) con el cómo los resuevlo (Modelo) de qué ve el usuario (Vista)

Desarrollando de esta manera, obtengo numerosas ventajas en cuanto a metodología de trabajo y mantenimiento

MVC - Ventajas

1. Pueden existir grupos de trabajo independientes, encargados de cada capa
2. Separo el qué hago, del cómo lo hago, de qué muestro. Podría modificar una página, sin alterar el resto del proyecto (pej)
3. Código estructurado y más fácil de mantener
4. El proyecto puede ser flexible ante cambios en la tecnología de una capa, sin afectar al resto

MVC


HTML or JSP
Java Code
(Business Logic)
Arguments based on form data
Results (beans)
Form
submit form
(Form action matches URL of servlet, which is either from @WebServlet or from url-pattern in web.xml)
Servlet
(Store beans in request, session, or application scope)
Forward to appropriate JSP page
return final result
JSP1
JSP2
JSP3
(Extract data from beans and put in output)

MVC - Método

1. Un Servlet, recibe las peticiones
2. Desde un Servlet, invoco a otras clases Java que resuelven la tarea invocada por el usuario
3. El resultado, es almacenado en JavaBeans
4. Los JavaBeans son almacenados en el ámbito apropadio (request, sesión o aplicación)
5. Redirijo (forward) a un JSP
6. El Jsp obtiene los datos almacenados en las clases Bean de Java

Scope – Recordatorio

- Request: Cuando quiero que los resultados sean visibles sólo en la página a la que redirijo
- Session: Cuando quiero que los resultados sean visibles a lo largo de todas las ventanas que el usuario abra
- Context: Cuando quiero que sea visible para todos los usuarios en todas las páginas

Expression Lenguaje

- A la hora de mostrar los resultados de un JSP, se ha definido un lenguaje muy sencillo, para que las personas que no sepan Java, puedan acceder a ellos.

- Definido como “EL”, usa una sintaxis muy económica (más que Java) para ser usada dentro de JSP que respetan el uso del MVC y su objeto es sólo mostrar información

Expression Lenguaje

- El código EL provee acceso fácil a:
  - Objetos
  - Beans
  - Colecciones
  - Elementos HTTP (parámetros, cabeceras, cookies)

Sintaxis

${nombreVariable} //variable primitiva
${bean.nombreCampo} //objeto Bean
${bean.c.nombreCampo} //objeto compuesto
${colección[n]} //lista o array
${map[clave]} //mapa
${‘literales’} //comillas simples

Sintaxis

- Además, puedo definir expresiones como el if ternario **${test? exp1:exp2}** y acceder a las varibles predefinidas como **${session.id}** pero se desaconseja como norma general, puesto que EL sólo debería usarse para tareas de “MOSTRAR” (la lógica, en los Servlets)

Ejemplos EL 1 Bean

```
<BODY>
<jsp:useBean id="randomNum"
type="coreservlets.NumberBean"
scope="request" />
<H2>Random Number:
<jsp:getProperty name="randomNum"
property="number" /></H2>
</BODY></HTML>
```

Ejemplos EL 1

<BODY>

<H2>Random Number:

${randomNum.number}

</H2>

</BODY></HTML>

Ejemplos EL 2 Ámbitos

```
public class ScopedVars extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
request.setAttribute("attribute1“ , "First Value");
HttpSession session = request.getSession();
session.setAttribute("attribute2", "Second Value");
ServletContext application = getServletContext();
application.setAttribute("attribute3", new java.util.Date());
request.setAttribute("repeated", "Request");
session.setAttribute("repeated", "Session");
application.setAttribute("repeated", "ServletContext");
RequestDispatcher dispatcher = request.getRequestDispatcher
("/WEB-INF/results/scoped-vars.jsp"); dispatcher.forward(request, response); }}
```

Ejemplos EL 2

```
<!DOCTYPE ...>
<BODY>
<UL> <LI><B>attribute1:</B> ${attribute1}
<LI><B>attribute2:</B> ${attribute2}
<LI><B>attribute3:</B> ${attribute3}
<LI><B>Source of "repeated" attribute:</B>${repeated}
</UL>
</BODY></HTML>
```

Ej EL 3 Compuestos

public class Employee {
private Name name;
private Company company;..}

public class **Company** {
private String companyName;
private String business; ..}

public class **Name** {
private String firstName;
private String lastName; ..}

Ejemplos EL 3

```
public class BeanProperties extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
Name name = new Name(“Juan", “Pérez");
Company company = new Company(“DEV ACADEMY");
Employee employee = new Employee(name, company);
request.setAttribute("employee", employee);
RequestDispatcher dispatcher = request.getRequestDispatcher
("/WEB-INF/results/bea
n-properties.jsp");
dispatcher.forward(request, response);
}
```

Ejemplos EL 3

```
<!DOCTYPE ...>
<UL>
<LI><B>First Name:</B>
${employee.name.firstName} <!- -sería imposible con jsp:getProperty - ->
<LI><B>Last Name:</B>
${employee.name.lastName}
<LI><B>Company Name:</B>
${employee.company.companyName}
<LI><B>Company Business:</B>
${employee.company.business}
</UL>
</BODY></HTML>
```

Ejemplos EL 4 Collection

```
public class Collections extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
String[] firstNames = { "Bill", "Scott", "Larry" };
List <String> lastNames = new ArrayList<String>();
lastNames.add("Ellison"); lastNames.add("Gates"); lastNames.add("McNealy");
request.setAttribute("first", firstNames);
request.setAttribute("last", lastNames);
RequestDispatcher dispatcher = request.getRequestDispatcher
("/WEB-INF/results/collections.jsp");
dispatcher.forward(request, response);
}
```

Ejemplos EL 4

```
<!DOCTYPE ...>
<BODY>
<UL>  <LI> ${first[0]} ${last[0]}
            <LI>${first[1]} ${last[1]}
            <LI> ${first[2]} ${last[2]}
</BODY></HTML>
```

JSP – Directivas - TagLib

- Como hemos visto antes, las acciones, nos permiten indicar código Java mediante XML
- Para referirnos a acciones predefinidas (jsp:forward, jsp:include, jsp:useBean, etc) no necesitamos más que usarlas
- Pero si queremos, podemos definir acciones personalizadas, que nos ayuden a generar código Java

JSP – Directivas - TagLib

- Para referirnos a acciones personalizadas, deberemos indicar la directiva taglib, donde detallamos la ruta donde se encuentran el código de nuestras etiquetas

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core"
    prefix="c" %>
```

JSP – Directivas - TagLib

- Si usamos etiquetas en lugar de código Java, estamos permitiendo que en mis JSP, se evite el código Java y limitarme a aspectos de presentación
- Además, suponiendo un entorno profesional, puede haber programadores sólo trabajando con tags, siendo meros usuarios del código Java que hay detrás, sin necesidad de saber Java.

JSTL

- Java Standar Tag Library, es una biblioteca de etiquetas predefinidas, que nos ayuda a invocar acciones ya implementadas, que nos ayudan a simplificar la presentación

- La librería, se acompaña en la distribución de Tomcat

JSTL + EL

- Con JSTL, vamos a poder crear ESTRUCTURAS como **sentencias** condicionales y BUCLES
- Con EL vamos a acceder a la **información** de los objetos
- El uso combinado de ambos, nos darán JSP’s con un código limpio y abreviado

JSTL

- Los grupos de librerías más importantes, son:

  - CORE <c:>
  - FORMATO <fmt:>
  - SQL <sql:>
  - XML <xml:>

JSTL CORE

- La librería core es la más empleada ya que contiene las instrucciones Java más comunes
- Es donde vamos a centrar nuestro interés

<c:out> Tags JSTL

```
<html>
<head>
    <title>c:out Tag Example</title>
</head>
<body>
    <c:out value="${‘Ejemplo de c:out'}"/>
</body>
</html>
```

<c:set> Tags JSTL

```
<body>
<c:set var="name" scope="application"
value=“Jose Moreno"/>
<a href="display.jsp">Mostrar</a> </body>


//display.jsp
<c:out value="${name}"/>
```

<c:if> Tags JSTL

```
<body>
<c:if test="${edad >= 18}">
    <c:out value=“Puede votar!"/>
</c:if>
<c:if test="${age < 18}">
    <c:out value=“Aún no puedes votar!"/>
</c:if> </body>
```

<c:when> <c:otherwise>

```
<body>
 <c:choose>
    <c:when test="${n1 < n2}">${"n1 menor q n2"}
    </c:when>
    <c:when test="${n1 > n2}">${"n2 menor q n1"}
    </c:when>
    <c:otherwise> ${“Son iguales!"} </c:otherwise>
  </c:choose>
</body>
```

<c:forEach> JSTL

```
<body>
    <c:forEach var="counter" begin="1" end="10">
        <c:out value="${counter}"/>
    </c:forEach>
</body>
```

<c:forTokens> JSTL

```
<body>
    <c:forTokens items=“online.es" delims="."
    var="site">
        <c:out value="${site}"/>
    </c:forTokens>
</body>
```

<c:forTokens> JSTL

```
<body>
    <c:forTokens items=“online.es" delims="."
    var="site">
        <c:out value="${site}"/>
    </c:forTokens>
</body>
```