Proyecto de Aplicaciones Web

Servlets + JSP

JSP – Java Server Pages

- Es una tecnología que permite combinar HTML estático con contenido dinámico.
- Las páginas JSP se traducen automáticamente a *servlets* que luego se compilan.

Ventajas de JSP

- Los *servlets* tradicionales se destacan en el procesamiento de datos.
- Las páginas JSP se destacan en la presentación de los datos.

Estrategias de implementación

Aplicaciones Simples

- Invocar código Java directamente o indirectamente
- Usar arquitectura MVC + Java beans
- Usar arquitectura MVC + JSP EL (Expression Language)
- Definir y usar nuevos tags.

Aplicaciones Complejas

JSP - Sintaxis Básica

- Comentarios:

```
<%-- es un comentario --%>
```

- Expresiones: Se evalúan y envían a la salida en cada *request*

```
<%= expresión Java %>
```

- *Scriptlets*: Se evalúan en cada *request*.

```
<% código Java %>
```


JSP - Sintaxis Básica

- Declaraciones: Se convierten en parte de la definición de la clase del *servlet*.

```
<%! definición de propiedades y/o métodos %>
```

- Directivas: Permite incluir archivos , manipular *headers*, agregar *taglibs*, etc.

```
<%@ directive att="val" %>
```

- Acciones: Se realizan en cada *request*

```
<jsp:xyz>....</jsp:xyz>
```

Código Java en páginas JSP

Las páginas JSP ofrecen un conjunto de variables pre-definidas:

- **request**: el *HttpServletRequest*
- **response**: el *HttpServletResponse*
- **session**: el *HttpSession*
- **out**: el *PrintWriter* de salida
- **application**: el *ServletContext*

Ejemplo

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
        <TITLE>Demo JSP</TITLE>
        <LINK REL=STYLESHEET HREF="style.css" TYPE="text/css">
</HEAD>
<BODY>
<H2>Demo JSP</H2>
  <LI>La hora actual es: <%= new java.util.Date() %>
  <LI>El parámetro <CODE>mensaje</CODE> vale:
                        <%= request.getParameter("mensaje") %>
</BODY>
</HTML>
```


Scriptlets en páginas JSP

Se puede generar un fragmento equivalente al anterior escribiendo un fragmento en Java que accede a las variables pre-definidas:

```
<LI>La hora actual es: <%= new java.util.Date() %>
<LI>
<%
   String dato = request.getParameter("mensaje");
   out.println("El parámetro <CODE>mensaje</CODE> vale:" + dato);
%>
```

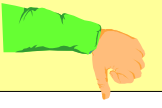
Otra forma de hacer lo mismo

```
<LI>La hora actual es: <%= new java.util.Date() %>
<LI>
<% String dato = request.getParameter("mensaje"); %>
El parámetro <CODE>mensaje</CODE> vale: <%= dato %>
```


Scriptlets en páginas JSP

Se puede generar una página con HTML condicional:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
        <TITLE>Demo JSP</TITLE>
        <LINK REL=STYLESHEET HREF="style.css" TYPE="text/css">
</HEAD>
<BODY>
<H2>Demo JSP</H2>

<% if (request.getParameter("mensaje").equals("hola") { %>
<H1> El parámetro es hola </H1>
<% } else { %>
<H1> El parámetro NO es hola </H1>
<% } %>
</BODY>
</HTML>
```



Declaraciones en páginas JSP

Se incorporan a la clase del *servlet* fuera de cualquier método:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML> <HEAD>
        <TITLE>Demo JSP</TITLE>
        <LINK REL=STYLESHEET HREF="style.css" TYPE="text/css">
</HEAD>
<BODY>
<H2>Demo JSP</H2>

<%! private int accessCount = 0; %>
<H1> Esta página fue accedida <%= ++accessCount %> veces </H1>
</BODY>
</HTML>
```


Directivas en páginas JSP

- **page**: Permite importar clases, extender una clase y definir el *content-type*, entre otros.
- **include**: Permite insertar un archivo en la página al momento de la traducción.
- **taglib**: Permite definir *tags* propios.

Directiva **page**

- Permite extender una clase específica

```
<%@ page extends="package.class" %>
```

- Permite importar clases

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

- Permite definir el valor de ciertos headers

```
<%@ page contentType="MIME-Type" %>
```

- Permite indicar si se utilizan sesiones

```
<%@ page session="true" %>
```


Directiva **include**

- Permite insertar en una página código de otra página al momento de la traducción a *servlet* de la primera.

```
<%@ include file="URL relativa" %>
```

Similar al #include de C

- El resultado es un único *servlet* con el contenido de ambas páginas

Estrategias de implementación

Aplicaciones Simples

- Invocar código Java directamente o indirectamente
- Usar arquitectura MVC + Java beans
- Usar arquitectura MVC + JSP EL (Expression Language)
- Definir y usar nuevos tags.

Aplicaciones Complejas

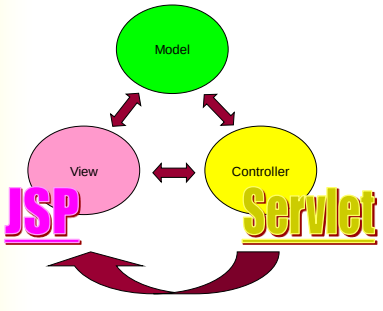
Java Beans

- Deben tener un constructor default.
- No deben tener variables de instancia públicas.
- Deben implementar *getters* y, tal vez, *setters* para todas sus propiedades.

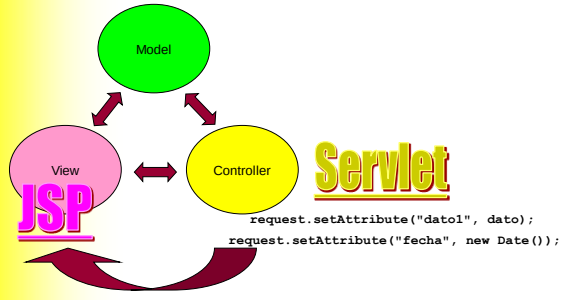
Ejemplo – Clase Person

```
public class Person {
   private String firstName;
   private String lastName;
   public Person() {
   }
   public String getFirstName() {
      return firstName;
   }
   public void setFirstName(String firstName) {
      this.firstName = firstName;
   }
   public String getLastName() {
      return lastName;
   }
   public void setLastName(String lastName) {
      this.lastName = lastName;
   }
}
```


Arquitectura MVC



Arquitectura MVC



Implementación MVC

- Definir los Java Bean que van a representar a los objetos (**Model**)
- Definir los servlets que van a procesar las solicitudes, generar los datos de respuesta y enviárselos a quien los presenta (**Controller**)
- Definir las páginas JSP que van a mostrar la información procesada (**View**)

Ejemplo MVC + Java Beans

- ViewHoroscopePage.java
- viewHoroscope1.jsp

Estrategias de implementación

Aplicaciones Simples

- Invocar código Java directamente o indirectamente
- Usar arquitectura MVC + Java beans
- Usar arquitectura MVC + JSP EL (Expression Language)
- Definir y usar nuevos tags.

Aplicaciones Complejas

JSP Expression Language

- Es un lenguaje que simplifica la forma de acceder a los beans.

`${expresión}`

```
<%@ include file="header.jsp" %>

<H3>Hola ${user.name} </H3>
<H3>El horóscopo para ${horoscope.sign}  es: </H3>
<H4>${horoscope.description} </H4>

<%@ include file="footer.jsp" %>
```


JSP Expression Language

- Acceso a propiedades simples:
  `${objeto.propiedad}`
- Acceso a propiedades de propiedades
  `${objeto.propiedadX.propiedadY}`
- Acceso a colecciones indexadas
  `${objeto.propiedad[index]}`

JSP Expression Language

- Operadores aritméticos: + - * / %
  `${objeto.prop1 + objeto.prop2}`
- Operadores relacionales: == !=
  `${objeto.prop1 == objeto.prop2}`
- Operadores lógicos: && || !
  `${objeto.propiedad[index]}`
- Operador **empty:**

Devuelve true si el objeto es null, string vacio o colección vacia.

Estrategias de implementación

Aplicaciones Simples

Aplicaciones Complejas

- Invocar código Java directamente o indirectamente
- Usar arquitectura MVC + Java beans
- Usar arquitectura MVC + JSP EL (Expression Language)
- Definir y usar nuevos tags.

JSTL

- **J**SP **S**tandard **T**ag **L**ibrary
- Define un conjunto de *tags* para resolver operaciones comunes.
- Evita que un archivo JSP contenga código Java.

JSTL

Ejemplo:

```
<ul>
<%
for(int i=0; i<messages.length; i++) {
   String message = messages[i];
   %>
   <li><%= message %></li>
<% } %>
</ul>
```

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
...
<ul>
<c:forEach var="message" items="${messages}">
   <li><c:out value="${message}" /></li>
</c:forEach>
</ul>
```


JSTL Tags

- JSTL define un conjunto de tags estándar, agrupados en:
  - **core**: control de flujo, variables y salida
  - **xml**: procesamiento de documentos xml
  - **i18n**: internacionalización y formato
  - **database**: acceso a bases de datos
  - **functions**: funciones para manipulación de colecciones y de strings

JSTL Core

- **out**: Imprime en la salida el resultado de una expresión:

```
<h1>Bienvenido <c:out value="${user.name}" /></h1>

<h1>Bienvenido
<c:out value="${user.name}" default="visita" />
</h1>
```

JSTL Core

- **if**: Evalúa su contenido si la condición es verdadera.

```
<c:if test="${user.age >= 18}">
   <p>Usted es mayor de edad.</p>
</c:if>
```

JSTL Core

- **choose**: Evalúa fragmentos de código en base a condiciones mutuamente excluyentes:

```
<c:choose>
 <c:when test="${poll.score == 1}"> MALO </c:when>
 <c:when test="${poll.score == 2}"> REGULAR </c:when>
 <c:when test="${poll.score == 3}"> BUENO </c:when>
 <c:when test="${poll.score == 4}"> MUY BUENO </c:when>
 <c:when test="${poll.score == 5}"> EXCELENTE </c:when>
 <c:otherwise> Valor desconocido </c:otherwise>
</c:choose>
```


JSTL Core

- **forEach**: Permite iterar sobre una colección:

```
<ul>
<c:forEach var="message" items="${messages}">
   <li><c:out value="${message}" /></li>
</c:forEach>
</ul>
```


JSTL Core

- **set**: Permite establecer el valor de una variable:

```
<c:set var="inc" value="${miVariable + 1}" />
El valor es: <c:out value="${inc}" />

<c:set var="cantAccesos"
    value="${cantAccesos + 1}" scope="session" />
Cantidad de acccesos:
       <c:out value="${cantAccesos}" />
```


JSTL Core

- **url**: Permite generar una URL, opcionalmente agregando parámetros.

```
<c:url var="url" value="/catalog" >
    <c:param name="id" value="${book.id}" />
</c:url>
<c:out value="${url}" />

<c:url value="/catalog" >
    <c:param name="id" value="${book.id}" />
</c:url>
```


Ventajas de utilizar JSTL

- Evita incluir código Java en archivos JSP.
- Simplifica operaciones comunes.
- Permite incluir expresiones EL.
- Los JSP resultan más sencillos de entender.

Tag Library

- La API de JSP permite definir tags propios.

TAG HANDLER CLASS

- Es una clase de Java que implementa la lógica de un tag.
- Debe implementar la interfaz `javax.servlet.jsp.tagext.SimpleTag`

TAG LIBRARY DESCRIPTOR FILE (TLD)

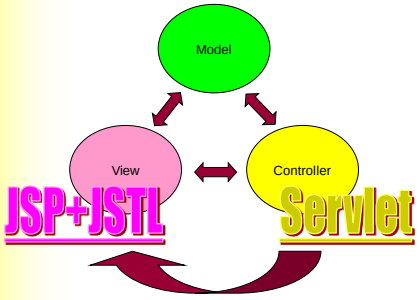
- Es un archivo XML que define los nombres y atributos de los tags, y especifica la clase handler que debe utilizarse para cada uno.
- Se debe ubicar en el directorio WEB-INF.

Tag Library

Ejemplo:

```
<?xml version="1.0" encoding="UTF-8" ?>

<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
        http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
    version="2.0">

  <tlib-version>1.1</tlib-version>
  <short-name>c</short-name>
  <uri>http://java.sun.com/jsp/jstl/core</uri>

  <tag>
    <description> Like &lt;%= ... &gt;, but for expressions. </description>
    <name>out</name>
    <tag-class> org.apache.taglibs.standard.tag.rt.core.OutTag </tag-class>
    ...
  </tag>

</taglib>
```


Arquitectura MVC




```
request.getRequestDispatcher("view.jsp")
       .forward(request, response);
```


Ejemplo MVC con JSTL

- ViewHoroscope2.java
- viewHoroscope2.jsp