

Java Server Pages (JSP)

Introducción a los Conceptos Básicos

Contenido

- ▶ Subtemas 
- ▶ ¿Dónde encaja JSP dentro de la arquitectura JEE? 
- ▶ ¿Qué es una página JSP? 
- ▶ Servlet vs. JSP 
- ▶ Arquitectura JSP 
- ▶ Ciclo de vida de una página JSP 
- ▶ Aplicaciones Web basadas en JSP 

Contenido

- ▶ JSP “es” Servlet 
- ▶ Técnicas de generación de contenido dinámico 
- ▶ Invocando código Java
- ▶ Incluyendo y reenviando a otros recursos Webs 
- ▶ Directivas 
- ▶ Objetos implícitos 
- ▶ Ámbito de Objetos 
- ▶ Componentes JavaBeans para JSP 
- ▶ Manejo de Errores 

Subtemas de JSP

» Elementos a estudiar

Subtemas

- ▶ JSP se puede dividir en 4 temas
 1. Lo **básico** de JSP
 2. Como los **JavaBeans** se utilizan en JSP
 3. Los **Tags personalizados** (custom tags)
 - Son un tipo especial de componentes Java orientados a la tecnología JSP
 4. Librería estándar de tags de JSP (*JSP Standard Tag Library*)
 - es una colección estándar de tags personalizados
- ▶ En esta presentación nos centraremos en los dos primeros temas
 - Los siguientes dos temas se desarrollarán más adelante

Temas avanzados

- ▶ Existen multitud de Frameworks para el desarrollo de aplicaciones Web
 - basados en la tecnología JSP y Servlet
 - siguen el patrón CVM
- ▶ El patrón o arquitectura CVM
 - El Controlador Vista Modelo es la mejor arquitectura para la capa web en términos de
 - reusabilidad de código
 - robustez de diseño
 - implementación y mantenimiento
 - El patrón no es más que la suma de las mejores prácticas (*best practice*)

Temas avanzados

- ▶ Entre los patrones de CVM la arquitectura **Model2** es el framework que ofrece la mejor
 - reusabilidad de código
 - mantenimiento
- ▶ En **Model2**
 - Un **Servlet** actúa como Controlador o distribuidor
 - Las **páginas JSP** se utilizan para contener las Vistas

Frameworks

- ▶ Hay varios frameworks basados en la arquitectura **Model2**
 - **Apache Struts**
 - El más popular
 - **Java Server Faces**
 - Es una iniciativa de Java para esta arquitectura
 - Existen otros como
 - **Echo**
 - **Tapastry**
- ▶ Estudiaremos Struts en presentaciones posteriores

Agenda

- ▶ ¿Dónde encaja JSP en la arquitectura JEE?
- ▶ Introducción a como funciona JSP
- ▶ El ciclo de vida de las páginas JSP
- ▶ Como desarrollar aplicaciones Web basadas en JSP
- ▶ La anatomía de una página JSP
- ▶ Como utilizar JavaBeans en una página JSP
- ▶ Como manejar errores en páginas JSP

¿Dónde encaja JSP dentro de la arquitectura JEE?

» JSP en el mundo de JEE

JSP y Servlet en JEE

- ▶ Un **Servlet** es un programa que extiende la funcionalidad de un servidor web
 - generando contenido dinámico e interactuando con clientes webs mediante el **Paradigma Solicitud-Respuesta**
 - *request-response paradigm*

JSP y Servlet en JEE

- ▶ **JSP** es un tecnología web extensible que utiliza
 - una plantilla de datos o datos estáticos (*template data*)
 - normalmente elementos HTML o XML
 - lenguajes de script
 - elementos a medida
 - objetos Java del lado servidor para devolver contenido dinámico al cliente

Contenido Estático y Dinámico

- ▶ La tecnología JSP y Servlet está diseñada para la generación de contenido web dinámico
 - no estático
- ▶ ¿Qué es contenido estático y dinámico?

Contenido Estático y Dinámico

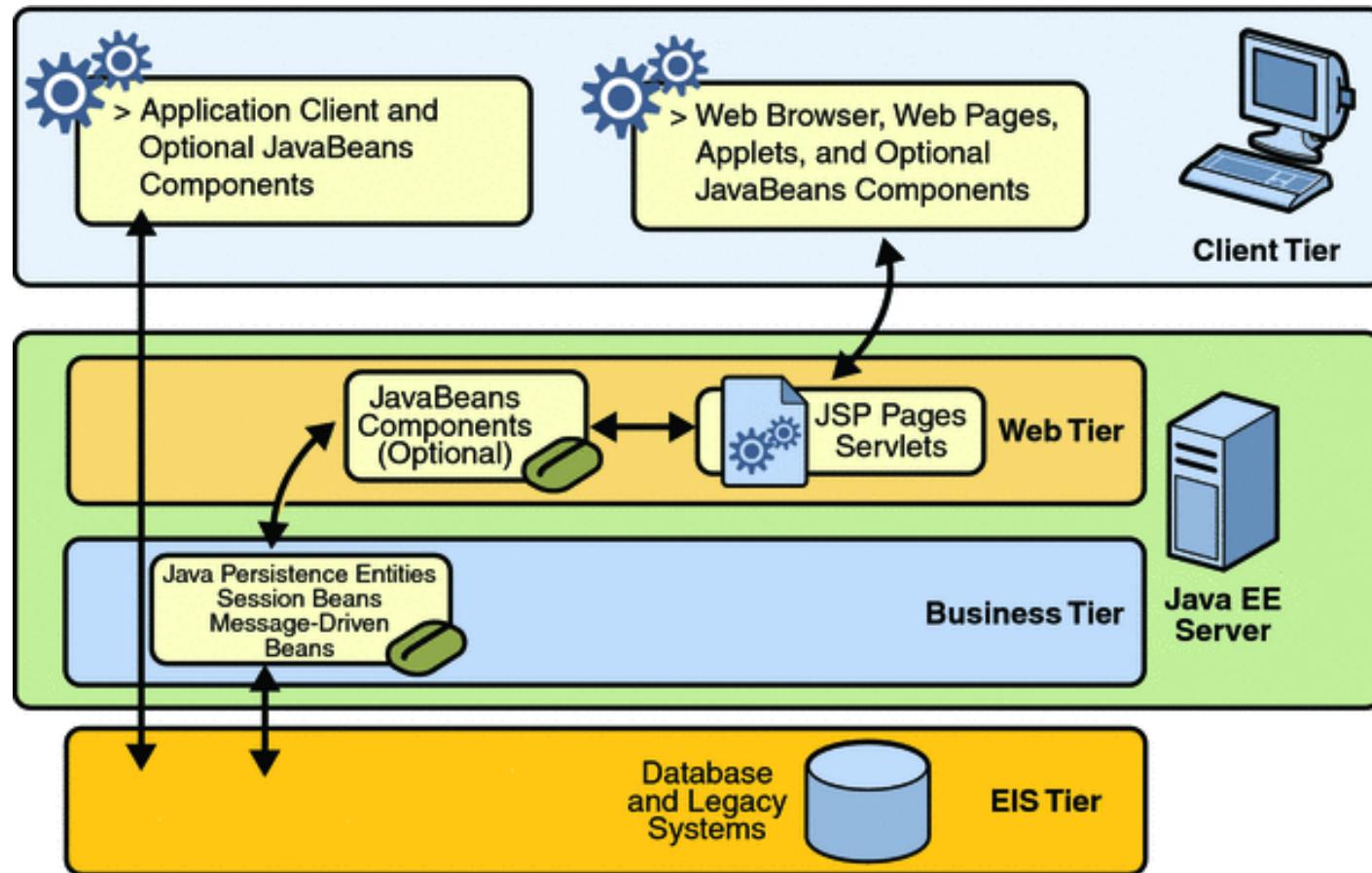
▶ Contenido Estático

- Páginas HTML cuyo contenido NO cambia independientemente de las condiciones externas
 - Muestra la misma información una y otra vez

▶ Contenido Dinámico

- Se genera dinámicamente basándose en condiciones externas
 - Por ejemplo, mediante valores introducidos por el usuario a través de un formulario web
 - Un servidor Web utiliza tecnologías como JSP y Servlet para la generación dinámica de contenidos
- ▶ Un aplicación Web hace uso de contenido Estático y Dinámico

JSP y Servlet en JEE



¿Qué es una página JSP?

» Primera aproximación

¿Qué es una página JSP?

- ▶ Es un **documento basado en texto** capaz de devolver tanto contenido estático como dinámico a un cliente Web

¿Qué es una página JSP?

- ▶ El contenido estático y el dinámico pueden entremezclarse
 - El contenido estático puede estar compuesto por tags HTML, XML o texto plano
 - El contenido dinámico puede generarse desde código Java, mostrar propiedades de una JavaBean o tras invocar lógica de negocio definida en tags personalizados
- ▶ Una página JSP está compuesta por **plantillas estáticas** en donde
 - código Java, JavaBeans y tags personalizados se utilizan para la generación dinámica de contenido

Ejemplo: ‘Hola mundo!’ en JSP

- ▶ En azul aparece marcado el contenido estático
 - tags HTML
- ▶ En rojo el contenido dinámico
 - es el único contenido de la página que cambiara dinámicamente en función de cuando se realice la carga de la página

```
<html>
  <body>
    Hola mundo!<br>Hoy es <%= new java.util.Date() %>
  </body>
</html>
```

Servlet vs. JSP

» Comparación entre Servlet y JSP

Servlet vs. JSP

- ▶ Vamos a comparar a groso modo las dos tecnologías
- ▶ En los **Servlet** la generación de código HTML se realiza directamente en código Java
 - En **JSP** las páginas HTML se representan mediante tags HTML
 - En **JSP** se trabaja directamente sobre la página HTML mediante tags específicos
- ▶ En **JSP** es sencillo que diseñadores Web trabajan sobre el código
 - no tienen que conocer código Java, se limitan a la parte estática y la parte visual (por ejemplo, css)
- ▶ En **Servlet**, esto es mucho mas complicado, no existe una separación clara

Servlet vs. JSP

- ▶ Existe una relación directa subyacente entre Servlet y JSP
 - Una página JSP se traduce internamente en código Servlet cuando se despliega en un contenedor de Servlet
- ▶ Hablaremos de esta hecho más adelante

Beneficios del uso de JSP

- ▶ Bajo la arquitectura JSP
 - Contenido y presentación están separados
- ▶ El desarrollo de aplicaciones Web se simplifica
 - la lógica de contenido (negocio) se realiza mediante JavaBeans o tags personalizados
 - la lógica de presentación se realiza mediante tags HTML y similares
- ▶ Reutilización de componentes
 - La lógica de negocio se realiza mediante componentes que pueden ser reutilizados

Beneficios del uso de JSP

- ▶ JSP trabaja bajo múltiples plataformas (como Servlets)
 - Sobre un contenedor de Servlets
- ▶ Permite el despliegue automático
 - las páginas JSP se recompilan automáticamente en Servlets cuando se realizan cambios sobre estas
- ▶ Facilita el trabajo de los diseñadores Web

¿JSP en vez de Servlet?

- ▶ ¿Es compatible el uso de ambos?
- ▶ No, es interesante mantener el uso de ambos para así poder beneficiarse de los puntos fuertes de ambas tecnologías
 - El punto fuerte de los Servlets
 - Controlar y entregar
 - El punto fuerte de JSP
 - Presentación
- ▶ Se han diseñado con propósitos bien diferenciados

¿JSP en vez de Servlet?

- ▶ En un entorno típico de producción, ambos Servlet y JSP se utilizan en el patrón CVM
 - **Servlet** realiza la parte de **Controlador**
 - **JSP** se encarga de gestionar las **Vistas**

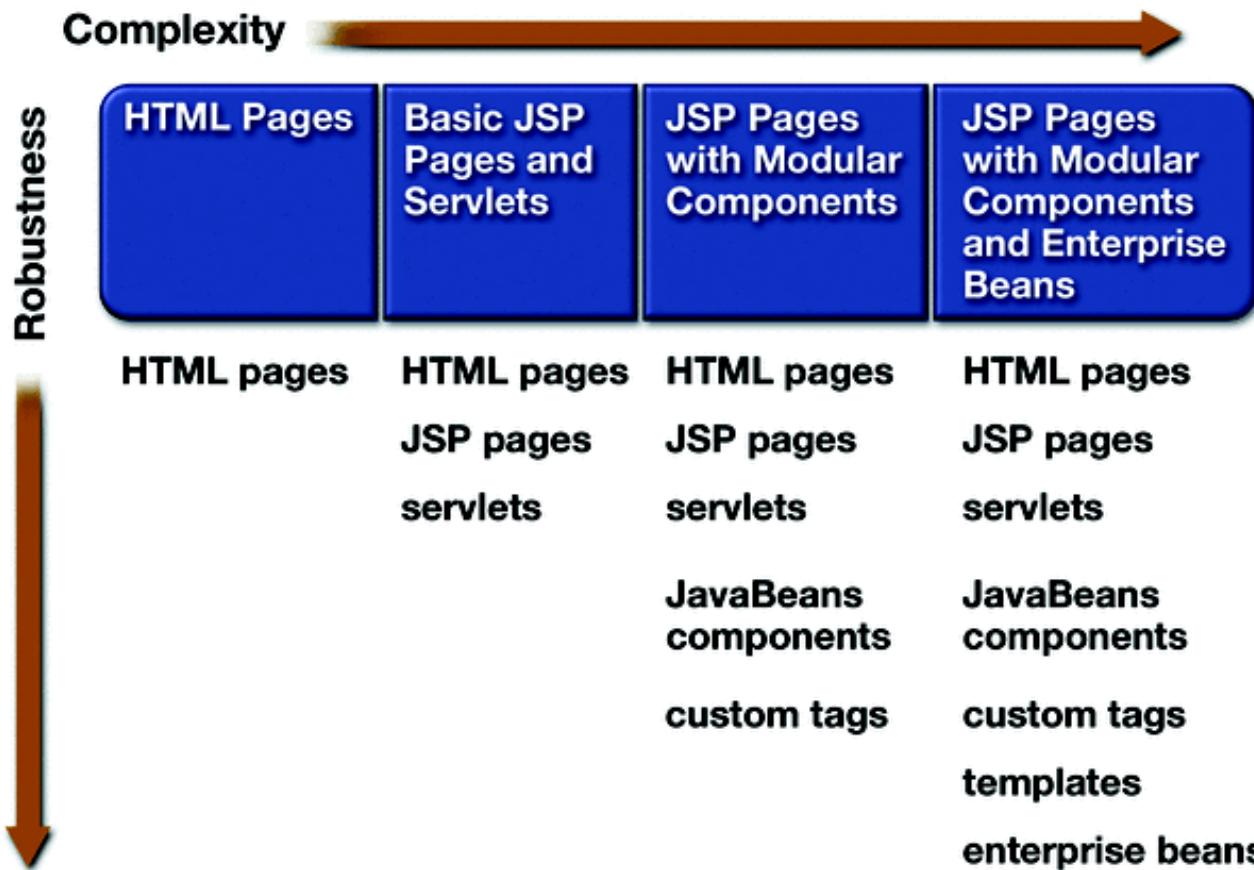
¿JSP en vez de Servlet?

- ▶ **Servlet, controlar y entregar**
 - Gracias a su forma de programa Java su punto fuerte se centra en el control y la entrega
- ▶ Por ejemplo, se quiere mostrar contenido en función de quien es el llamador o un parámetro de entrada
 - Es sencillo expresar la lógica como condición de un programa java (**controlador**)
 - Una vez hecho esto es sencillo elegir que página mostrar (**entregar**)
- ▶ JSP se encargará de mostrar el contenido mediante páginas HTML o XML

Arquitectura JSP

»» Organización y estructura

Diseño de una Aplicación Web



Separación Solicitud/Presentación

```
public class ServletPedidos...{
    public void doGet(...) {
        if(isPedidoValido(req)) {
            salvarPedido(req);

            out.println("<html>");
            out.println("<body>");
            ...
        }
    }
}
```

```
private void isPedidoValido(...){
    ...
}

private void salvarPedido(...){
    ...
}
```

Servlet completo

Procesamiento de Solicitud

Presentación

Lógica de Negocio

```
public class ServletPedidos...{
    public void doGet(...) {
        ...
        if(bean.isPedidoValido(...)) {
            bean.salvarPedido(...);

            forward("conf.jsp");
        }
    }
}
```

Servlet

```
<html>
<body>
    <ora:loop name="pedido">
        ...
    </ora:loop>
</body>
</html>
```

JSP

Java Server Pages (JSP)

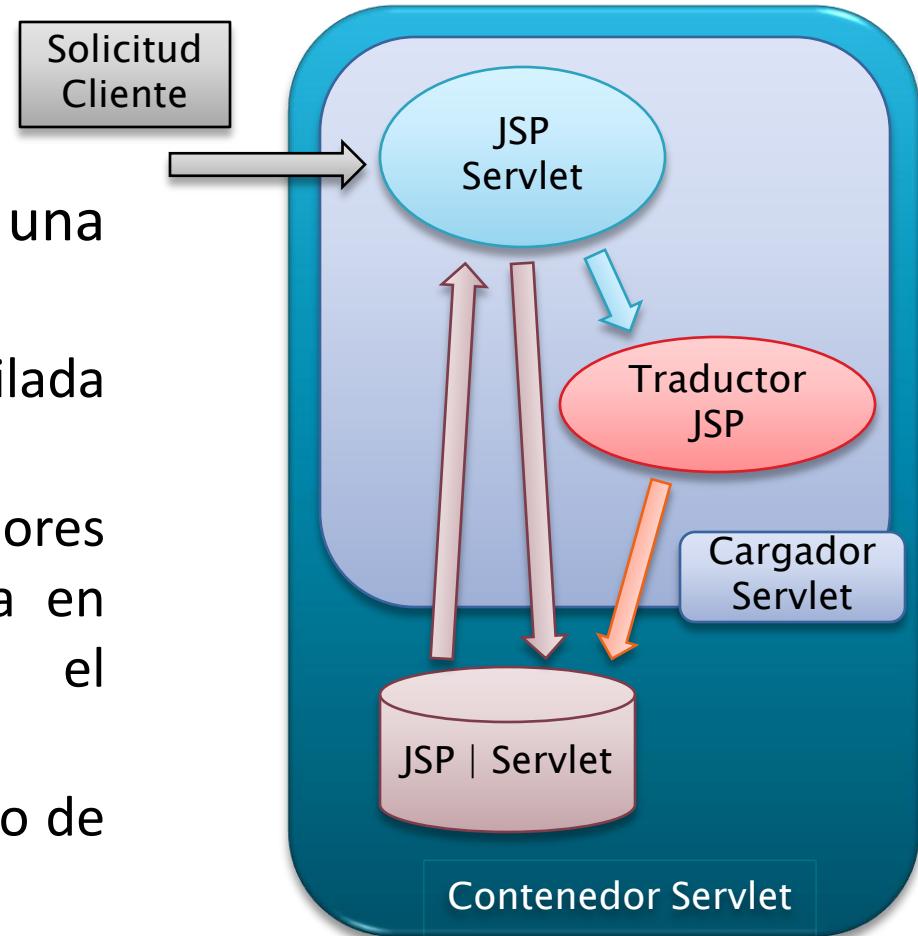
isPedidoValido()

salvarPedido()

JavaBean

Arquitectura JSP

- ▶ Cuando un Cliente envía una solicitud HTTP a una página JSP
 - La página es traducida y compilada en una Clase Servlet
 - O en la mayoría de los Contenedores Web, la página JSP se compila en una Clase Servlet durante el despliegue.
 - Una vez compilada, sigue en ciclo de vida de los Servlet



Ciclo de vida de una página JSP

»» Life-Cycle tras la compilación en su correspondiente Servlet

Como funciona JSP

- ▶ Cuando una solicitud HTTP se mapea a su correspondiente página JSP
 - un Servlet especial del contenedor se encarga de comprobar si la página JSP ha cambiado con respecto a su versión Servlet
 - si es así, la traduce y compila otra vez
 - entrega la solicitud a dicho Servlet compilado para que la procese
 - Esto se realiza de manera transparente y automática por el contenedor

Fases del Ciclo de Vida

- ▶ Se divide en 3 fases

- 1) Fase de traducción

- donde la página JSP se traduce en un código Servlet

- 2) Fase de compilación

- donde este código Servlet se compila

- 3) Fase de ejecución

- donde la instancia de este Servlet se crea para atender solicitudes HTTP

- Normalmente la fase de traducción y compilación ocurren a la vez

Fases de traducción/Compilación

- ▶ En esta fase si la página JSP no dispone de Clase compilada se traduce y compila. ¿Cuando?
 - Si la página JSP es desplegada por primera vez
 - Si es modificada
 - ▶ Esto es por lo que a veces se observan ciertos tiempos de espera tras realizar una solicitud por primera vez
-
- La mayoría de los contenedores realizan siempre este proceso en la fase de despliegue*

Fases de traducción/Compilación

- ▶ Por ejemplo, una página JSP llamada **miPagina.jsp** tiene el siguiente código fuente generado
 - **miPagina\$jsp.java**
- ▶ Esta página se almacena en la raíz del contexto
- ▶ Esta Clase se compila obteniendo su correspondiente .class

Fase de Ejecución

- ▶ Una vez traducida y compilada sigue el siguiente ciclo de vida
 1. El contenedor **crear** una instancia de la nueva Clase Servlet
 2. El contenedor **inicializa** la instancia llamando al método `jsplnIt`
 3. El contenedor invoca el método **jspService()** pasando el objeto solicitud (request) y respuesta (response)
- ▶ Si el contendor necesita eliminar la instancia invoca el método **jspDestroy()**

Inicialización de una página JSP

- ▶ Se puede personalizar el proceso de inicialización para
 - Leer datos de configuración de persistencia (BBDD)
 - Inicializar recursos
 - Realizar cualquier otra actividad que tenga que realizarse una vez
- ▶ Para ello hay que sobrescribir el método **jsplInit()** de la interface JspPage

Finalización de una página JSP

- ▶ Sobrescribiendo el método `jspDestroy()` se pueden liberar los recursos obtenidos en el proceso de inicialización

Ejemplo

```
<%@ page import="database.*" %>
<%@ page errorPage="errorpage.jsp" %>
<%-- Declare initialization and finalization methods using JSP declaration --%>
<%!

private BookDBAO bookDBAO;

public void jspInit() {

    // retrieve database access object, which was set once per web application
    bookDBAO =
        (BookDBAO) getServletContext().getAttribute("bookDB");
    if (bookDBAO == null)
        System.out.println("Couldn't get database.");
}

public void jspDestroy() {
    bookDBAO = null;
}
%>
```

Aplicaciones Web basadas en JSP



Pasos para el desarrollo

Pasos para el Desarrollo

1. **Escribir (y compilar)** el código del componente Web (Servlet y/o JSP)
 - y las clases de ayuda referenciadas por estos componentes
2. Crea cualquier **recurso estático** necesario
 - por ejemplo, imágenes o páginas HTML
3. Crear el **descriptor de despliegue** (web.xml)
4. **Construye** la aplicación web
 - crear el fichero *.war
5. Instala o **despliega** el *.war en un contenedor Web
6. Acceso a la aplicación desde el navegador Web

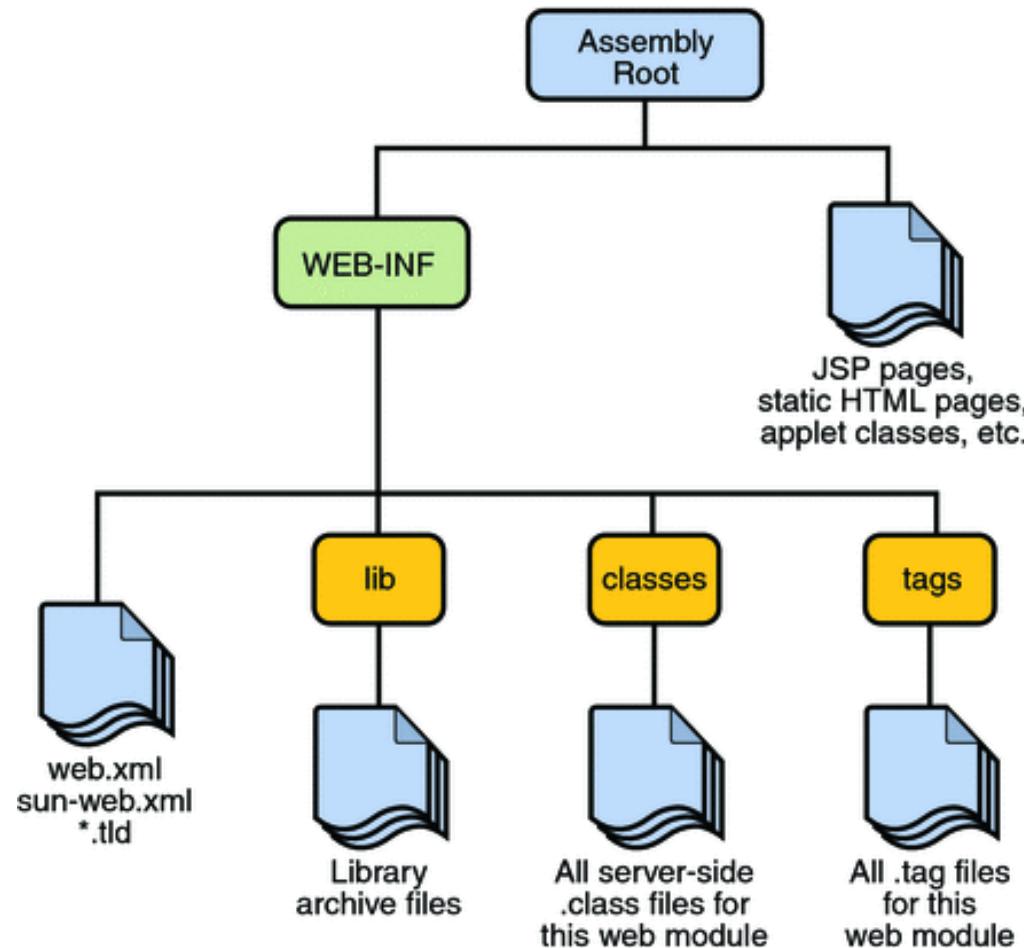
1. Escribe y compila

- ▶ Crea la estructura del módulo Web que contendrá todo lo necesario para la aplicación Web
 - existe una estructura recomendada para aplicaciones Web
- ▶ Escribe el código Servlet y/o las páginas JSP junto con las Clases de ayuda
- ▶ Crea el fichero build.xml
 - La herramienta ANT permite construir aplicaciones mediante un fichero build.xml
 - ANT es una utilida make independiente de la plantaforma

Estructura del Módulo Web

- ▶ Se define la siguiente estructura para todo módulo o aplicación web
- ▶ El directorio raíz contiene un directorio llamado **WEB-INF** que contiene
 - **web.xml**: El descriptor de despliegue de la aplicación web
 - Descriptores de Librerías Tag (Tag Library Descriptors)
 - **classes**: Un directorio que contiene todas las Clases Java del lado servidor
 - Servlet, Clases de utilidad y componentes JavaBeans
 - **tags**: directorio que contiene ficheros tags que son implementaciones de librerías tag
 - **lib**: Un directorio conteniendo las librerías (JAR) específicas utilizadas y necesarias en el lado servidor

Estructura del Módulo Web



2. Creación de Recursos Estáticos

- ▶ El siguiente paso es la creación de los recurso estáticos utilizados por los componentes web
- ▶ Como recursos estáticos
 - Páginas HTML
 - De error
 - De bienvenida
 - Ficheros de Imagen utilizadas por las páginas HTML o JSP

3. Crear el descriptor de despliegue

- ▶ El descriptor de fichero se llama **web.xml**
- ▶ Este **web.xml** contiene las instrucciones de despliegue y de ejecución para el contenedor Web
 - Por ejemplo, contiene los identificadores que los clientes utilizan para acceder a los componentes webs
- Este fichero es un requisito imprescindible para TODA aplicación Web

Ejemplo de web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app PUBLIC '-//Sun Microsystems, Inc.//DTD Web Application
 2.3//EN' 'http://java.sun.com/dtd/web-app_2_3.dtd'>

<web-app>
  <display-name>Hello2</display-name>
  <description>Web Application description</description>
  <servlet>
    <servlet-name>greeting</servlet-name>
    <display-name>greeting</display-name>
    <description>no description</description>
    <jsp-file>/greeting.jsp</jsp-file>  <!-- que es lo que se llama -->
  </servlet>
  <servlet-mapping>
    <servlet-name>greeting</servlet-name>
    <url-pattern>/greeting</url-pattern>  <!-- URL para el navegador -->
  </servlet-mapping>
</web-app>
```

Ejemplo de web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app P
  2.3//EN' 'http://
                               osystems, Inc./DTD Web Application
                               eb-app_2_3.dtd'>
<web-app>
  <display-name>Hello2</display-name>
  <description>Web Application description</description>
  <servlet>
    <servlet-name>greeting</servlet-name>
    <display-name>greeting</display-name>
    <description>no description</description>
    <jsp-file>/greeting.jsp</jsp-file>  <!-- que es lo que se llama -->
  </servlet>
  <servlet-mapping>
    <servlet-name>greeting</servlet-name>
    <url-pattern>/greeting</url-pattern>  <!-- URL para el navegador -->
  </servlet-mapping>
</web-app>
```

Inicio de la definición de una Aplicación Web

Nombre de la Aplicación

Ejemplo de web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-
  2.3//EN' 'ht
  Microsystems, Inc.//DTD Web Application
  /dtd/web-app_2_3.dtd'>
<web-app>
  <display-name>Hello2</display-name>
  <description>Web Application description</description>
  <servlet>
    <servlet-name>greeting</servlet-name>
    <display-name>greeting</display-name>
    <description>no description</description>
    <jsp-file>/greeting.jsp</jsp-file> <!-- que es lo que se llama -->
  </servlet>
  <servlet-mapping>
    <servlet-name>greeting</servlet-name>
    <url-pattern>/greeting</url-pattern> <!-- URL para el navegador -->
  </servlet-mapping>
</web-app>
```

**Componentes Web
(Servlet/JSP)**

Nombre del componente

Componente JSP

Ejemplo de web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app PUBLIC '-//Sun Microsystems, Inc.//DTD Web Application
 2.3//EN' 'http://java.sun.com/dtd/web-app_2_3.dtd'>

<web-app>
  <display-name>Hello2</display-name>
  <description>Web Appli<!-- que es lo que se llama -->
  <servlet>
    <servlet-name>greeting</servlet-name>
    <display-name>greeting</display-name>
    <description>no description</description>
    <jsp-file>/greeting.jsp</jsp-file>      <!-- URL para el navegador -->
  </servlet>
  <servlet-mapping>
    <servlet-name>greeting</servlet-name>
    <url-pattern>/greeting</url-pattern>
  </servlet-mapping>
</web-app>
```

Especificación del Mapeo (asociación componente – URL)

URL para el Navegador

Ejemplo de web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app PUBLIC '-//Sun Microsystems, Inc.//DTD Web Application
 2.3//EN' 'http://java.sun.com/dtd/web-app_2_3.dtd'>

<web-app>
  <display-name>Hello2</display-name>
  <description>Web Application description</description>
  <servlet>
    <servlet-name>greeting</servlet-name>
    <display-name>greeting</display-name>
    <description>no description</description>
    <jsp-file>/greeting.jsp</jsp-file>      <!-- que es lo
  </servlet>
  <servlet-class> GreetingServlet</servlet-class>
    <service-name>greeting </service-name>
      <url-pattern>/greeting</url-pattern>  <!-- URL para el navegador -->
    </servlet-mapping>
  </web-app>
```

Componente JSP

Componente Servlet

4. Construcción de la Aplicación Web

- ▶ Una aplicación web es un paquete desplegable (*deployable package*) que contiene todo lo necesario para el despliegue
- ▶ Esta en la forma de **fichero *.war** o en la forma desempaquetada del fichero *.war
 - con la misma estructura y contenido que el fichero
- ▶ Una vez disponemos de todo lo necesario
 - hay que colocar cada uno de los elementos en su localización correspondiente dentro de la estructura

5. Despliegue de la Aplicación Web

▶ Método manual

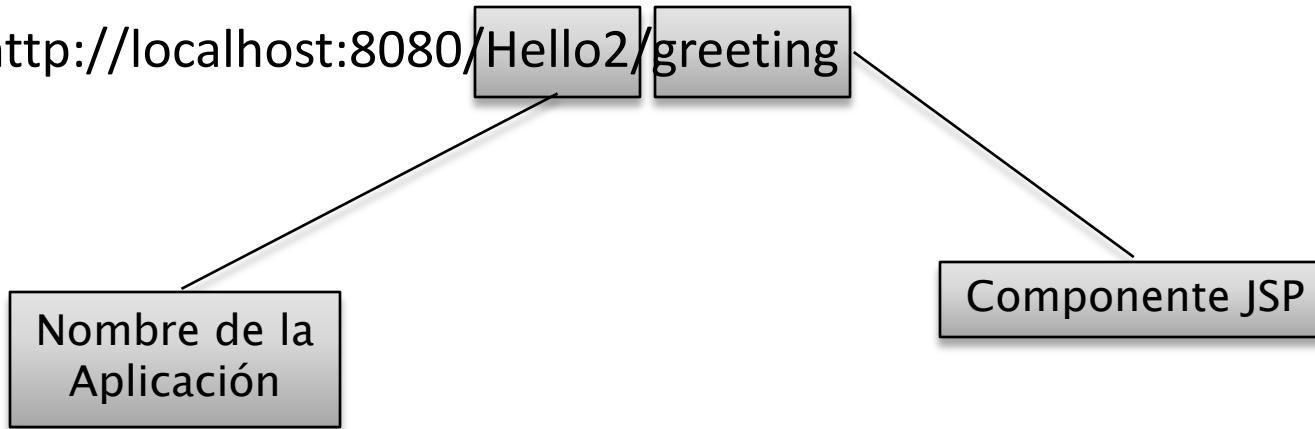
- Copiar el **fichero *.war** o el directorio desempaquetado en el directorio **deploy** del contenedor de Servlet

6. Acceso desde el Navegador Web

- ▶ Una vez la aplicación ha sido desplegada correctamente
 - desde el navegador vamos a la URL de la aplicación web
 - Por ejemplo
 - <http://localhost:8080>Hello2/greeting>

6. Acceso desde el Navegador Web

- ▶ Una vez la aplicación ha sido desplegada correctamente
 - desde el navegador vamos a la URL de la aplicación web
 - Por ejemplo
 - `http://localhost:8080>Hello2/greeting`



JSP “es” Servlet

» Una relación directa

JSP “es” Servlet

- ▶ Es importante conocer la relación directa que existe entre JSP y Servlet
 - incluso se puede decir que JSP “es” un Servlet
- ▶ Como ya hemos comentado
 - JSP provee una separación entre la generación de contenido y la lógica de presentación
 - Mientras, mantiene los beneficios de los Servlet

JSP “es” Servlet

- ▶ El proceso en más detalle es el siguiente
 1. La página JSP se traduce directamente en una **Clase Servlet** cuando se despliega
 - TomCat traduce `greeting.jsp` en `greeting$jsp.java`
 2. El código Java de la página JSP se inserta directamente en el método **`jspService()`** de la Clase Servlet resultante si ningún tipo de modificación
 3. Los **Objetos implícitos** disponibles a todo Servlet también están disponibles en la página JSP
 - Por ejemplo, los Objetos HttpSession o ServletContext

Técnicas de Generación de Contenido Dinámico

» en JSP

Generación de Contenido Dinámico

- ▶ Se pueden utilizar varias técnicas dependiendo de los siguientes factores
 - tamaño y complejidad del proyecto
 - requisitos de reusabilidad del código, mantenibilidad y grado de robustez
- ▶ Existen de las más simples a las más complejas

Generación de Contenido Dinámico

- ▶ Técnicas de generación de código dinámico
 - 1) Invocar código Java directamente desde la página JSP
 - 2) Invocar código Java indirectamente desde la página JSP
 - 3) Hacer uso de JavaBeans desde la página JSP
 - 4) Desarrollar e utilizar tus propios tags personalizados
 - 5) Aprovechar las librerías de terceros de tags personalizados o la JSTL (JSP Estándar Tag Library)
 - 6) Hacer uso de frameworks que siguen el patrón CVM

1) Invocar código Java directamente

- ▶ Es la técnica menos sofisticada
- ▶ Se coloca código Java directamente en la página JSP
- ▶ La lógica de negocio (código Java) y la lógica de presentación están mezcladas
 - ofrece una mala separación entre las dos
- ▶ Apropiado SOLO para proyecto simples
 - es difícil de mantener
 - difícil de reutilizar su código
- ▶ En definitiva, mala separación entre la generación de contenido y la presentación

2) Invocar código Java indirectamente

- ▶ Otra manera es desarrollando Clases de utilidad
- ▶ Se insertar en la página JSP SOLO el código Java necesario para invocar estás Clases de Utilidad
- ▶ Las Clases de Utilidad se mantienen fuera de las páginas JSP
- ▶ Esto resulta en
 - mejor separación de la generación de contenido de la lógica de presentación
 - mejor reusabilidad y mantenibilidad

3) Uso de JavaBeans

- ▶ Desarrollar Clases de Utilidad en la forma de JavaBeans
- ▶ El uso de JavaBeans en las páginas JSP tienen la ventaja con respecto a los dos métodos anteriores
 - JSP soporta de manera específica el uso de JavaBeans
 - facilita la creación de instancias de los JavaBeans
 - el acceso a las propiedades de los JavaBeans es más sencillo
 - mediante los getters y setters
- ▶ Por lo tanto, el acceso desde la página JSP a los JavaBeans es mucho más sencillo
- ▶ Aumenta la reusabilidad y mantenibilidad

4) Desarrollo y Uso de Tags Personalizados

- ▶ Desarrollar sofisticados componentes llamados Tags Personalizados
 - están especialmente diseñados para JSP
- ▶ Mas potentes que los componentes JavaBeans
 - mucho más que métodos getter y setters
- ▶ Ofrece mayor reusabilidad, mantenibilidad y robustez
- ▶ El desarrollo de tags personalizados es mucho más complicado que crear JavaBeans

5) Uso de Tags Personalizados y JSTL

- ▶ Existen multitud de tags personalizados
 - de proyectos libres y comerciales
 - Por ejemplo, Apache Struts
- ▶ JSTL (JSP Standard Tag Library) es un conjunto de tags personalizados disponible por la plataforma Java EE

6) Uso del Patrón CVM

- ▶ La técnica más sofisticado es mediante el uso de frameworks basados en el patrón de CVM
 - La vista se genera mediante JSP
 - El controlador mediante Servlet
- ▶ Lo recomendado es hacer uso de uno de los frameworks existentes
 - Apache Struts
 - JavaServer Faces

Invocando Código Java

» con Elementos de Script de JSP

Elementos de Script de JSP

- ▶ Permiten la inserción de código Java directamente en una página JSP
 - por lo tanto, será insertado finalmente en el código del Servlet generado tras la compilación de la página JSP
- ▶ La práctica recomendada es minimizar al máximo el uso de elementos de script si es posible
- ▶ Existen 3 formas
 - **Expresiones**: <%= Expresión %>
 - **Scriptlets**: <% Code %>
 - **Declaraciones**: <%! Declaraciones %>

1. Expresiones

- ▶ Durante la fase de Ejecución
 - La expresión se evalúa y convierte en una ristra
 - La ristra se inserta directamente en el Objeto de salida
 - sería algo como `out.println(expresion)` si lo hicéramos con un Servlet
- ▶ Se pueden utilizar Objetos predefinidos (implícitos) en las expresiones
- ▶ Formato
 - `<%=` Expresión `%>` (en JSP 1.1)
 - `<jsp:expresion>` Expresión `</jsp:expresion>` (en JSP 1.2)

1. Expresiones: Ejemplos

- ▶ Mostrar la fecha actual mediante la Clase Date
 - Fecha actual: <%= new java.util.Date() %>
- ▶ Mostrar un numero aleatorio mediante la Clase Math
 - Número aleatorio: <% Math.random() %>

1. Expresiones: Ejemplos

- ▶ Uso de Objetos implícitos como
 - **request**, de la Clase HttpServletRequest
 - **application**, de la Clase ServletContext
 - **session**, de la Clase HttpSession
- ▶ Por ejemplo
 - **El hostname**: <%= request.getRemoteHost() %>
 - **Tu parámetro**: <%= request.getParameter("tuParametro") %>
 - **Servidor**: <%= application.getServerInfo() %>
 - **ID de Sesión**: <%= session.getId() %>

2. Scriptlets

- ▶ Se utilizan para insertar código Java
 - este código se transformará en sentencias de Java y se insertará en el método `jspService()` del Servlet correspondiente a la página JSP
- ▶ Una variable declarada en un scriptlet es accesible desde cualquier sitio de la página JSP

2. Scriptlets

- ▶ Pueden hacer cosas que no pueden hacer las Expresiones
 - dar valor a la cabecera de respuesta y códigos de estado HTTP
 - escribir en log del servidor
 - actualizaciones de BBDD
 - ejecutar código con estructuras de control (bucles, if-else, etc)
- ▶ Pueden hacer uso de los Objetos implícitos
- ▶ Formato
 - <% código Java %> (en JSP 1.1)
 - <jsp:scriptlet> código Java </jsp:scriptlet> (en JSP 1.2)

2. Scriptlets: Ejemplos

- ▶ Mostrar la ristra de solicitud (query string)

```
<%  
    String queryData = request.getqueryString();  
    out.println("Attached GET data: " + queryData);  
%>
```

- ▶ Especificar el tipo de respuesta

```
<% response.setContentType("text/plain"); %>
```

2. Scriptlets: Ejemplo con bucle

```
<%
Iterator i = cart.getItems().iterator();
while (i.hasNext()) {
    ShoppingCartItem item = (ShoppingCartItem)i.next();
    BookDetails bd = (BookDetails)item.getItem();
%>
<tr>
    <td align="right" bgcolor="#ffffff">
        <%=item.getQuantity()%>
    </td>
    <td bgcolor="#ffffaa">
        <strong>
            <a href="<%=request.getContextPath()%>/bookdetails?bookId=<%=bd.getBookId()%>">
                <b><%=bd.getTitle()%></b>
            </a>
        </strong>
    </td>
    ...
<%
// End of while
%
%>
```

The diagram illustrates the flow of data from JSP scriptlets to UI components. It shows three boxes: a blue box labeled "Ruta del Contexto" pointing to the href attribute of the anchor tag; an orange box labeled "Título del libro" pointing to the bd.getTitle() method call; and a purple box labeled "Id del Libro" pointing to the bd.getBookId() method call.

Fragmentos de una Página JSP

- ▶ Veamos en que se traducen los siguiente fragmentos de una página JSP
 - <h2>HTML</h2>
 - <%= Expression %>
 - <% scriptletCode(); %>

Fragmentos de una Página JSP

```
public void _jspService(HttpServletRequest request,
                        HttpServletResponse response) throws ServletException, IOException {
    response.setContentType("text/html");
    HttpSession session = request.getSession(true);
    JSPWriter out = response.getWriter();

    // El fragmento estático HTML se envía al output stream tal cual
    out.println("<H2>sangHTML</H2>");

    // La expresión se convierte en un String y se envia por al output stream
    out.println(Expression());

    // Scriptlet se inserta como código Java
    scriptletCode();
    ...
}
```

3. Declaraciones

- ▶ Se utilizan para definir variables o métodos que serán insertadas en el cuerpo principal de la Clase Servlet resultante
 - fuera del método `_jspService()`
 - Los Objetos implícitos (como HttpSession) no son accesibles desde la declaraciones
- ▶ Normalmente son utilizadas por expresiones y scriptlets

3. Declaraciones

- ▶ Uno de los uso de las declaraciones es para la inicialización y limpieza de la página JSP
 - para sobrescribir los métodos jsplnIt() y/o jspDestroy()
- ▶ El formato:
 - **<%!** código de declaración de un método o variable **%>**
 - **<jsp:declaration>** declaración método/variable **</jsp:declaration>**

3. Declaraciones: Ejemplo

```
<H1>Alguna Cabecera</H1>
```

```
<%!
```

```
private String cabeceraRandom() {  
    return("<H2>" + Math.random() + "</H2>");  
}
```

```
%>
```

```
<%= cabeceraRandom() %>
```

Código Servlet Resultante

```
public class xxxx implements HttpJSPPage {  
    private String cabeceraRandom() {  
        return("<H2>" + Math.random() + "</H2>");  
    }  
}
```

Declaración

```
public void _jspService(HttpServletRequest request,  
                        HttpServletResponse response)  
                    throws ServletException, IOException {  
    response.setContentType("text/html");  
    HttpSession session = request.getSession(true);  
    JSPWriter out = response.getWriter();  
    out.println("<H1>Alguna Cabecera</H1>");  
    out.println(cabeceraRandom());  
    ...  
}  
...  
}
```

3. Declaraciones: Ejemplo 2

<%!

private BookDBAO bookDBAO;

public void jsplInit() {

...

}

public void jspDestroy() {

...

}

%>

¿Por qué sintaxis XML?

- ▶ Hemos estado mostrando dos tipos de formato
 - el soportado por la versión 1.1 y por la versión 1.2 de JSP
- ▶ El compatible con el formato XML es la versión 1.2
- ▶ Ejemplos:
 - <jsp:expression>Expresión</jsp:expression>
 - <jsp:scriptlet>Código Java</jsp:scriptlet>
- ▶ Mediante el uso de XML tenemos los siguientes beneficios
 - Validación XML vía el esquema XML
 - Otras herramientas XML como editores, API de Java específica, etc.

Incluyendo y Reenviando a otros Recursos Web



Including and forwarding

Incluir contenido en una JSP

- ▶ Hay dos mecanismos para incluir otro recurso Web en la página JSP
 - Directiva **include**
 - El elemento **jsp:include**

Directiva **Include**

- ▶ Se procesa cuando la página JSP se traduce a la Clase Servlet
- ▶ El efecto es la inserción del texto contenido en otro fichero en la página JSP
 - contenido tanto **estático** como **otra página JSP**

Directiva **Include**

- ▶ Normalmente se utiliza para
 - incluir banners
 - información de copyright
 - o cualquier código que se quiera reutilizar
- ▶ Sintaxis
 - <%@ include file = “nombreFichero” %>
 - <%@ include file = “banner.jsp” %>

Elemento jsp:include

- ▶ Se procesa cuando la página JSP se ejecuta
- ▶ Permite incluir tanto código estático como dinámico
 - **Estático**
 - su código se inserta en el fichero JSP
 - **Dinámico**
 - La solicitud (*request*) se envía la recurso incluido
 - El recurso incluido se ejecuta
 - Finalmente, el resultado del recurso incluido se incluye en la respuesta (*response*) de la página JSP
- ▶ Sintaxis
 - <jsp:include page = “páginaIncluida”>
 - <jsp:include page = “date.jsp”>

¿Cuál utilizar?

- ▶ La convenciones de codificación recomiendan utilizar la directiva **include** cuando el fichero no suele cambiar
 - es un método más rápido que el jsp:include
- ▶ Utiliza **jsp:include** cuando el contenido suele cambiar a menudo
 - o si no se tiene segura la inclusión, o sea, es una decisión en tiempo de ejecución

Reenvío a otro componente Web

- ▶ Mecanismo para transferir el control a otra componente Web desde una página JSP
 - el mismo mecanismo que un Servlet
- ▶ Sintaxis
 - `<jsp:forward page = "/main.jsp">`
- ▶ Cuando se invoca un elemento include o forward
 - el Objeto *request* original es pasado a la página objetivo

Reenvío a otro componente Web

- ▶ También se pueden pasar parámetros
 - <jsp:forward page = "... ">
 - <jsp:param name = "param1" value = "value1"/>
 - </jsp:forward>

Directivas

» Comunicación con el Contenedor

Directivas

- ▶ Son **mensajes o instrucciones** al contenedor JSP
 - para afectar a la estructura general del Servlet
- ▶ No producen salida en el output stream actual
- ▶ Sintaxis
 - `<%@ directive{attr=value}* %>`

Tipos de Directivas

- ▶ Hay 3 tipos de directivas
 - directiva **page**
 - directiva **include**
 - directiva **taglib**

Tipos de Directivas: **page**

- ▶ Directiva **page**
 - Para comunicar atributos dependientes de la página al contenedor JSP
 - Por ejemplo, la importación de Clases
- ▶ Sintaxis
 - **<%@ page import = “java.util.*” %>**

Directiva page

- ▶ Da información de alto nivel sobre el Servlet resultante de la página JSP
- ▶ Controla
 - Que Clases se importan
 - <%@ page import = "java.util.* %>
 - El tipo MIME generado
 - <%@ page contentType = " MIME-Type" %>
 - Que página manejara los errores inesperados
 - <%@ page errorPage = "errorpage.jsp" %>

Tipos de Directivas: **include**

- ▶ Directiva **include**
 - Para incluir texto y/o código a la página JSP en tiempo de traducción
- ▶ Sintaxis
 - **<%@ include file = “headet.html” %>**

Tipos de Directivas: **taglib**

- ▶ Directiva **taglib**
 - Indica una librería tag que el contenedor JSP ha de interpretar
- ▶ Sintaxis
 - **<%@ taglib uri = "mytags" prefix = "codecamp" %>**

Objetos implícitos

» Creados por el contenedor Web

Objetos Implícitos

- ▶ Los Objetos Implícitos se crean por el contendor Web de manera transparente
 - Contienen información relativa a una solicitud (request), página o aplicación
- ▶ Son Objetos accesibles por TODA página JSP sin tener que declararlos

Objetos Implícitos

- ▶ **request** (HttpServletRequest)
- ▶ **response** (HttpServletResponse)
- ▶ **session** (HttpSession)
- ▶ **application** (ServletContext)
- ▶ **out** (of type JspWriter)
- ▶ **config** (ServletConfig)
- ▶ **pageContext**

Ámbito de los Objetos

» Scope

Diferentes Ámbitos

Aplicación

- **application**
- Objetos accesibles para páginas de la misma Aplicación

Sesión

- **session**
- Objetos accesibles para páginas que pertenecen a la misma Sesión

Solicitud

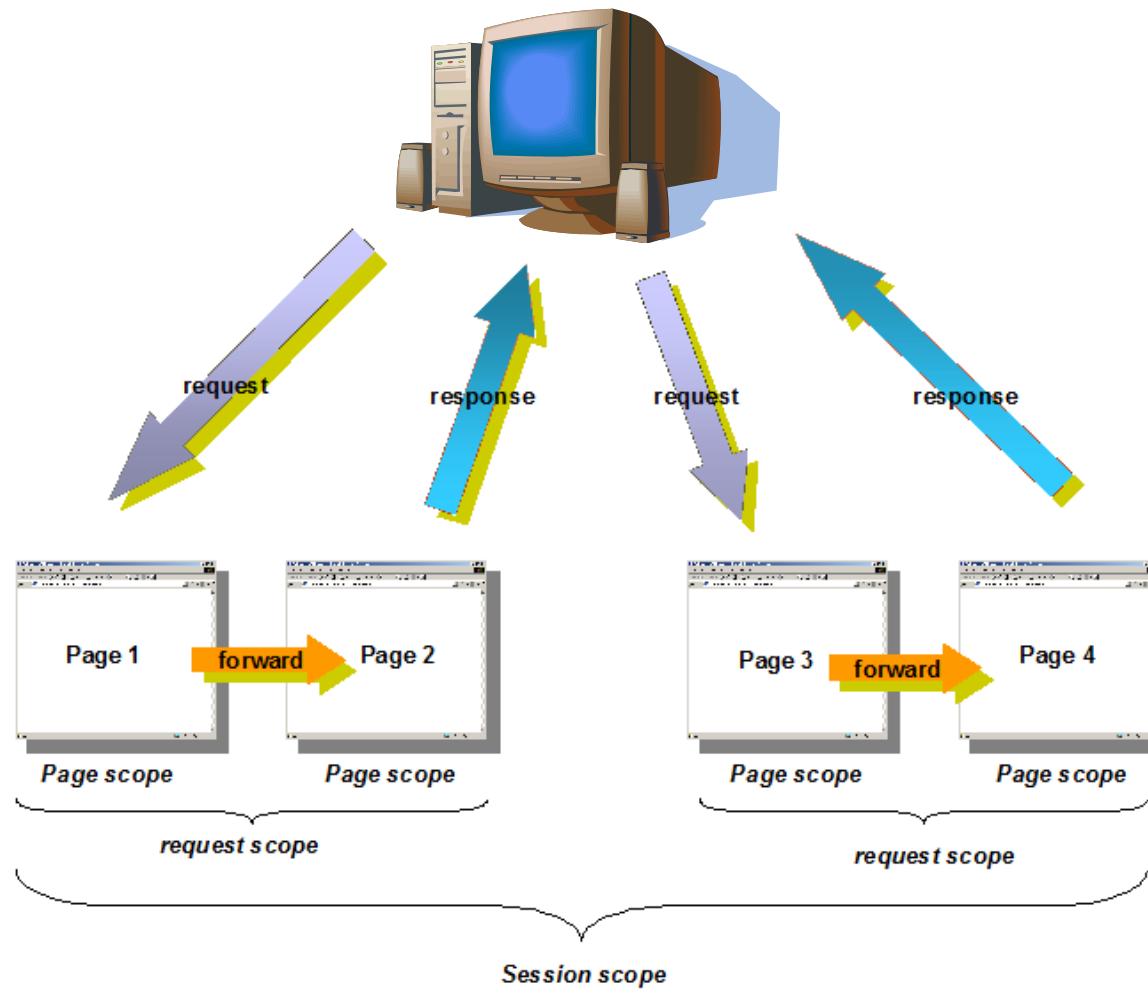
- **request**
- Objetos accesibles para páginas procesando la solicitud

Página

- **page**
- Objetos accesibles solo en la página donde han sido creados

Nota: Ordenados de más a menos visibilidad

Ámbitos: Session, Request y Page



Componentes JavaBeans para JSP

» *JavaBeans: Un tipo especial de Clase de Java*

¿Qué son los JavaBeans?

- ▶ Son Clases Java que pueden fácilmente ser utilizadas por una aplicación Web
- ▶ Cualquier Clase Java que siga ciertas convenciones puede ser una **JavaBean**
 - Las propiedades han de ser privadas
 - Tienen que existir métodos públicos para acceder a estos

¿Qué son los JavaBeans?

- ▶ Dentro de una página JSP se pueden
 - crear, inicializar y acceder a sus propiedades mediante los get y sets
- ▶ Otro de los puntos fuertes es que
 - pueden contener lógica de negocio
 - acceso a BBDD
 - etc.

Convenciones de Diseño

- ▶ Los JavaBeans mantienen propiedades internas
- ▶ Una propiedad puede ser
 - lectura/escritura
 - lectura solo
 - escritura solo
 - simple o indexada
- ▶ Las propiedades han de accederse via métodos getXxx y setXxx (**getters y setters**)
- ▶ Han de incluir un **constructor por defecto**
 - sin parámetros
 - si no se especifica se creará uno por defecto automáticamente

JavaBeans: Ejemplo

```
public class Currency {  
    private Locale locale;  
    private double amount;  
  
    public Currency() {  
        locale = null;  
        amount = 0.0;  
    }  
    public void setLocale(Locale loc) {  
        locale = loc;  
    }  
    public void setAmount(double a) {  
        amount = a;  
    }  
    public String getFormat() {  
        NumberFormat nf = NumberFormat.getCurrencyInstance(locale);  
        return nf.format(amount);  
    }  
}
```

¿Por qué utilizar JavaBeans?

- ▶ Una página JSP puede crear y utilizar cualquier tipo de Objeto Java en la declaración de un scriptlet
- ▶ Por ejemplo

```
<%  
    ShoppingCart cart =  
        (ShoppingCart) session.getAttribute("cart");  
    // Si el usuario no dispone de carro, crear uno  
    if (cart == null) {  
        cart = new ShoppingCart();  
        session.setAttribute("cart", cart);  
    }  
%>
```

Añade el Objeto a la Sesión

¿Por qué utilizar JavaBeans en JSP?

- ▶ Las páginas JSP pueden utilizar elementos para crear y acceder a Objetos que cumplan con las convenciones de JavaBeans

```
<jsp:useBean id="cart" class="cart.ShoppingCart" scope="session"/>
```

- ▶ Crea una instancia de *ShoppingCart* si no existe
 - lo almacena como un atributo en el ámbito de sesión
 - a partir de ahora será accesible a través de la sesión

Comparando las dos opciones

```
<%  
    ShoppingCart cart = (ShoppingCart) session.getAttribute("cart");  
    // Si el usuario no dispone de una Objeto cart como  
    // atributo de la session, el objeto se crea.  
    // Si no, utiliza la instancia existente  
    if (cart == null) {  
        cart = new ShoppingCart();  
        session.setAttribute("cart", cart);  
    }  
%>
```

Comparado con

```
<jsp:useBean id="cart" class="cart.ShoppingCart" scope="session"/>
```

¿Por qué utilizar JavaBeans en JSP?

- ▶ No es necesario aprender Java
- ▶ Gran separación entre contenido y presentación
 - JavaBeans contienen la lógica de negocio
 - JSP la presentación
- ▶ Alta reusabilidad de código
 - Los JavaBeans se pueden reutilizar en otras páginas JSP
- ▶ Sistema sencillo para compartir Objetos entre múltiples páginas JSP

Creación de JavaBeans

- ▶ Se pueden declarar en una página JSP como

```
<jsp:useBean id="nombreBean" class="nombreClase" scope="ambito"/>  
    ○  
<jsp:useBean id="nombreBean" class="nombreClase" scope ="ambito"/>  
        <jsp:setProperty .../>  
</jsp:useBean>
```

Inicializa propiedades

- ▶ El **jsp:useBean** declara que la página utilizará un bean
 - identificándolo por el campo “id”
 - almacenado y accesible en un ámbito determinado
 - aplicación, sesión, solicitud o página
- ▶ Si no existe el bean lo crea y lo almacena en el ámbito especificado

Seteando propiedades JavaBean

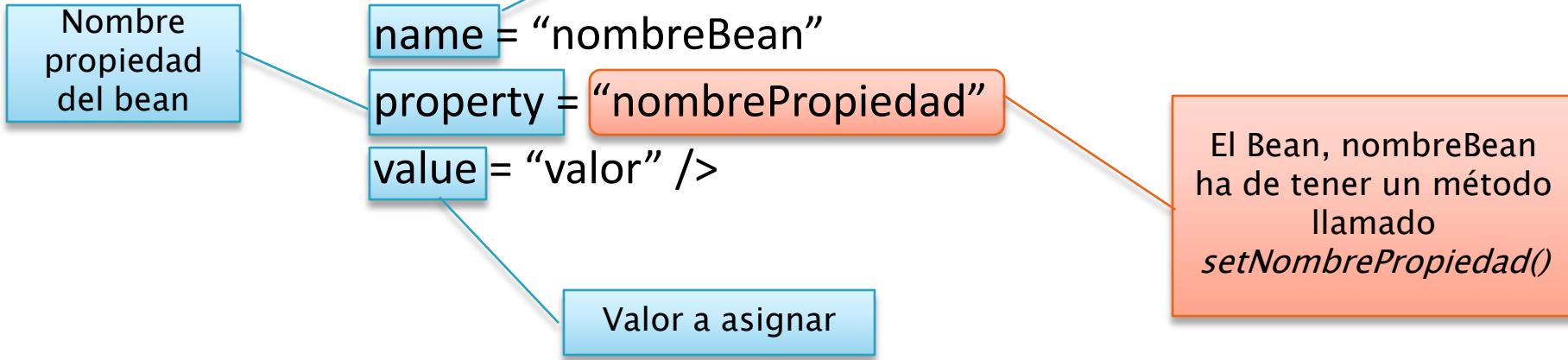
- ▶ Hay dos maneras de dar valor a los propiedades de un JavaBean

- Vía scriptlet

- `<% nombreBean.setNombrePropiedad(valor); %>`

- Vía JSP

- `<jsp:setProperty`



Seteando propiedades JavaBean

- ▶ La sintaxis de **jsp:setProperty** depende de las fuentes

```
<jsp:setProperty name="nombreBean"  
    property="nombrePropiedad" value="constante string"/>
```

```
<jsp:setProperty name="nombreBean"  
    property="nombrePropiedad" param="nombreParametro"/>
```

```
<jsp:setProperty name="nombreBean"  
    property="nombrePropiedad" value=<%= expresion %>/>
```

Leer una Propiedad de una JavaBean

- ▶ Existen dos maneras diferentes de leer una propiedad de un Bean
 - Vía scriptlet
 - `<%= nombreBean.getNombrePropiedad() %>`
 - Vía `jsp:getProperty`
 - `<jsp:getProperty name = "nombreBean" property = "nomProp" />`
- ▶ Requisitos
 - *nombreBean* tiene que ser el mismo que el especificado en el atributo *id* de un elemento *useBean*
 - tiene que tener un método *getNomProp()*

Manejo de Errores

» Error Handling

Manejo de Errores

- ▶ Cualquier tipo de excepción puede lanzarse cuando se ejecuta una página JSP
- ▶ Se puede especificar una página de error para que el contenedor Web reenvíe el control
 - `<%@ page errorPage = "nombreFichero" %>`

Manejo de Errores

- ▶ La página ha de incluir la directiva
 - `<%@ page isErrorPage="true" %>`
 - Esta directiva hace que el Objeto excepción de tipo
 - `javax.servlet.jsp.JspException`
 - este disponible en la página de error
 - de esta manera podemos acceder a este para obtener información
- ▶ Con la referencia a la excepción podemos mostrar información sobre el error
 - `<%= exception.toString() %>`