

Universidad Nacional Autónoma de México

Facultad de ingeniería



Java web

**Edgar E. García Cano
Jorge A. Solano**

Temario

- 1. JDBC (Java DataBase Connectivity)**
- 2. HTML (HyperText Markup Language)**
- 3. Servlets**
- 4. JSP (Java Server Page)**
- 5. JavaBeans**
- 6. Integración de JDBC, Servlets y JSP's**
- 7. Arquitectura Modelo-Vista-Controlador
(MVC)**
- 8. Aplicaciones WEB seguras**
- 9. JavaMail**
- 10. XML**



1

JDBC (Java DataBase Connectivity)



1. JDBC (Java DataBase Connectivity)

1.1 Introducción

1.2.¿Qué es JDBC?

1.3 ¿Qué hace JDBC?

1.4 Conexión a BD utilizando JDBC

1.4.1 Drivers JDBC

1.4.2 Pasos para utilizar JDBC

1.5 Sentencias SQL



1.1 Introducción

Java Database Connectivity (JDBC) es una interfaz de acceso a bases de datos estándar SQL que proporciona un acceso uniforme a una gran variedad de bases de datos relacionales.

JDBC también proporciona una base común para la construcción de herramientas y utilidades de alto nivel.



1.2 ¿Qué es un JDBC?

JDBC es el API para la ejecución de sentencias SQL. Consiste en un conjunto de clases e interfaces escritas en el lenguaje de programación Java. JDBC suministra un API estándar para los desarrolladores y hace posible escribir aplicaciones de base de datos usando un API puro Java.



1.3 ¿Qué hace un JDBC?

JDBC hace posible las siguientes tres cosas:

1. **Establecer una conexión con la base de datos.**
2. **Enviar sentencias SQL.**
3. **Procesar los resultados.**



El API de JDBC permite realizar la conexión con la base de datos, a través de cualquier componente de Java (Servlet, JSP, GUI o aplicación standalone).



Existen diferentes drivers JDBC dependiendo del gestor de base de datos al que se desee conectar. Generalmente, el driver tiene la siguiente sintaxis:

gestorBD-connector-java-versión.jar

donde gestorBD se refiere al gestor al que se quiere conectar (DB2, Sybase, Oracle, SQLServer, PostgreSQL o mySQL)



El driver se debe declarar dentro de la variable CLASSPATH para, posteriormente, pueda ser utilizado dentro de una aplicación:

1. **Agregar a la variable de entorno CLASSPATH la dirección donde se instaló el driver.**
2. **Utilizar el método estático forName de la clase Class de la siguiente manera: Class.forName("clase_del_driver");**

**Class.forName("com.sybase.jdbc2.jdbc.SybDriver");
Class.forName("oracle.jdbc.driver.OracleDriver");**



1. Crear una instancia del JDBC driver.

- `Class.forName("paquete.driver.nombreDriver");`
- `Class.forName("org.postgresql.Driver");`

2. Especificar la dirección de la base de datos.

- `String url = "jdbc:subprotocolo:dbms"`
- `String url= "jdbc:postgresql://127.0.0.1:5432/"`



3. Establecer una conexión utilizando el driver que crea el objeto Connection.

- **Connection conn =
DriverManager.getConnection(url, usuario, contraseña);**

4. Crear un objeto Statement, usando la Connection realizada.

- **Statement stmt = con.createStatement();**



5. Realizar la consulta SQL y ejecutarla usando el objeto Statement creado.

- **ResultSet rs = stmt.executeQuery("Consulta_Select_SQL");**
- **ResultSet rs = stmt.executeUpdate("Consulta_Insert_SQL");**

6. Recibir los resultados en el objeto ResultSet (executeQuery) o el resultado (executeUpdate).

- **while(rs.next()) {**
- **System.out.println("Columna 1: " + rs.getString(1));**
- **System.out.println("Columna 2: " + rs.getString(2));**
- **}**



La conexión utilizada (JDBC) permite comunicarse con la base de datos requerida utilizando sentencias SQL. JDBC no pone ninguna restricción sobre los tipos de sentencias que pueden enviarse, permitiendo el uso de sentencias específicas de la base de datos o incluso sentencias no SQL.

Por tanto, es necesario asegurarse que las sentencias SQL enviadas son soportadas por la base de datos a la que se intenta acceder, así como soportar las consecuencias si no es así.



El objeto de tipo Statement se utiliza para crear peticiones SQL estáticas y regresar el resultado que éstas producen. Por lo general, el objeto Statement se utiliza para la ejecución de sentencias SELECT y en ese caso, se invoca al método executeQuery() el cual regresa un objeto ResultSet que contiene los registros resultantes de la consulta:

```
Statement stm = con.createStatement();
```

```
ResultSet res =stm.executeQuery("select * from emp");
```



Un objeto tipo Statement también se utiliza para realizar sentencias INSERT, UPDATE y DELETE y, en ese caso, se utiliza el método executeUpdate() llevando como argumento la cadena de caracteres (String) con la sentencia SQL:

```
String consulta = “insert into emp values(‘123’,‘Juan’)”
```

```
Statement stm = con.createStatement();
```

```
int res = stm.executeUpdate(consulta);
```



El objeto de tipo PreparedStatement se utiliza para ejecutar sentencias SQL precompiladas (o dinámicas en cuanto a los valores de los parámetros en la condición de la sentencia SQL).

Por lo general, PreparedStatement se utiliza para la ejecución de sentencias INSERT, DELETE y UPDATE ya que éstas últimas requieren condicionantes por lo que se crea la cadena (String) con la sentencia SQL desde la instancia del objeto PreparedStatement.



```
// Se crea el objeto  
String peticion = "update emp set nom_emp=? where  
cve_emp=?"  
PreparedStatement pstmt = con.prepareStatement(peticion);
```

```
// Se envían los parámetros  
pstmt.setString(1,"Juan");  
pstmt.setInt(2,123);  
  
// Se ejecuta la sentencia  
int res = pstmt.executeUpdate();
```



El objeto de tipo CallableStatement se utiliza para ejecutar procedimientos almacenados (stored procedures) SQL.

La API de JDBC provee una sintaxis para que todos los procedimientos almacenados sean llamados de la misma manera en los diferentes sistemas manejadores de bases de datos.



Ejemplo 1.1

Conexion.java



Ejemplo 1.2

MetaDatos.java



Ejemplo 1.3

SeleccionarDatos.java



Ejemplo 1.3

SeleccionarBD.java



Ejemplo 1.4

InsertarBD.java



2

HTML (HyperText Markup Language)



2. HTML (HyperText Markup Language)

2.1 Definición

2.2 Etiquetas básicas



HTML (HyperText Markup Language) es un lenguaje muy sencillo que permite describir hipertexto, es decir, texto presentado de forma estructurada y agradable con enlaces (hyperlinks) que conducen a otros documentos o fuentes de información relacionadas, así como elementos multimedia (gráficos, sonido, videos, etc.).



El lenguaje que se emplea para crear páginas web se denomina HyperText Markup Language (Lenguaje de Margen (o marcado) de Hiper Texto).

HTML se basa en un conjunto de etiquetas (de apertura y de cierre) que definen la funcionalidad en la página creada. A continuación se presentan las etiquetas más utilizadas dentro de HTML.



Etiqueta	Descripción
<HTML> </HTML>	Describe el inicio y fin de la página.
<HEAD> </HEAD>	Cabecera de la página.
<TITLE> </TITLE>	Título de la página.
<BODY> </BODY>	Cuerpo de la página.
<BODY bgcolor="#345678">	Fondo de la página.
<CENTER> </CENTER>	Alinea al centro su contenido.
<H1> </H1> ... <H6> </H6>	Cabecera de mayor a menor tamaño.
<DIV ALIGN=X> </DIV>	Permite justificar el texto del párrafo a la izquierda (ALIGN=LEFT), derecha (RIGHT), al centro (CENTER) o a ambos márgenes (JUSTIFY)
<BLOCKQUOTE> ... Para dejando </BLOCKQUOTE>	citar un texto ajeno. Se suele implementar márgenes tanto a izquierda como a derecha, razón por la que se usa habitualmente.

Etiqueta	Descripción
 	Pone el texto en negrita.
<I> </I>	Representa el texto en cursiva.
<U> </U>	Para subrayar.
<S> </S>	Para tachar.
	Letra superíndice.
	Letra subíndice.
<HR>	Inserta una barra horizontal
 	Salto de línea
<P>	Inicio de un párrafo
<MARQUEE> </MARQUEE>	Desplaza el texto por toda la pantalla
<BIG> </BIG>	Incrementa el tamaño del tipo de letra.
<SMALL> </SMALL>	Disminuye el tamaño del tipo de letra.

Etiqueta	Descripción
 	Crea un enlace (hipervínculo).
	Inserta la imagen especificada.
<FORM> </FORM> action=url method=tipo name=nombre	Crea un formulario. Página que gestiona el formulario. Método empleado para el envío. Nombre del formulario
<SELECT> </SELECT>	Menú de selección.
<OPTION> </OPTION>	Genera las opciones del menú selección.
<TEXTAREA> </TEXTAREA>	Área de texto.
<INPUT> maxlength type	Caja de texto Número máximo de caracteres Tipo: text, password, radio, checkbox, submit, reset.

EJEMPLO 2.1



EJEMPLO 2.2



EJEMPLO 2.3



3

Servlets



3. Servlets

3.1 Protocolo HTTP

3.2 Ciclo de vida de un servlet

3.3 API de Servlets

3.4 Interfaz ServletConfig

3.5 Interfaz ServletContext

3.6 Interfaz RequestDispatcher

3.7 Descriptor de aplicaciones

3.8 Manejo de sesiones

3.9 Integración de Servlets con JDBC



En java, los Servlets son utilizados para manejar la lógica de negocio de una aplicación web. Los Servlets pueden ser aplicados en cualquier protocolo, pero, en la práctica, son más utilizados en el protocolo HTTP.

Los Servlets son módulos que extienden los servidores orientados a petición-respuesta, como los servidores web compatibles con Java.



Un Servlet puede ser responsable de leer los datos de un formulario HTML para aplicarle la lógica de negocios que permita actualizar la base de datos de pedidos de la compañía, por ejemplo.

Los paquetes javax.servlet y javax.servlet.http proveen interfaces y clases que permiten definir Servlets. Todos los Servlets deben implementar la interfaz Servlet.



El protocolo de transferencia de hipertexto (HTTP, HyperText Transfer Protocol) es el medio usado en cada transacción Web.

El hipertexto es el contenido de las páginas web y el protocolo de transferencia es el sistema mediante el cual se envían las peticiones de acceso a una página y la respuesta con el contenido.



El protocolo también sirve para enviar información adicional en ambos sentidos.

HTTP es un protocolo sin estado, es decir, que no guarda ninguna información sobre conexiones anteriores. Al finalizar la transacción todos los datos se pierden.





Métodos de HTTP

La primera línea de una petición contiene los comandos HTTP, conocidos como métodos. Existen diferentes métodos, los más conocidos y utilizados son: GET, HEAD y POST.



Método GET

El método GET se utiliza para recuperar información identificada por un URI por parte de los navegadores. Este método también se puede utilizar para pasar una pequeña cantidad de información al servidor en forma de pares atributo-valor añadidos al final del URI detrás de un símbolo de interrogación (?).



Método HEAD

El método HEAD es similar al método GET, salvo que el servidor no tiene que devolver el contenido, sólo las cabeceras. Estas cabeceras que se devuelven son las mismas que las que se devolverían si fuese una petición GET.

Este método se puede usar para obtener información sobre el contenido de un sitio. Se suele usar también para revisar la validez de un enlace, la accesibilidad o las modificaciones recientes del sitio deseado.



Método PUT

El método PUT permite guardar el contenido de una petición en el servidor bajo la URI de petición. Si esta URI ya existe, entonces el servidor considera que esta petición proporciona una versión actualizada del recurso. Si la URI indicada no existe y es válida para definir un nuevo recurso, el servidor puede crear el recurso con esa URI.

Si se crea un nuevo recurso, debe responder con un código 201 (creado), si se modifica se contesta con un código 200 (OK) o 204 (sin contenido). En caso de que no se pueda crear el recurso se devuelve un mensaje con el código de error apropiado.



Método POST

El método POST se usa para hacer peticiones en las que el servidor destino acepta el contenido de la petición como un nuevo subordinado del recurso pedido.

POST se creó para cubrir funciones como la de enviar un mensaje a grupos de usuarios, dar un bloque de datos como resultado de un formulario a un proceso de datos y/o añadir nuevos datos a una base de datos.



La tecnología de Servlets de Java es comúnmente usada para manejar la lógica de negocio de una aplicación web. Pueden ser aplicados en cualquier protocolo, pero en la práctica, son escritos para ser utilizados mediante el protocolo HTTP.

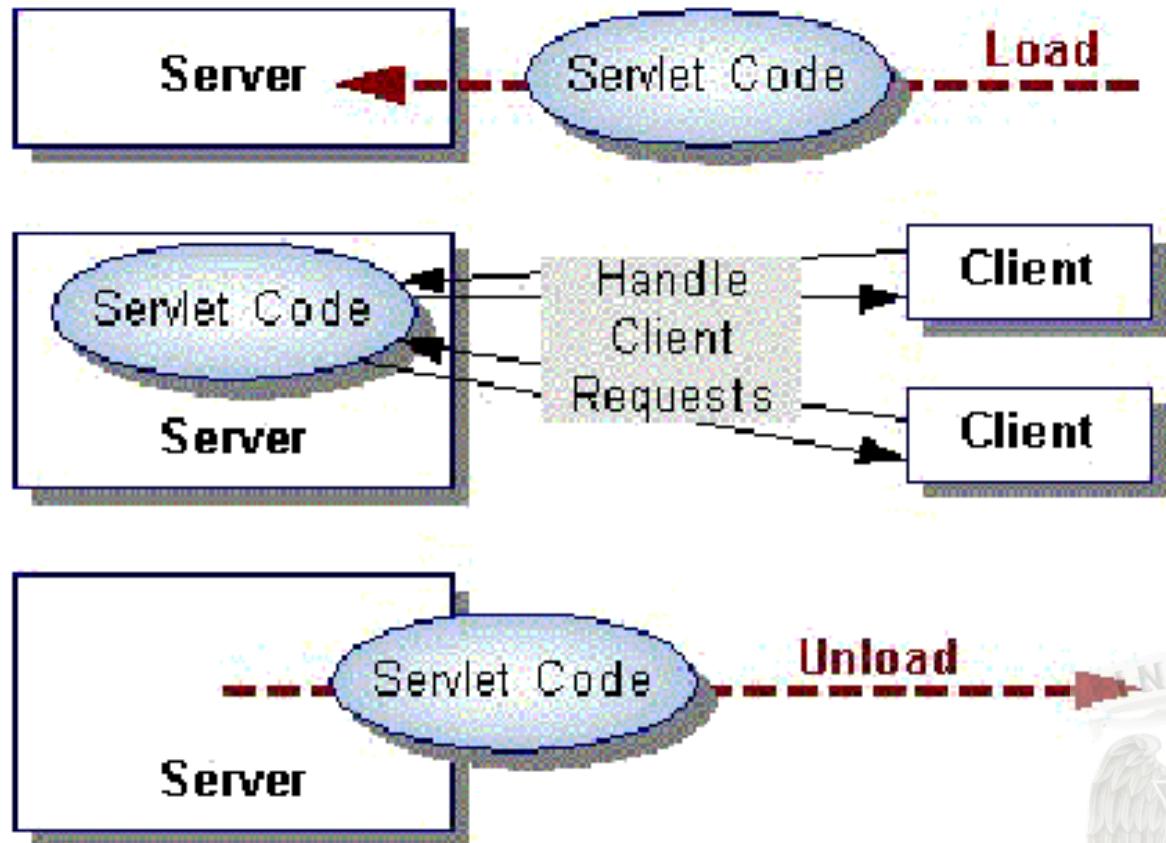
Los Servlets son módulos que extienden los servidores orientados a petición-respuesta, como los servidores web compatibles con Java.



El ciclo de vida de un servlet está dado por los siguientes pasos:

- **Cargar e instanciar el servlet.**
- **Iniciar el Servlet**
- **Interacción con Clientes (espera de peticiones)**
- **Destrucción del Servlet**
- **Descarga del Servlet**





Cuando el contenedor de Servlets se inicia, busca los archivos de configuración de las aplicaciones.

Cada aplicación web tiene su propio descriptor llamado web.xml, que incluye una descripción de cada Servlet que compone la aplicación.



El contenedor de Servlets crea una instancia del Servlet usando la sintaxis Class.forName(className).newInstance().

Para hacer lo anterior el Servlet debe de tener un constructor público y sin argumentos, generalmente no se define ningún constructor en la clase.



Cuando el servidor carga un Servlet, se ejecuta el método init(ServletConfig) del Servlet. La inicialización se completa antes de manejar peticiones de clientes y, obviamente, antes de que el Servlet sea destruido.



Aunque muchos Servlets se ejecutan en servidores multi hilo, los Servlets no tienen problemas de concurrencia durante su inicialización. El servidor llama sólo una vez al método init(), cuando carga el Servlet, y no lo llamará de nuevo a menos que vuelva a recargar el servlet.

El servidor no puede recargar un Servlet sin primero haber destruido el servlet llamando al método destroy().



Una vez que el Servlet ha iniciado es posible atender peticiones de clientes.

Después de que la instancia es propiamente inicializada el Servlet esta listo para dar servicio.

Cuando el contenedor recibe una petición para el Servlet, éste la atenderá llamando al método de la interfaz Servlet `Servlet.service(ServletRequest,ServletResponse)`.



Los Servlets se ejecutan hasta que el servidor los destruye. Para destruir un Servlet se invoca al método `destroy()` del propio Servlet.

Este método sólo se ejecuta una vez. El servidor no ejecutará de nuevo el Servlet hasta haberlo cargado e inicializado de nuevo.



Una vez destruido un servlet, la instancia es candidata a ser recogida por el recolector de basura. Si el Servlet ha sido destruido porque el Servicio del contenedor se detuvo, el Servlet también es descargado.



La especificación de los Servlets de Sun proveen un estándar e independencia de la plataforma para poder realizar comunicación entre los Servlets y sus contenedores.

El grupo de clases e interfaces es llamado **Servlet Application Programming Interfaces (**Servlet API**).**

El API de Servlet está dividido en dos paquetes: `javax.servlet` y `javax.servlet.http`.



Este paquete contiene las interfaces y clases genéricas de los Servlets que son independientes de la plataforma.

La interfaz básica de los servlets es javax.servlet.Servlet. Cada Servlet implementa directa o indirectamente esta interfaz y contiene los métodos:

- **init()**
- **service()**
- **destroy()**
- **getServletConfig()**
- **getServletInfo()**



La clase abstracta javax.servlet.GenericServlet tiene implementación para todos los métodos excepto service().

La interfaz javax.servlet.ServletRequest provee una vista genérica de la petición que fue enviada por un cliente.

La interfaz javax.servlet.ServletResponse provee una forma genérica para el envío de respuestas.



Este paquete provee la funcionalidad básica requerida para los HTTP Servlets. Las clases e interfaces de éste paquete tienen soporte para utilizar el protocolo HTTP.

La clase abstracta javax.servlet.http.HttpServlet herada de GenericServlet y asigna un método service() con la siguiente firma.

```
protected void  
service(HttpServletRequest,HttpServletResponse)  
throws ServletException, IOException
```



La interfaz javax.servlet.http.HttpServletRequest hereda de ServletRequest y provee formas específicas HTTP para las peticiones.

La interfaz javax.servlet.http.HttpServletResponse hereda de ServletResponse y provee formas específicas para el envío de respuestas HTTP.



EJEMPLO 3.1



Un Servlet HTTP maneja peticiones del cliente a través del método service(). Este método soporta peticiones estándar de cliente HTTP atendiendo cada petición a un método designado para manejarla.



Los métodos de la clase HttpServlet que manejan peticiones de cliente toman dos argumentos.

- **Un objeto HttpServletRequest, que encapsula los datos desde el cliente.**
- **Un objeto HttpServletResponse, que encapsula la respuesta hacia el cliente.**



HttpServletRequest

Un objeto HttpServletRequest proporciona acceso a los datos de cabecera HTTP. El objeto HttpServletRequest también permite obtener los argumentos que el cliente envía como parte de la petición.



HttpServletResponse

Un objeto HttpServletResponse proporciona dos formas de regresar datos al usuario:

- **Por el método getWriter devuelve un Writer.**
- **Por el método getOutputStream devuelve un ServletOutputStream.**



PrintWriter

El método `getWriter()` regresa un objeto del tipo `java.io.PrintWriter` que permite enviar datos. `PrintWriter` es usado por los Servlets para generar páginas HTML dinámicas

Una vez que se ha analizado la petición, el Servlet puede decidir si quiere re-direccionar a algún recurso. Para realizar éste trabajo `HttpServletResponse` provee el método `sendRedirect()`



EJEMPLO 3.2



Los Servlets poseen el método init() que permite inicializar su configuración.

El manejador de Servlets envía como argumento al método init() un objeto de tipo ServletConfig (init(ServletConfig)). La clase ServletConfig sólo tiene métodos para obtener los parámetros, no para enviarlos.



Método	Descripción
String getInitParameter (String paramName)	Regresa sólo un valor asociado con el parámetro dado o null si no esta disponible.
Enumeration getInitParameterNames ()	Regresa una Enumeración de Strings de todos los nombres de los parámetros.
ServletContext getServletContext ()	Regresa el ServletContext de este servidor.
String getServletName ()	Regresa el nombre del servlet especificado en el archivo de configuración.

Para poder acceder a la configuración del Servlet es necesario:

- **Declarar el archivo web.xml**
- **Obtener la configuración del Servlet**



Archivo web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app>
    <servlet>
        <servlet-name>TestServlet</servlet-name>
        <servlet-class>TestServlet</servlet-class>
        <init-param>
            <param-name>driverclassname</param-name>
            <param-value>
                sun.jdbc.odbc.JdbcOdbcDriver
            </param-value>
        </init-param>
```



```
<init-param>
    <param-name>dburl</param-name>
    <param-value>jdbc:odbc:MySQLODBC</param-value>
</init-param>
<init-param>
    <param-name>username</param-name>
    <param-value>testuser</param-value>
</init-param>
<init-param>
    <param-name>password</param-name>
    <param-value>test</param-value>
</init-param>
<load-on-startup>1</load-on-startup>
</servlet>
</web-app>
```



Para obtener la configuración del Servlet, en el archivo se debe insertar el siguiente código.

```
ServletConfig config = getServletConfig();
String driverClassName = config.getInitParameter("driverclassname");
String dbURL = config.getInitParameter("dburl");
String username = config.getInitParameter("username");
String password = config.getInitParameter("password");
```



EJEMPLO 3.3



La interfaz ServletContext es la ventana para que un Servlet vea su entorno. El Servlet usa a esta interfaz para obtener información como la inicialización de parámetros.

Cada aplicación web tiene un ServletContext. El contexto es inicializado al mismo tiempo que la aplicación es cargada.



Por lo tanto, la información del contexto del servidor web está disponible en cualquier momento a través del objeto ServletContext.

Un Servlet puede obtener este objeto llamando al método getServletContext() del objeto ServletConfig.

Hay que recordar que el objeto ServletConfig se pasa al servlet en el método init(), al cargar el servlet.



Método	Descripción
<code>getAttribute()</code>	Obtiene información acerca del servidor en pares de atributos nombre/valor.
<code>getMimeType()</code>	Regresa el tipo MIME de un archivo dado.
<code>getRealPath()</code>	Convierte una ruta relativa o virtual a una nueva ruta en relación con el directorio raíz de los documentos HTML.
<code>getServerInfo</code>	Regresa el nombre y la versión de los servicios de red en los que está ejecutándose el servlet.
<code>getServlet()</code>	Regresa un objeto Servlet de un nombre dado. Útil cuando se desea tener acceso a los servicios de otro servlet.
<code>getServletNames()</code>	Regresa un objeto Enumeration con los nombres de los servlets disponibles en el actual espacio de nombres.
<code>log()</code>	Escribe información al archivo de bitácora. Ese archivo y su formato son específicos de cada servidor web.

EJEMPLO 3.4



Esta interfaz permite al saber cuando el servlet context es inicializado o destruido.

Por ejemplo, se puede crear una conexión a una base de datos tan pronto como el contexto sea inicializado y cerrarla cuando el contexto sea destruido.



EJEMPLO 3.5



Un componente web puede invocar a otros recursos de manera directa y de manera indirecta.

Para invocar indirectamente a otros recursos web se agrega a la respuesta enviada al cliente una URL que apunta a otro componente web.



Para invocar un recurso disponible en el servidor en el que se está ejecutando el componente web primero se debe obtener un objeto RequestDispatcher utilizando el método:

getRequestDispatcher(“URL”).

El método getRequestDispatcher toma la ruta del recurso solicitado como un argumento. El contexto web requiere una ruta relativa.



Si el recurso no está disponible o si el servidor no ha implementado un objeto RequestDispatcher para ese tipo de recurso, el método getRequestDispatcher regresará null.

En algunas aplicaciones se desea tener un componente web que realice procesamiento preliminar de una petición y tener otro componente que genere la respuesta. Para transferir el control a otro componente web se debe invocar al método forward del objeto RequestDispatcher.



Cuando una petición es re-direccinada, el URL de la petición se establece a la ruta de la nueva página a la que se direccionó la petición. Si el URL original es requerido para algún proceso, se puede salvar como un atributo más de la petición.

El método forward debe ser utilizado para dar a otro recurso la responsabilidad de responderle al cliente.



EJEMPLO 3.6



El descriptor de una aplicación, como su nombre lo indica, describe las clases que existen en la aplicación para que el contenedor de Servlets la entienda.

El descriptor se maneja a través de un archivo llamado web.xml.



```
<?xml version="1.0" encoding="ISO-8859-1" ?>           ← Declares the XML version and character set used in this file
<web-app version="2.4"
    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-app_2_4.xsd" >      ← Declares the schema definition for this file
<display-name>Test Webapp</display-name>
<context-param>
    <param-name>author</param-name>
    <param-value>john@abc.com</param-value>
</context-param>                                         ← Specifies a parameter for this web application

<servlet>      ← Specifies a servlet
    <servlet-name>test</servlet-name>
    <servlet-class>com.abc.TestServlet</servlet-class>
    <init-param>      ← Specifies a parameter for this servlet
        <param-name>greeting</param-name>
        <param-value>Good Morning</param-value>
    </init-param>
</servlet>

<servlet-mapping>
    <servlet-name>test</servlet-name>
    <url-pattern>/test/*</url-pattern>
</servlet-mapping>                                         ← Maps /test/* to test servlet

<mime-mapping>
```

Cada elemento entre las etiquetas <servlet> define un servlet en la aplicación.

```
<servlet>
    <servlet-name>us-sales</servlet-name>      ← The servlet name
    <servlet-class>com.xyz.SalesServlet</servlet-class>   ← The servlet class
    <init-param>
        <param-name>region</param-name>
        <param-value>USA</param-value>
    </init-param>
    <init-param>
```

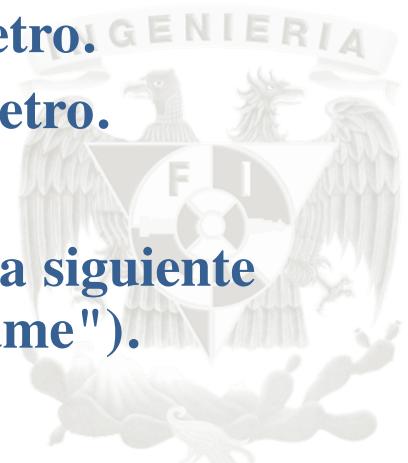
The servlet parameters



El contenedor de servlets hace una instancia y lo asocia con el nombre del servlet.

- **<servlet-name>:** Esta etiqueta define el nombre del servlet.
- **<servlet-class>:** Esta etiqueta especifica la clase Java que será usada por el contenedor de Servlets.
- **<init-param>:** Esta etiqueta sirve para agregar parámetros iniciales al servlet. Se tiene que crear una etiqueta por cada parámetro que se necesite. Contiene las etiquetas **<param-name>** y **<param-value>**.
 - **<param-name>:** indica el nombre del parámetro.
 - **<param-value>** que indica el valor del parámetro.

Dentro del servlet, los parámetros se obtienen de la siguiente manera: `ServletConfig.getInitParameter("paramname")`.



En esta etiqueta hay que especificar el URL que será manejado por el Servlet. El contenedor de Servlets utiliza este mapeo para invocar el Servlet apropiado.

```
<servlet-mapping>
    <servlet-name>accountServlet</servlet-name>
    <url-pattern>/account/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
    <servlet-name>accountServlet</servlet-name>
    <url-pattern>/myaccount/*</url-pattern>
</servlet-mapping>
```



El manejo de sesión es un mecanismo que los Servlets utilizan para mantener el estado sobre la serie de peticiones desde un mismo usuario (peticiones originadas desde el mismo navegador) durante algún periodo de tiempo.

Las sesiones son compartidas por los Servlets a los que accede el cliente. Esto es conveniente para aplicaciones compuestas por varios Servlets.



Para poder manejar sesiones se debe:

- **Obtener una sesión (un objeto HttpSession) para un usuario.**
- **Almacenar u obtener datos desde el objeto HttpSession.**
- **Invalidar la sesión (opcional).**



Obtener un sesión

El método `getSession` del objeto `HttpServletRequest` devuelve una sesión de usuario. Cuando se invoca al método con su argumento `create` como `true`, la implementación creará una sesión si es necesario.

```
HttpSession session = request.getSession(true);
```

Para mantener la sesión apropiadamente se debe llamar a `getSession` antes de escribir cualquier respuesta.



Almacenar y obtener datos desde la sesión

La Interfaz HttpSession proporciona métodos que permiten almacenar y recuperar diferentes aspectos de una sesión, como.

- Propiedades de sesión estándar (identificador de sesión).
- Datos de la aplicación que son almacenados como parejas nombre-valor (donde el nombre es un string y los valores son objetos del lenguaje de programación Java).

```
session.setAttribute(session.getId(), cart);
```



Invalidar una sesión

Invalidar una sesión significa eliminar el objeto HttpSession y todos sus valores del sistema.

Una sesión de usuario puede ser eliminada de manera manual o de manera automática, dependiendo del contexto del servlet. Por ejemplo, Java Web Server invalida una sesión cuando no hay peticiones de página por un periodo de tiempo (30 minutos por defecto).

Para invalidar manualmente una sesión se utiliza el método *invalidate* del objeto *session*.

```
session.invalidate();
```



EJEMPLO 3.7



3.9 Integración de Servlets con JDBC

El siguiente paso consiste en unir todas las herramientas que ya se poseen con un fin común.

Debido a que ya se tiene la clase que realiza la conexión con la base de datos, es posible llamar dicha clase desde el Servlet para así interactuar directamente con la BD.



EJEMPLO 3.8



4

JSP (Java Server Page)



4. JSP(Java Server Page)

4.1 Sintaxis

**4.2 Atributos de la directiva
page**

4.3 Scriptlets

4.4 Objetos implícitos

4.5 Alcances



4. JSP (Java Server Page)

Un JSP está constituido por una página Web (código HTML) que posee código Java embebido. Todo el código Java es interpretado del lado del servidor, por lo tanto, el cliente no puede ver ningún tipo de código, lo único que ve es una página HTML.



Debido a que un JSP contiene código Java es posible realizar páginas dinámicas. Una página dinámica es aquella que no tiene un contenido fijo sino que éste puede variar con el tiempo.

El contenido dinámico puede ser obtenido de una base de datos, de archivos de texto plano, del usuario, etc.



4.1 Sintaxis

Los JSPs poseen gramática propia bien definida. La siguiente tabla muestra los elementos pertenecientes a los JSPs.

- **Directivas:** información acerca de la página.
- **Declaraciones:** declarar y definir métodos y variables.
- **Scriptlets:** elementos que soportan código java.
- **Expresiones:** permite imprimir variables.
- **Acciones:** comando hacia el motor jsp.
- **Comentarios:** permite documentar.



4.1.1 Directivas

Las directivas poseen información acerca de las páginas dentro de los JSPs que procesa la máquina. Existen tres tipos de directivas:

- **page**
- **include**
- **taglib**



- **page:** Informa al motor acerca de las propiedades de un JSP.
`<%@ page language="java" %>`
- **include:** Permite insertar los contenidos de otro archivo en la página actual.
`<%@ include file="copyright.html" %>`
- **taglib:** Se usa para asociar una librería.
`<%@ taglib prefix="test" uri="taglib.tld" %>`



4.1.2 Declaraciones

Declaran y definen tanto variables como métodos que pueden utilizarse dentro de los JSPs. Las variables sólo son inicializadas una vez y retiene el valor para las siguientes peticiones.

```
<%! int count = 0; %>
```

Generalmente, las declaraciones son utilizadas para crear variables o métodos que se requieran utilizar en un JSP para evitar declararlos cada vez que se tenga que utilizar.



```
<%!
    String color[] = {"red", "green", "blue"};
    String getColor(int i){
        return color[i];
    }
%>
```



```
<%! private int j = 0; %>

<%!
    public int suma(int a, int b) {
        return a + b;
    }
%>
```



4.1.3 Scriptlets

Son fragmentos de código Java. El scriptlet es ejecutado cada vez que la página es accedida.

```
<%
    out.print("<html><body>");
    count++;
    out.print("Bienvenido visitante nUmero " + count);
    out.print("</body></html>");
%>
```



4.1.4 Expresiones

Sirven para imprimir el valor de una variable o definir una expresión. La expresión es evaluada cada vez que se accede a la página.

Básicamente, son una única línea de código y no incluyen operadores de control de flujo como if, while, for, etc.



```
<html><body>
<%@ page language="java" %>
<%! int count = 0; %>
    Bienvenido! Visitante numero:<%= ++count %>
</body></html>
```

Es importante hacer notar que las expresiones no terminan con punto y coma (;) mientras que las declaraciones y los scriptlets sí.

Las expresiones son tomadas de manera interna por el motor de JSPs como cadenas (objetos del tipo java.lang.String).



4.1.5 Acciones

Son comandos dados al motor de JSP que sirven para tareas específicas al momento en que se ejecuta la página. Existen 6 acciones estándar:

- **jsp:include**
- **jsp:forward**
- **jsp:useBean**
- **jsp:setProperty**
- **jsp:getProperty**
- **jsp:plugin**



Las acciones jsp:incluye y jsp:forward habilitan la reutilización de componentes dentro de los JSP's.

Las acciones jsp:useBean, jsp:setProperty y jsp:getProperty están relacionadas con los javaBeans (que se verán en el siguiente tema).

Finalmente, la acción jsp:plugin le indica al motor de JSPs que genere el código apropiado para componentes embebidos como applets.



4.1.4 Comentarios

Los comentarios no tienen efecto alguno sobre la salida de las JSPs. Se utilizan, generalmente, para hacer documentación sobre la página.

```
<HTML>
<BODY>
    Bienvenido!
    <%-- Comentario en JSP --%>
    <% // Comentario en Java %>
    <!-- Comentario en HTML -->
</BODY>
</HTML>
```



EJEMPLO 4.1



4.2 Atributos de la directiva page

La directiva page le informa al motor de JSP's sobre todas las propiedades de la página a ejecutar.

Existen 12 posibles atributos, pero en la práctica los que más se utilizan son cuatro: import, session, errorPage e isErrorPage.



Atributo *import*

Sirve para importar clases y paquetes. Los valores que tiene por default son los siguientes:

```
java.lang.*;  
javax.servlet.*;  
javax.servlet.jsp*;  
javax.servlet.http.*;
```



```
<%@ page import="java.util.* , java.io.* , java.text.* ,  
com.mycom.* , com.mycom.util.MyClass " %>  
  
<%@ page import="java.util.* " %>  
<%@ page import="java.io.* " %>  
<%@ page import="java.text.* " %>  
<%@ page import="com.mycom.* , com.mycom.util.MyClass " %>
```



Atributo *session*

Este atributo indica que el JSP es parte de la sesión HTTP. El valor por defecto es true.

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



Atributo *errorPage*

Se utiliza para especificar una URL de error, por defecto tiene el valor null.

```
<%@ page errorPage="errorHandler.jsp" %>
```



Atributo isErrorPage

Indica si la JSP es capaz de manejar errores. Su valor predeterminado es false.

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



EJEMPLO 4.2



Códigos de HTTP

Los siguientes son códigos comunes capturados por el servidor y procesados correctamente, devolviendo información concreta acerca del contexto.

- **Error 200: Ok. (Correcto)**
- **Error 201: Created. (Creado)**
- **Error 204: No content. (No hay contenido)**



Códigos mediante los cuales se indica una redirección al usuario o que la página actual ha cambiado de dirección.

- **Error 301: Moved Permanently. (Movido permanentemente)**
- **Error 307: Moved Temporarily. (Movido temporalmente)**
- **Error 303: See others. (Ver otros)**
- **Error 304: Not modified. (No modificado)**
- **Error 300: Multiple choices. (Múltiples opciones)**



Errores de la comunicación en la parte del usuario (cliente).

- **Error 400: Bad request.** (Petición incorrecta)
- **Error 401: Unauthorized.** (No autorizado)
- **Error 403: Forbidden.** (Prohibido)
- **Error 404: Not found.** (No se encuentra)
- **Error 405: Method not allowed.** (Método no permitido)
- **Error 406: Not acceptable.** (No aceptable)
- **Error 409: Conflict.** (Conflicto)
- **Error 410: Gone.** (El recurso ya no existe)
- **Error 412: Precondition failed.** (Precondición fallida)
- **Error 413: Request Entity too large.** (Petición de entidad demasiado grande)
- **Error 414: Request URI too long.** (Dirección demasiado larga)
- **Error 415: Unsupported Media Type.** (Tipo multimedia no soportado)

Errores de la comunicación en la parte del servidor.

- **Error 500: Internal Server Error. (Error interno del servidor)**
- **Error 501: Not implemented. (No implementado)**
- **Error 503: Service Unavailable. (Servicio no disponible)**



EJEMPLO 4.3



4.3 Scriptlets

Todos los miembros definidos en un JSP son convertidos a un Servlet, por lo tanto, no importa el orden donde se declaran las etiquetas ni el lugar.

```
Usando pi = <%=pi%>, el &acute;rea de un  
c&iacute;rculo<br>  
con radio 3 es <%=area(3)%>  
<%!  
    double area(double r) {  
        return r*r*pi;  
    }  
%>  
<%! final double pi=3.14159; %>
```



EJEMPLO 4.4



Las variables declaradas en un scriptlet se convierten en locales, por lo tanto el orden en que aparecen sí tiene importancia.

```
<html>
<body>
    <% String s = s1+s2; %>   <- No se ha definido la variable s2
    <%! String s1 = "hello"; %><- Variable s1
    <% String s2 = "world"; %><- Variable s2
    <% out.print(s); %>
</body>
</html>
```



EJEMPLO 4.5



Inicializar variables

En lenguaje Java, las variables de instancia son automáticamente inicializadas con sus valores por defecto mientras que las variables locales deben de ser inicializadas explícitamente cuando son utilizadas.

En los scriptlets sucede lo mismo, las variables definidas dentro de las etiquetas <%! %> (declaraciones) son iniciadas automáticamente mientras que las variables definidas dentro de las etiquetas <% %> deben de ser inicializadas explícitamente.



Por lo mencionado anteriormente, el siguiente código fallaría:

```
<html>
<body>
<%! int i; %>
<% int j; %>
El valor de i es <%= i++ %> <br> <- El valor es 0.
El valor de j es <%= j++ %> <br> <- No ha sido inicializado.
</body>
</html>
```



Dentro de un scriptlet también es posible realizar lo siguiente:

```
<html>
<body>
    <%! int i=1010; %>
    <% if( i == 1010 ) { %>
        <input tipe=text name=cuadro value=<%=i%>>
    <% } %>
</body>
</html>
```



EJEMPLO 4.6



EJEMPLO 4.7



4.4 Objetos implícitos

En JSPs se poseen nueve recursos que están disponibles en cualquier parte de la página y, por tanto, no necesitan crearse ni iniciarse. Estos recursos son los llamados objetos implícitos.



Objetos Implícitos	Tipo	Alcance	Métodos más utilizados
request	javax.servlet.ServletRequest	Petición (objeto Request)	getAttribute, getParameter, getParameterNames, getParameterValues
response	javax.servlet.ServletResponse	Página	Comúnmente utilizada de manera interna y no por los autores de documentos JSP
pageContext	javax.servlet.jsp.PageContext	Página	findAttribute, getAttribute, getAttributesScope, getAttributeNamesInScope
session	javax.servlet.http.HttpSession	Sesión	getId, getValue, getValueNames, putValue
application	javax.servlet.ServletContext	Aplicación	getMimeType, getRealPath

Objetos Implícitos	Tipo	Alcance	Métodos más utilizados
out	javax.servlet.jsp.JspWriter	Página	clear, clearBuffer, flush, getBufferSize, getRemaining
config	javax.servlet.ServletConfig	Página	getInitParameter, getInitParameterNames
page	java.lang.Object	Página	Comúnmente utilizada de manera interna y no por los autores de documentos JSP
exception	java.lang.Throwable	Página	getMessage, getLocalizedMessage, printStackTrace, toString

4.5 Alcances

El alcance se refiere a la manera en que se comparten los datos dentro de los Servlets, para ello se tienen tres contenedores de objetos ServletContext, HttpSession y HttpServletRequest. Los alcances asociados con los objetos son, respectivamente, aplicación (application), sesión (session) y petición (request).

Los JSPs poseen los mismos alcances que los Servlets (ServletContext, HttpSession y HttpServletRequest) y, además, se agrega el objeto PageContext bajo el alcance página (page).



Aplication

En el alcance *application* los objetos son compartidos a través de todos los componentes de la aplicación web y son accesibles durante el tiempo de vida de la aplicación.

Estos objetos son mantenidos en el objeto `ServletContext`. Para compartir u obtener elementos de dicho objeto se utilizan los métodos `setAttribute()` y `getAttribute()`.



Session

Los elementos en sesión son compartidos a través de todas las peticiones que haga un solo usuario y están accesibles mientras que la sesión esté activa. Para compartir objetos a este nivel se utilizan los métodos `session.setAttribute` y `session.getAttribute`.



Request

Los elementos en el alcance request son compartidos a través de los componentes que hayan realizado la misma petición y seguirán activos mientras siga la misma petición.

Para obtener un objeto se utilizan los métodos `request.setAttribute` y `request.getAttribute`.



Page

Los objetos en éste tipo de alcance son accesibles sólo en la traducción de la JSP. Estos objetos están disponibles sólo a través de pageContext. También posee los métodos setAttribute() y getAttribute().



5

JavaBeans



5. JavaBeans

5.1 JavaBeans

5.2 JavaBeans dentro de JSP
's



5.1 JavaBeans

Los JavaBeans son componentes de software independiente que se usan para ensamblar otras aplicaciones.

Son elementos encapsulados en forma de variables de instancia; estas variables de instancia son referidas como propiedades del Bean.

La clase provee métodos para obtener (get) y modificar (set) el valor de un elemento.



Los JavaBeans se utilizan en los JSP's mediante la etiqueta <jsp:useBean> la cual crea una instancia de un JavaBean.

Los JavaBeans se utilizan debido a que:

- **Tanto en JSPs como en servlets, sus propiedades pueden ser modificadas dinámicamente y accedidas en tiempo de ejecución.**
- **En JSPs, tienen un tiempo de vida de una única página o de una petición de usuario. Si un JavaBean está siendo utilizado con el alcance de sesión, entonces será serializado de manera automática.**



Para que una clase sea considerada un JavaBean debe de seguir las siguientes características:

- La clase debe de tener un constructor público sin argumentos. Esto permita que la clase sea instanciada por el motor de JSPs.



- Por cada propiedad la clase debe contener dos métodos públicos que permiten obtener o cambiar el valor de la propiedad.
- Los métodos que acceden a un valor (*accesors*) tienen la forma `getXXX()`. Los métodos que permiten cambiar la propiedad (*mutators*) tienen la forma `setXXX()`. Donde XXX se refiere al nombre de la propiedad.



Ejemplo

```
public class Alumno {  
    private long numCta;  
    private String nombre;  
  
    public long getNumCta(){  
        return numCta;  
    }  
    public void setNumCta(long cta){  
        numCta = cta;  
    }  
  
    public String getNombre(){  
        return nombre;  
    }  
    public void setNombre(String n){  
        nombre = n;  
    }  
}
```



5.2 JavaBeans dentro de JSPs

Hay tres acciones que pueden implementar los JavaBeans en los JSPs.

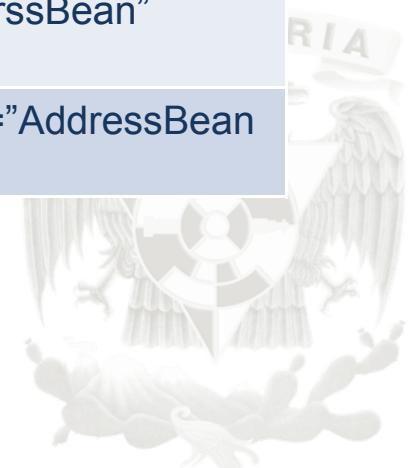
- **<jsp:useBean>**Declara el uso del Javabean en una página JSP.
- **<jsp:setProperty>**Envío de valores a las propiedades de los JavaBeans.
- **<jsp:getProperty>**Obtención de los valores de las propiedades del JavaBean.



Etiqueta <jsp:useBean>

La etiqueta jsp:useBean declara las variables en el JSP y asocia la instancia con un JavaBean. Esta acción posee cinco atributos.

Atributo	Descripción	Ejemplo
id	Especifica el nombre con el que el bean es identificado en el JSP.	Id="direccion"
scope	Define el alcance del bean.	scope="session"
class	Define la clase Java que contiene al bean.	class="AddressBean"
type	Especifica el tipo de la variable que será usada para referirse al bean.	type="AdderssBean"
beanName	Define nombre del bean	beanName="AddressBean" "



Dados los atributos anteriores, id es forzoso, scope es opcional y class, type y beanName deben de ser usadas bajo las siguientes combinaciones:

- class
- type
- class y type
- beanName y type



```
<jsp: useBean id="NombreInstancia"  
scope="page|request|session|application"  
{  class=" paquete.clase " |  
   type=" paquete.clase " |  
   class=" paquete.clase " [ type=" paquete.clase " ] |  
   beanName="{paquete.clase | <%= expresión %>}" type="package.class"  
}  
  
{  />| > otros elementos </ jsp: useBean>  
}
```



```
<jsp:useBean id="miContador" class="HitCountBean" scope="session"/>
```

Lo anterior etiqueta es equivalente a:

```
HitCountBean miContador = (HitCountBean)
session.getAttribute("miContador ");
if (miContador == null) {
    miContador = new HitCountBean ();
    session.setAttribute("miContador ", miContador);
}
```



- Inicializando las propiedades del bean

Dado que el Javabean es instanciado por el motor de JSPs usando un constructor sin argumentos, para inicializar las propiedades del bean se realiza lo siguiente:

```
<jsp:useBean id="address" scope="session"  
class="AddressBean" >  
  <%  
    address.setCalle("Av. principal #23");  
  %>  
</jsp:useBean>
```



Etiqueta <jsp:setProperty>

Esta etiqueta se utiliza para traer los valores de las propiedades del bean.

Atributo	Descripción
name	El nombre del bean con que se va identificar dentro del JSP.
property	El nombre de la propiedad del bean.
value	El nuevo valor asignado a la propiedad.
param	El nombre del parámetro disponible en HttpServletRequest, que será asignado como nuevo valor de la propiedad del bean.



A continuación se muestra un ejemplo del uso de la etiqueta setProperty utilizando el atributo value:

```
<jsp:setProperty name="dir" property="ciudad"  
value="León"/>  
  
<jsp:setProperty name="dir" property="estado" value="Gto"/>
```

El código anterior es equivalente a:

```
<%  
    address.setCiudad("León");  
    address.setEstado("Gto");  
%>
```



A continuación se muestra un ejemplo del uso de la etiqueta `setProperty` utilizando el atributo `param`:

```
<jsp:setProperty name="dir" property="ciudad"  
param="miCiudad"/>  
  
<jsp:setProperty name="dir" property="estado"  
param=miEdo/>
```

Donde el valor asignado a la propiedad del bean proviene de la petición. Lo anterior es equivalente a:

```
<jsp:setProperty name="dir" property="city"  
value="<% = theCity %>" />
```

Dentro de un scriptlet, el siguiente código es equivalente:

```
<%  
    address.setCiudad(peticion.getParameter("miCiudad"));  
    address.setEstado(peticion.getParameter("miEdo"));  
%>
```



Las anteriores técnicas se utilizan cuando los nombres de los parámetros que se obtienen de la petición no encajan con las propiedades de los beans. De lo contrario se puede utilizar lo siguiente:

```
<jsp:setProperty name="dir" property="ciudad"  
/>  
<jsp:setProperty name="dir" property="estado"
```

El código anterior es equivalente a:

```
<jsp:setProperty name="dir" property="ciudad"  
param="ciudad" />  
<jsp:setProperty name="dir" property="estadi" param="estado"  
/>
```

Cuando los nombres de todos los parámetros enviados a través de la petición son iguales a los de las propiedades del JavaBean, es posible enviar todas las propiedades al mismo tiempo.

```
<jsp:setProperty name="dir" property="*"
/>
```



Etiqueta <jsp:getProperty>

Esta etiqueta permite obtener los valores de las propiedades del bean. La sintaxis es la siguiente:



EJEMPLO 5.1



EJEMPLO 5.2



6

Integración de JDBC, Servlets y JSP's



6. Integración de JDBC, servlets y JSP's

6.1 Integración de JSPs y servlets con JDBC

6.2 Transacciones



6.1 Integración de JSPs y Servlets con JDBC

El siguiente paso consiste en unir todas las herramientas que ya se poseen con un fin común.

Entonces, por un lado se posee el archivo java que realiza la conexión con la base de datos. Por otro lado, se posee un JSP que hace una llamada a la clase java (para que ésta se conecte a la BD).

A continuación se muestra un ejemplo para realizar la unión entre el JSP y la clase.



EJEMPLO 6.1



6.2 Transacciones

Una conexión, por defecto, funciona en modo autocommit, es decir, cada vez que se ejecuta una sentencia SQL se abre y se cierra automáticamente una transacción que sólo afecta a dicha petición.

Es posible modificar esta opción mediante el método `setAutoCommit()`. También se puede consultar el valor que posee autocommit mediante el método `getAutoCommit()`.



Si no se está trabajando en modo autocommit será necesario que se cierren explícitamente las transacciones mediante commit() si las consultas tienen éxito o rollback() si las consultas fallan.

Después de cerrar una transacción, la próxima vez que se ejecute una sentencia SQL se abrirá automáticamente una nueva, por lo que no existe ningún método que permita iniciar una transacción.

```
con.setAutoCommit(false);
```



Una vez que se han deshabilitado los commits, las sentencias SQL se ejecutan hasta que se realiza la llamada explícita al método commit(). Todas las sentencias que se hayan realizado antes de llamar al método commit(), se van a realizar al momento de hacer la invocación del mismo.



```
// Se desactiva auto-commit  
conn.setAutoCommit(false);  
// se realiza una actualizaciOn  
PreparedStatement act1 = conn.prepareStatement(  
    "UPDATE animal SET nombre = ? WHERE id = ?");  
act1.setString(1, "Tiger...");  
act1.setInt(2, 3);  
res[0] = act1.executeUpdate();  
// Se realiza otra actualizaciOn  
PreparedStatement act2 = conn.prepareStatement(  
    "UPDATE animal SET nombre = ? WHERE id = ?");  
act2.setString(1, "Venaditou");  
act2.setInt(2, 1);  
res[1] = act2.executeUpdate();  
// se ejecutan las consultas  
conn.commit();  
// se vuelve a activar auto-commit  
conn.setAutoCommit(true);
```



Desde JDBC 3.0 se implementó el método `setSavePoint()` de la clase `Connection` que permite guardar un punto específico en una transacción.

```
stm = conn.createStatement();
conn.setAutoCommit(false);
res[0] = stm.executeUpdate("INSERT INTO animal
(id_especie,id_tipo,nombre,edad)
VALUES ('4','4','Gema','5')");
// Se salva un punto específico
Savepoint sp = conn.setSavepoint("SAVEPOINT_1");
// Se genera otra inserción
res[1] = stm.executeUpdate("INSERT INTO animal
(id_especie,id_tipo,nombre,edad)
VALUES ('4','4','Dorado','2')");
// Se hace un rollback hasta el punto de salvamento
conn.rollback(sp);
conn.commit();
conn.setAutoCommit(true);
```



En el ejemplo anterior se realiza una inserción de una fila dentro de una tabla. Posteriormente se guarda un punto y después se inserta otra fila.

Al realizar la última transacción se deshace la inserción de la segunda fila, pero la primera inserción se mantiene intacta y, por lo tanto, sólo se realiza la primera inserción.



EJEMPLO 6.2



7

Arquitectura Modelo-Vista-Controlador (MVC)



7. Arquitectura Modelo-Vista-Controlador (MVC)

7.1 Patrones

**7.2 Definición del patrón
MVC**



7.1 Patrones

Un patrón es un conjunto de información que aporta la solución a un problema que se presenta en un contexto determinado. Para elaborarlo se aíslan sus aspectos esenciales y se añaden cuantos comentarios y ejemplos sean necesarios. En particular se debe identificar:

- El contexto en el que es posible aplicar el patrón.
- Las fuerzas (objetivos y restricciones) que intervienen en ese contexto.
- El diseño a aplicar para equilibrar objetivos y restricciones.



Las características de un buen patrón son:

- **Resuelve un problema cuya solución no es obvia.**
- **Ha sido revisado por una comunidad de desarrolladores.**
- **Ha sido experimentado en la práctica.**
- **Muestra como equilibrar restricciones y objetivos.**



Los patrones son importantes por varios motivos:

- Encapsulan conocimiento detallado sobre un tipo de problema y sus soluciones.
- Proporcionan un vocabulario común.
- Estimula la reutilización del software.
- Al aplicar soluciones probadas, se evitan los riesgos y se ahorra el esfuerzo de volver a implementar soluciones.



7.2 Definición del patrón MVC

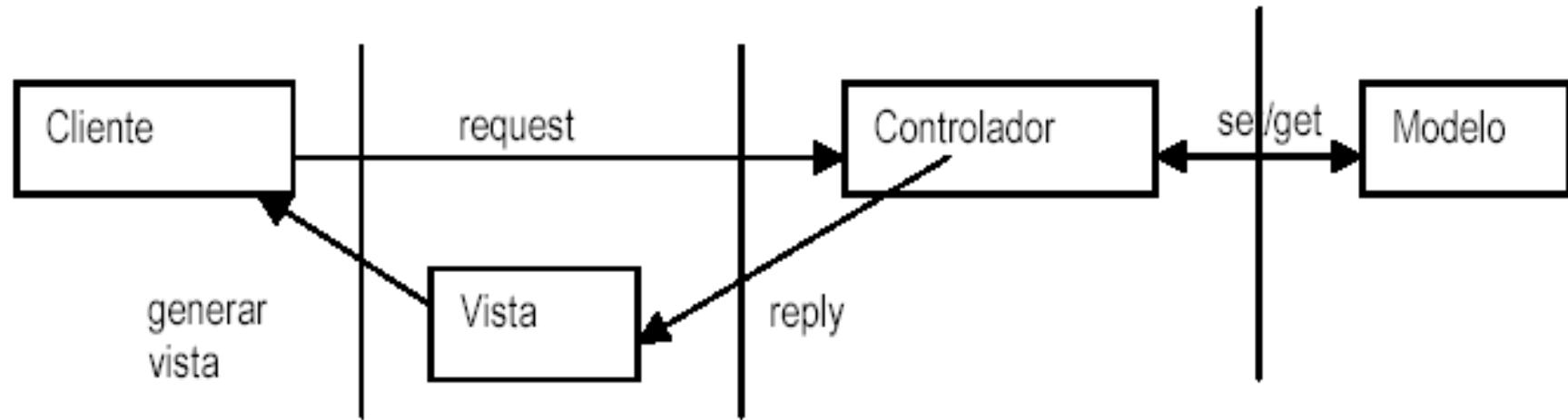
El Modelo-Vista-Controlador permite aislar tanto los datos de la aplicación, como el estado (modelo) de la misma y el mecanismo utilizado para representar (vista) dicho estado, así como para hacer modular la vista y poder modelar la transición entre estados del modelo (controlador).



Las capas del patrón MVC se dividen en tres grandes áreas funcionales:

- **Vista:** La presentación de los datos al usuario final.
- **Controlador:** El que atenderá las peticiones y componentes para toma de decisiones de la aplicación. Controla la transición entre el procesamiento de los datos y su visualización.
- **Modelo:** Es la lógica del negocio o servicio y los datos asociados con la aplicación, representa sus datos y comportamientos.





EJEMPLO 7.1

