Tutorial JSP

JavaServer Pages

Tutorial de JavaServer Pages
Autor: ***Miguel Angel García***
E-Mail: webmaster@verextremadura.com

Presentación

Este tutorial de páginas JSP, está basado en textos extraídos de diferentes artículos y en mi experiencia profesional en el mundo de las páginas JSP, Servlets y Beans de Java.
El motivo que me lleva a escribirlo es porque no he encontrado ni una sóla página en español que trate a nivel básico-medio este tema.

Lo Nuevo

- Otros links de JSP
- Debido a la excesiva demanda de peiticiones sobre como hacer paginación de registros de una consulta, me he puesto manos a la obra y estoy acabando una página sobre ese tema. Por favor, ten paciencia.

Contenido del tutorial

- Parte I Introducción a JSP (4 Dic. 2.000)
  1. Introducción a JSP.
  2. Resumen de la arquitectura de una página JSP.
  3. Una página JSP simple.
  4. Elementos de una página JSP.
- Parte II Servlets y JavaBeans (5 Dic. 2.000)
  1. Introducción a los Servlets.
  2. Ciclo de vida de un Servlet.
  3. Un Servlet Básico.
  4. Contruyendo e instalando el Servlet Básico.
  5. Que es un JavaBean.
- Parte III Recuperando datos desde un Servlet (5 Dic. 2.000)
  1. Obteniendo los datos de un formulario web.
- Parte IV Acceso a Datos
  1. Que es JDBC.
  2. Configurar la conexión a la Base de Datos
  3. Ejecutando comandos SQL.
- Parte V Paginar registros de una consulta (28 Mar 2.001)
  1. ¿ Que es la paginación ?
  2. Formas de hacer la paginación con JSP
  3. Ejemplo de paginación simple
- Otras páginas sobre JSP
  1. www.jspin.com Muy bueno (Ingles)
  2. www.jsptut.com Bastante bueno (Ingles)

Tutorial JSP

JavaServer Pages

Parte I Introducción a JavaServerPages

Indice del tema:

?? Introducción
?? Resumen de la arquitectura de una página JSP
?? Una página JSP simple
?? Elementos de una página JSP

Java***S***erver ***P***ages (JSP) combinan HTML con fragmentos de Java para producir páginas web dinámicas.
Cada página es automáticamente compilada a servlet por el motor de JSP , en primer lugar es recogida y a continuación ejecutada.
JSP tiene gran variedad de formas para comunicarse con las clases de Java, servlets, applets y el servidor web; por esto se puede aplicar una funcionalidad a nuestra web a base de componentes.

Una página JSP es archivo de texto simple que consiste en contenido HTML o XML con elementos JSP. Cuando un cliente pide una página JSP del sitio web y no se ha ejecutado antes, la página es inicialmente pasada al motor de JSP, el cual compila la página convirtiéndola en Servlet, la ejecuta y devuelve el contenido de los resultados al cliente.
Es posible ver el código del servlet generado, este código debe estar en el directorio que se informa en la estructura de directorios del servidor.
Si nos fijamos en este archivo podemos encontrar las siguientes clases:
· JSPPage
· HttpJspPage
Ellas definen la interface para el compilador de páginas JSP.
Nos encontramos también tres métodos:
· JspInit()
· JspDestroy()
· _jspService(HttpServletRequest request, HttpServletResponse response)
Los dos primeros métodos pueden ser definidos por el autor de la página JSP, pero el tercer método es una versión compilada de la página JSP, y su creación es responsabilidad del motor de JSP.

Es el momento de ver el código de una página JSP bastante simple.

```
<HTML>
<HEAD>
<TITLE>Página simple JSP</TITLE>
</HEAD>
<BODY>
      <%// INFORMACION GLOBAL PARA LA PAGINA%>
      <%@ page language=”java” %>
      <% // DECLARACION DE VARIABLES %>
      <% char c = 0;%>
      <%
            for (int i=0;i < 26;i++)
            {
                  for (j = 0;j < 26;j++)
                  {
                        c = (char) (0x41 + (26 – i +
j)%26); %>
                        <%// IMPRIME LAS LETRAS DEL
```

```
ALFABETO EN MAYUSCULAS %>
                                <%=c%>
                        <%}
                }
        %>
</BODY>
</HTML>
```

El código fuente de una página JSP incluye:

1. Directivas: Dan información global de la página, por ejemplo, importación de estamentos, página que majena los errores o cuando la página forma parte de una sesión, en el ejemplo anterior informamos del tipo de script de Java.
2. Declaraciones: Sirven para declarar métodos y variables.
3. Scripts de JSP: Es el código Java embebido en la página.
4. Expresiones de JSP: Formatea las expresiones como cadenas para incluirlas en la página de salida.

Estos elementos siguen una sintaxis como XML, así se obtiene un significado con una presentación totalmente separada de la lógica.Un buen ejemplo es `<jsp:useBean .../>` el cual busca o crea una instancia de un bean. Con el mecanismo de extensiones de tag se tiene la posibilidad de definir tags con acciones similares y poner la funcionalidad en una librería de tags.

1. Directivas

Una directiva de JSP es una estamento que proporciona la información del motor de JSP para la página que la pide. Su sintaxis general es `<%@ directiva {atributo ="valor"} %>` dónde la directiva debe tener un número de atributos. Cada directiva tiene un XML opcional equivalente, pero esto son intentos para una futura herramienta JSP, por esto no lo consideraremos ahora.

Posibles directivas en JSP 1.0 son:

Page: Información para la página.

Include: Incluye archivos completos palabra por palabra.

Taglib: La dirección de la librería de tags que se usará en la página.

Tal y como esperabamos, la directiva Page posee varios atributos.

Atributos y posibles valores	Descripción
language="java"	Comunica al servidor el lenguaje que va a ser utilizado en el archivo. Java es el único posible es esta especificación
extends="package.class"	La variale extends, define la clase padre del servlet generado. Normalmente no es necesario utilizar otras que no sean las clases base del proveedor.
import="package.,package.class"*	Sirve para especificar los paquetes y clases que se quieran utilizar.
session="true|false"	Por defecto session vale true, manteniendo los datos de las sesión para la página.
isThreadSafe="true|false"	Por defecto vale *true*, le hace señales al motor de JSP para que multiples pedidos del cliente puedan ser tomadas como una.
info="text"	Información en la página a la que puede accederse a través del método Servlet.getServletInfo()
errorPage="pagina_error"	Página que manejará las excepciones de errores.
isErrorPage="true|false"	Marca a la página como la página que manejará

	los errores

2. Declaraciones

Una declaración de JSP, puede definirse como una definición de variables y métodos a nivel de clase que son usadas en la página.
Un bloque de declaraciones típico sería <%! declaración %>
Un ejemplo de declaración de script sería el siguiente:

```
<HTML>
<HEAD>
<TITLE>Página simple JSP</TITLE>
</HEAD>
<BODY>
      <%! String strCadena = "x";
          int intContador = 0;
      %>

</BODY>
</HTML>
```

3. Scripts de JSP

Los Scripts son bloques de código Java residentes entre los tags <% y %>.
Este bloques de código estarán dentro del servlets generado incluídos en método _jspService().
Los Scripts pueden acceder a cualquier variable o Beans que haya sido declarado. También hay algunos objetos implícitos disponibles para los Scripts desde entorno del Servlet. Vamos a verlos a continuación.

Objetos implícitos	Descripción
request	Es la petición del cliente. Es normalmente una subclase de la case HttpServletRequest.
response	Es la página JSP de respuesta y es una subclase de HttpServletResponse.
pageContext	Los atributos de la página y los objetos implicitos necesitan ser accesibles a través de API, para permitir al motro de JSP compilar la página. Pero cada servidor tiene implementaciones específicas de cada uno de esos atributos y objetos. Para solucionar este problema, el motor de JSP utilizar la clase Factory para devolver la implementación de clase PageContext del servidor. Esta clase PageContext es inicializada con los objetos response y request y algunos atributos de la directiva de la página (erropage, session,buffer and autoflush) y facilita los otros objetos implícitos para la pagina de petición.Veremos más más adelante.
session	El objeto de sesión HTTP asociado a la petición.
application	Lo que devuelve el servlet cuando se llama a getServletConfig().getContext()
out	El objeto que representa la salida de texto por pantalla.
config	El objeto ServletConfig de la página.
page	Es la forma que tiene la página para referirse a si misma. Se usa como alternativa al objeto this
exception	Es una subclase libre de Throwable que es pasada a la página que majena los errores.

El siguiente fragmento de código muestra como obtener el valor de una parámetro mediante el objeto request, y como pasarlo a una cadena para mostrarlo en pantalla.

```
<%
   String strNombre  = request.getParameter("nombre");
   out.println(strNombre);
%>
```

4. Expresiones de JSP

Las expresiones son una magnifica herramienta para insertar código embebido dentro de la página HTML. Cualquier cosa que este entre los tags <%= y %> será evaluado, convertido a cadena y posteriormente mostrado en pantalla. La conversión desde el tipo inicial a String es manejada autómaticamente.

Es importante remarcar que que la expresión no termina en punto y coma (;) . Esto es así porque motro de JSP, pondrá la expresión automáticamente entre out.println().

Las expresiones JSP te permiten parametrizar las páginas HTML (es parecido a cuando parametrizas una consulta SQL pero difieren la forma de los valores). Una y otra vez , en el código de la página HTML, ser verán bucles o condiciones usando código Java, simplemente empezando y acabando las condiciones o bucles entre los tags <% y %>. Un ejemplo sería:

```
<% for (int i=0;i<5;i++) { %>
    <BR>El valor del contador es <%=i%>
<% } %>
```

Tutorial JSP

JavaServer Pages

Parte II Servlets y JavaBeans

Indice del tema:

- ?? Introducción a los Servlets
- ?? Ciclo de vida de un servlet
- ?? Un Servlet Básico
- ?? Contruyendo e instalando el Serlvet Básico
- ?? Qué es un JavaBean

La herramienta más importante que se usa a la hora de desarrollar web con Jsp son los Servlets; los servlets son la primera línea de batalla del desarrollo de las aplicaciones web. Estos aportan una manera facil para que nuestro servidor se comunique con el lado cliente.
Los servlets dan un modelo general de clases para ejecutar servicios. Al más básico nivel, este es la definición de un servidor.
Tanto la API de los servlets como el motor de servlets pueden ser bajados como parte de Java Servlet Development Kit (JSDK) en la página web de Sun.

Contenedor de Servlet

El contenedor de servlets tiene que encargarse, entre otras cosas, de pasar las peticiones del cliente al servlet y este último de devolver la respuesta solicitada al cliente. La implementación actual del servlet difiere de un programa a otro, pero la interface entre contenedor del servlet y el servlet se especifica en la API del servlet.
Basicamente, el ciclo de vida de un servlet es como se detalla a continuación:

- ?? El contenedor de servlet crea una instancia del servlet
- ?? El contenedor llama al método init() del servlet
- ?? Si el contenedor tiene una petición para el servlet, se llama al método service()
- ?? Después de destruir la instancia, el contenedor llama al método destroy()
- ?? La instancia es destruída y marcada como una colección desechada.

El ciclo de vida de un Servlet tiene un diseño muy simple orientado a objetos. Un Servlet es construído e inicializado, después se procesan cero o varias peticiones y por último se destruye. En este punto el servlet es totalmente destruído y es una colección de deshechos. Este diseño explica porque un Servlet reemplaza perfectamente a un CGI. El servlet es cargado un sóla vez y está residente en memoria mientras se procesan las respuestas,

La interface que define esta estructura es javax.servlet.Serlvet. La interface del Serlvet define lo métodos del ciclo de vida. Estos métodos son *Init()*, el método *service()* y el método *destroy()*;

Init()

En el método Init() es dónde empieza la vida de un servlet. Es llamado inmediatamente después de ser instanciado. Y es llamado un sólo vez. El método Init() crea e inicializa los recursos que serán usados mientras se manejan las peticiones. Este método tiene la siguiente forma:

```
public void init(ServletConfig config) throws ServletException;
```

Service()

El método Service() majena las peticiones enviadas por el cliente.No pueden dar comienzo los servicios de las peticiones hasta que el método Init() no ha sido ejecutado.La implementación

más habitual del metódo Service() está en la clase HttServlet.
La forma del método Services es como sigue:

```
public void service(ServletRequest peticion, ServletResponse respuesta)
throws ServletExpetion, IOException;
```

El método service() implementa el paradigma de la respuesta y de la petición. El objeto ServletRequest contiene información sobre la petición del servicio e información encapsulada que proporciona el cliente. El objeto ServletResponse contiene la información que se devuelve al cliente.

Destroy()

Este método significa el final de la vida de un Servlet. Cuando un servicio se finaliza se llama al método Destroy(). Este método es dónde todos los recursos creados en el método Init() deben ser limpiados. Por ejemplo es el lugar dónde, si tenemos una conexión a una base de datos, debe cerrarse. También es el lugar dónde debe guardarse información persistente en caso de que la utilicemos en algún otro servicio. Esta es la forma que tiene el método Destroy():

```
public void destroy();
```

A continuación se muestra la estructura que tiene un Servlet Básico.

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

public class ServletBasico extends HttpServlet{
  public void init(ServletConfig config) throws SerlvetException {
   // Siempre se pasa el objet ServletConfig a la superclase
     super.init(config);
}
// Proceso HTTP Get de la petición
public void doGet(HttpSerlvetRequest peticion,HttpserlvetResponse respuesta)
                  throws SerlvetException, IOException {
     respuesta.setContentType("text/html");
     PrintWriter out = respuesta.getWriter();
     out.println("<html>");
     out.println("<head><title>Servlet Basico</title></head>");
     out.println("<body>");
     // Imprime en pantalla el método enviado por la petición
     out.println("El método de la petición es" + peticion.getMethod()+"\n");
     out.println("</body></html>");
     out.close();
}

// Proceso HTTP Post de la petición
public void doPost(HttpSerlvetRequest peticion,HttpServletResponse respuesta)
                   throws ServletException,IOException {

     respuesta.setContentType("text/html");
     PrintWriter out = respuesta.getWriter();
     out.println("<html>");
     out.println("<head><title>Servlet Basico</title></head>");
     out.println("<body>");
     // Imprime en pantalla el método enviado por la petición
     out.println("El método de la petición es" + peticion.getMethod()+"\n");
     out.println("</body></html>");
     out.close();
}
```

```
// Devuelve la información del Servlet
public String getServletInfo(){
     return "Información del Servlet de ejemplo básico";
}
```

En este Servlet, se han pasado por encima métodos también muy importantes, pero debido a la naturaleza básica de este tutorial no los comentaremos en detalle al menos de momento, pero si merece la pena la menos nombrarlos.

Método	Descripción
getAuthType	Devuelve el equema de auténtificación de la petición.
getCookies	Devuelve un array de Cookies encontrados en la petición.
getHeader()	Devuelve las especificaciones del campo de cabecera de la petición.
Hay otros muchos métodos, como getServetPath, getQueryString(), getSession(), getMethod() etc, que de momento no comentaremos.	

Existen dos pasos necesarios para completar la ejecución de todo servlet:

1. Compilar el Servlet.
2. Colocarlo en un lugar dónde Java Web Server pueda econtrarlo.

Para compilar el Servlet puede hacerse mediante cualquier entorno de programación de Java, como puede ser *JBuilder* o bien mediante *JSDK*. Hay que asegurarse que el *SDK* de Servlet está en el *CLASSPATH*. Si se tiene instalado en Java Web Server puede encontrarse en el archivo *lib/jws.jar*.
Después de compilar, lo primero que debe hacer es poner el archivo generado .class en el *CLASSPATH*. La forma más sencilla de hacer esto es moverlo a la carpeta */servlet*.
Por ultimo para llamar al servlet debe hacerse de la siguiente manera:
http://localhost:8080/servlet/ServletBasico o mejor aún creando una simple página web que invoque al método *doGet()* o al método *doPost()*.
Esta sería la página web sencilla que llamaría al servlet:

```
<html>
<head>

<title>Servlet Básico</title>

</head>
<body>
<form action="http://localhost:8080/servlet/ServletBasico"
      method="post">

      <input type="submit" value="Enviar" name="btn_enviar">
</form>

</body>

</html>
```

Cuando analizamos el desarrollo de la arquitectura de una aplicación que envuelve JavaSerPages, es una buena idea intentar poner toda la lógica de negocio en componentes reutilizables. Estos componentes pueden ser insertados dentro de una página JSP cuando sean requeridos.

El lenguage Java implementa la idea de componentes con los llamados JavaBeans. Un JavaBean es una clase de Java que se adapta a los siguientes criterios:

- ?? Clase publica.
- ?? Contructor público sin argumentos.
- ?? Posee métodos publicos "Set" y "Get" con el fin de simular propiedades. El método "Get" no tiene argumentos al menos que haga función de propiedad indexada.

Las propiedades son siempre colocadas y recuperadas utilizando una conveción denominada común. Para cada propiedad, deben existir dos métodos, uno getxxx() y otro setxxx() dónde xxx es el nombre de la propiedad.
Ya sabemos que un Bean es como cualquier otra clase de Java y tipicamente un Bean es incorporado o insertado dentro de un programa, sus propiedades son colocadas y sus métodos llamados.
Casi en todos los casos los Beans son usados para encapsular elementos de la GUI visuales y no visuales. Hay muchas maneras elegantes y eficientes de enlazar un Bean para implementar drag-and-drop y salvar el estado de un Bean entre instancias.
La meta final de los JavaBeans, es tener una herramienta gráfica en nuestro código, pero si no pensamos en ellos como algo meramente gráfico, el poder entenderlos nos resultará más facil.
Por esta historia, los JavaBeans, no tiene sitio en JSP, al menos no si los tenemos en cuenta con el objetivo para lo que fueron diseñados. Si pensamos en ellos como componentes, como simple encapsulación de código Java, entonces su propósito está más claro. Los Beans hacen que nuestras páginas no estén aisladas.
Mucha de la lógica de negocio que hayamos podido pensar estaría mejor colocada en un Enterprise JavaBean, dónde las transacciones y el escalado son problemas del contenedor y no del Bean. Estos pesos pesados son necesarios algunas veces pero pienso que los JavaBeans gráficos están por encima del trabajo sencillo que estamos viendo en este tutorial.

Indice del tema:

?? Obteniendo los datos de un formulario web

Ahora veremos como se recuperan los datos de un formulario típico de cualquier página web.
Los servlets normalmente reciben los datos de entrada a través de las peticiones POST o GET.
Los métodos que se usan para recuperar los datos son los mismo en cada caso.
Los tres métodos usados para recuperar los parametros son getParameterNames(), getParameter() y getParameterValues() y tiene la siguiente definición:
public Enumeration ServletRequest.getParameterNames();
public String ServletRequest.getParameter(String nombre);
public String ServletRequest.getParameterValues(String nombre);
getParameterNames() devuelve los nombres de los paramétros de la petición como una enumeración de cadenas, o una enumeración vacía si no hay parámetros en la petición. Se usa como un método soportado de getParameter(). Cuando tienes una enumeración de la lista de los nombres de los parámetros, puedes iterar con ellos, llamando al método getParameter() con cada uno de los nombres de la lista.
getParameter() devuelve una cadena que contiene el valor simple del parámetro especificado, o null si el parámetro no existe en la petición. Este método, aunque es el más utilizado, sólo debería se usado si estás totalmente seguro de que existe el parámetro que se va a recoger. Si el paramétro tiene multiples valores debe utilizarse getParamterValues().
getParameterValues() devuelve los valores de los parámetros especificados como una array de caracteres, o null si si no existe el parámetro en la petición.
Vamos a ver un ejemplo de un servlet que implementa los casos anteriores:

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

public class EjemploRecuperarDatos extends HttpServlet{

  public void init(ServletConfig config) throws SerlvetException {
   // Siempre se pasa el objet ServletConfig a la superclase
     super.init(config);
}
// Proceso HTTP Get de la petición
public void doGet(HttpSerlvetRequest peticion,HttpserlvetResponse respuesta)
                  throws SerlvetException, IOException {
     doPost(peticion, respuesta);
}

// Proceso HTTP Post de la petición
public void doPost(HttpSerlvetRequest peticion,HttpServletResponse respuesta)
                   throws ServletException,IOException {

     respuesta.setContentType("text/html");
     PrintWriter out = respuesta.getWriter();
     out.println("<html>");
     out.println("<head><title>Ejemplo de Recuperar Datos</title></head>");
     out.println("<body>");

     // Obtiene todos los nomnbres de los parámetros
     Enumeration parametros = request.getParameterNames();
```

```
        String parametro = null; // variable para guardar el parámetro.

        // Obteniendo los parametros pasados con la petición.
        while (parametros.hasMoreElements()){
           parametro = (String)parametros.nextElement();
           out.println("<B>"+ parametro + ":" + peticion.getParameter(parametro)
+ "<B><BR>");
        }

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

    // Devuelve la información del Servlet
    public String getServletInfo(){
        return "Información del Servlet de Recuperación de Parametros";
    }
```

Este serlvet, también debe ser compilado y colocado en el directorio adecuado para que Java Web Server pueda encontrarlo.
Ahora veremos como hacer la página web que contiene el formulario con los datos y cómo llamar al servlet anterior.

```
<html>
<head>

<title>Servlet Básico</title>

</head>
<body>
<form action="http://localhost:8080/servlet/EjemploRecuperarDatos"
      method="POST">
      Nombre: <input name="txt_Nombre" type="text">
      Apellidos: <input name="txt_Apellidos" type="text">
      Clave: <input name="txt_Clave" type="password">
      <input type="submit" value="Enviar" name="btn_enviar">
</form>

</body>

</html>
```

Después de esto sólo nos queda probar la llamada al servet, y la ejecución del mismo.

Indice del tema:

?? Qué es JDBC
?? Configurar la conexión a la Base de Datos
?? Ejecutando comandos SQL

JDBC es una API pura de Java que se usa para ejecutar comandos de SQL. Suministra una serie de clases e interfaces que permiten al desarrollador de web escribir aplicaciones que gestionen Bases de Datos.
La interacción típica con una base de datos consta de los siguientes cuatro pasos básicos:

?? Abrir la conexión a la base de datos
?? Ejecutar consultas contra la base de datos
?? Procesar los resultados
?? Cerrar la conexión a la base de datos

Y sin más demora, vamos a ver como se usan los cuantro pasos anteriores:

```
// Paso 1. Abrir la conexión a la base de datos.
 Connection conexion =
DriverManager.getConnection("jdbc:odbc:Nombre_ODBC","usuario","password");


// Paso 2. Ejecutar consultas a la base de datos.
 Statement Estamento = conexion.creStatement();
 ResultSet rs = Estamento.executeQuery("select dni,nombre,apellidos,edad from
agenda");


// Paso 3. Procesar los resultados. En este caso los muestra en pantalla.
 while (rs.next()) {
    out.println("DNI ->"+ rs.getString("dni"));
    out.println("NOMBRE ->"+ rs.getString("nombre"));
    out.println("APELLIDOS ->"+ rs.getString("apellidos"));
    out.println("EDAD ->"+ rs.getInt("edad"));
    }

// Paso 4 . Cerrar la conexión a la base de datos.
 rs.close();
 Estamento.close();
 conexion.close();
```

Nota: Mas adelante explicaremos cada una de las instrucciones vistas en este fragmento de código.

Lo primero que se debe hacer para poder atacar a bases de datos con páginas JSP o con Servlets, es instalar el Driver correcto de la base de datos que se desee utilizar.
Por ejemplo, supongamos que tenemos una base de datos Microsoft Access. Para instalar el puente JDBC-ODBC, simplemente bajamos el JDK y seguimos la directrices de instalación del mismo. El puente JDBC-ODBC está incluído en JDK desde la versión 1.1.

Configurando ODBC para Access

El puente JDBC-ODBC no necesita pasos específicos para su instalación, pero ODBC si. Por ejemplo si asumimos que estamos utilizando un máquina con Windows/9x o NT. necesitaremos

configurar nuestra conexión mediante ODBC, si has llegado hasta aquí estoy seguro que alguna vez has configurado una conexión mediante ODBC a cualquier base de datos, ya sea Access, Oracle o SQL Server, así que la configuración de la conexión de ODBC, no vamos a detallarla demasiado, sólo informaré de los pasos básicos, que se ven en la siguiente lista:

1. Ir al Panel de Control.
2. Fuentes de datos ODBC.
3. Agregar.
4. Seleccionar Driver *Controlador para Microsoft Access (*.mdb).*
5. Facilitar a la conexión ODBC las características básicas de la base de datos.

Mediante estos pasos suponemos que ya hemos configurado el ODBC.

Estableciendo la conexión

La primera cosa que debe hacerse para trabajar con bases de datos es establecer la conexión con la base de datos, para ello tenemos dos simples pasos, cargar el driver y establecer la conexión. Cargar el driver es una operación muy simple que requiere sólo una única línea de código.Si vamos a cargar una fuente ODBC utilizaremos una clase llamada sun.jdbc.odbc.jdbcOdbcDriver y la instrucción de código quedaría así

```
Class.forName("sun.jdbc.odbc.JdbcOdbDriver);
```

Cuando se llama a Class.forName, lo que se está haciendo es crear una instancia al driver y registrarlo con el DriverManager. Esto es lo único que hay que hacer para cargar el Driver Después de esto, necesitamos esblacer una conexión a la base de datos mediante el ODBC que hemos creado anteriormente, para ello bastaría con esta línea de código.

```
conexion =
DriverManager.getConnection("jdbc:odbc:NombreODBC","nombre_usuario","clave");
```

La conexión a la base de datos, ya está realizada, esto es debido al método estático DriverManager.getConnection(); que lo que hace es devolver una conexión a la base de datos. Los paramétros son tres, uno es la URL dónde se encuentra la base de datos, y los otros se explican por si sólo, el segundo es el nombre de usuario y tercero es el password para poder acceder a la base de datos.
Imaginemos ahora que tenemos una base de datos ***ORACLE*** llamada **BASE** en un servidor cuya dirección HTTP es ***171.10.10.1*** en el puerto ***1515***.El nombre de usuario es ***miguel*** y la contraseña es ***gato***.
En este caso la conexión se haría de la siguiente manera:

```
String dbURL = "jdbc:oracle:thin:@171.10.10.1:1515:BASE";
String usuario = "miguel";
String pwd = "gato";
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection conexion = DriverManager.getConnection(dbURL,usuario,pwd);
```

Con estas instrucciones tan simples y teniendo el diver adecuado de Oracle se haría una conexión a una base de datos ORACLE.

Para ejecutar cualquier instrucción SQL contra la base de datos debemos usar el objeto estamento. Para crearlo tiene que llamarse al método Connection.createStatements(). Este devolverá un estamento que sirve para enviar las consultas a la base de datos. Con la siguiente instrucción se crea un estamento.

```
Statement estamento = conexion.createStatement();
```

Para los que estén familiarizados con Visual Basic, podrán enterderlo mejor si lo consideran como si fuera el objeto Workspace
El paso siguiente es ejecutar instrucciones SQL, estas instrucciones pueden ser cualquiera que nos haga falta para gestionar nuestra base de datos. Por ejemplo, gracias al Estamento, podemos ejecutar instrucciones como SELECT, UPDATE o DELETE. Veamos un ejemplo completo:

```
     import java.sql.*;
     public class CrearTablaAgenda{
         public void CrearTablas(){
           Connection conexion = null;
           String strSQL = "";
           try {

                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                conexion =
    DiverManager("jdb:odbc:NombreODBC","nombre_usuario","clave");
                Statement Estamento = conexion.createStatement();
                strSQL = "CREATE TABLE AGENDA (DNI VARCHAR(5),NOMBRE
    VARCHAR(30),APELLIDOS VARCHAR(30))";
                Estamento.executeUpdate(strSQL);

           } catch (SQLException excepcion_SQL) {

                    System.err.println(excepcion_SQL.getMessage());

           } catch (ClassnotFoundException excepcion_ClassForName) {

                    System.err.println(excepcion_ClassForName.getMessage());


           } catch (Exception excepcion_general) {
                    System.err.println(excepcion_general.getMessage());
           } finally {
                try {
                    if (conexion != null) {
                          conexion.close();
                     }
                    }
            }catch (SQLException sql_excepcion) {
                     System.err.println(sqq_excepcion.getMessage());
             }
            }
          }
         public static void main(String[] argumentos) {
             CrearTablaAgenga tablas_agenda = new CrearTablaAgenga();
             tablas_agenda.CrearTablas();
          }
    }
```

En realidad puede llegar a asustar tanto código, pero la verdad es que lo que más ocupa es el control de las excepciones de error, como todos sabemos los errores pueden o no controlarse, yo recomiendo controlarlos al máximo posible y creo además que es una buena práctica a la hora de programar. Si nos fijamos la conexión a la base de datos, y la ejecución de la sentencia SQL ocupa sólo cinco lineas (cuatro si quitamos la variable strSQL).
Las instrucciones INSERT, UPDATE y DELETE al ser su codificación igual que la que acabamos de ver, no vamos a remarcar más en ellos, pero lo que si vamos a ver es la clausula SELECT, ya que tiene una menejo distinto de los resultados.
Lo más común es mostrar los resultados en pantalla de una sentencia SELECT, para ello valga el siguiente fragmento de código:

```
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      cn = DiverManager("jdb:odbc:NombreODBC","nombre_usuario","clave");
```

```
StringBuffer strsql = new StringBuffer();

strsql.append(" SELECT ");
strsql.append(" DNI, ");
strsql.append(" NOMBRE, ");
strsql.append(" APELLIDOS, ");
strsql.append(" EDAD ");
strsql.append(" FROM AGENDA ");
strsql.append(" ORDER BY NOMBRE ");

Statement s=cn.createStatement();

ResultSet rs = s.executeQuery(strsql.toString());

out.println("<TABLE>");
out.println("<TR>");
out.println("<TD>DNI</TD>");
out.println("<TD>NOMBRE</TD>");
out.println("<TD>APELLIDOS</TD>");
out.println("<TD>EDAD</TD>");
out.println("</TR>");
while (rs.next()){
   out.println("<TR>");
   out.println("<TD>" + rs.getString("DNI")       + "</TD>");
   out.println("<TD>" + rs.getString("NOMBRE")    + "</TD>");
   out.println("<TD>" + rs.getString("APELLIDOS") + "</TD>");
   out.println("<TD>" + rs.getString("EDAD")      + "</TD>");
   out.println("</TR>");
}
out.println("</TABLE>");

rs.close();
s.close();
cn.close();
```

JavaServer Pages
Parte V Paginación de registros

Indice del tema:

?? ¿ Que es la paginación ?
?? Formas de hacer la paginación con JSP
?? Ejemplo de paginación simple

La paginación con JSP, tiene más dificultad de implementar que por ejemplo en ASP, que te permite hacer posicionamiento en páginas de los resultados de la consulta.
Por normas de usabilidad y estilo, no se debe utilizar una pagina con estructura lineal en internet es decir, no se deben hacer páginas demasiado largas, y obligar al usuario a realizar uns scroll indefinido. Por todos es sabido que una consulta puede tener un número bastante grande de registros en su ejecución, y en primer lugar queda bastante feo mostrar todos los registros en estructura lineal o hacia abajo en la página y segundo lugar puede quedarse el servidor bloqueado al tener que mostrar tal cantidad de datos.
Por este motivo se aconseja hacer paginación de registros, la paginación consiste en mostrar los resultados de una consulta en intervalos definidos, por ejemplo de diez en diez o de quince en quince o de lo que se quiera.

Como deciamos en el apartado anterior, JSP, no te permite situarte en una página en concreto de la consulta que se realiza contra la base de datos, pero lo que si tiene es la posibilidad de situarte en un registro de la consulta. Es decir, puedes tener una consulta con 100 registros por ejemplo y situarte en 27, esto se hace con el método Absolute.
Por ejemplo en el siguiente código nos posicionaremos en el registro 27 y mostraremos en pantalla a partir de ese en concreto.

```
<%@ page language="java"
import="java.sql.*,java.io.*,java.util.*,com.coolservlets.foru
m.database.*" %>
<HTML>
<HEAD>
<TITLE>Paginacion en JSP</TITLE>
</HEAD>
<BODY>

  <%


       // PARA ESTE EJEMPLO YO USO EL POOL DE CONEXIONES,
PUEDES UTILIZAR TU CONEXION COMO QUIERAS
                // POOL DE CONEXIONES
                Connection cn = null;
                cn = DbConnectionManager.getConnection();
       Statement
s=cn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultS
et.CONCUR_READ_ONLY);
                 ResultSet rs = s.executeQuery("select
nombre,apellidos from clientes order by des_nombre");
       rs.absolute(27);
       while (rs.next()) { %>
          <tr>
            <td><%=rs.getString("NOMBRE")%></td>
             <TD><%=rs.getString("APELLIDOS")%> </TD>
          </tr>
     <%}%>
<%// este ejemplo muestra los registros de la consulta a
partir del 27 %>
<%
```

```
   rs.close();
   s.close();
   cn.close();
%>
</BODY>
</HTML>
```

Comentarios al código anterior
El ejemplo anterior es tan básico que casi no hay nada que comentar, tan solo me gustaria que te fijaras en el tipo de Statement que he puesto en el ejemplo.
El statement se crea exactamente igual pero si te fijas en los parametros que se le pasan son `el tipo de Scroll (TYPE_SCROLL_INSENSITIVE), y el modo de abrirlo (CONCUR_READ_ONLY). Estos parametros , que podria explicar que significan pero no lo voy a hacer, permiten entre otras cosas, hacer paginación de registros mediante el posicionamiento en un registros de la salida.`
Y tambien comentar el método absolute del resultset, fijate cual es el parametro que se le pasa, 27, que es justo donde quiero que empiece la página a mostrar datos.

Hasta ahora lo único que hemos hecho es posicionarnos en un registro, pero para hacer la paginación hay que hacer bastantes cosas más, entre ellas

- Controlar el número de registros que tiene la consulta.
- Mostrar de "n" en "n".
- Saber si se pueden mostrar los "n" anteriores y/o los "n" siguientes.

Otra forma de hacer la paginación es mediante el método de toda la vida, yo le llamo así, porque no tiene nada que ver con métodos de java, ninguna clase especial ni con nada que sea nuevo. Es decir, contar los registros anteriores hasta que estemos en el que queremos que sea el inicial y mostra a partir de ese de "n" en "n". Facil no?, pues te puedo asegurar que es muy sencillo y funciona perfectamente. Desde luego, tiene sus ventajar pero tiene un inconveniente bastaten secillo de ver a simple vista, y es que es mas lento, por que?, pues porque tienes que sacar todos los datos y recorrer por ejemplo con un For hasta que compruebes dónde estas, si estás en el adecuado pues mostrar a partir de ahí.
Una pregunta obvia que puedes hacerte es por qué hacer este método de paginación mas sencillo y más lento y no el que utiliza el método Absolute(). La verdad es que esta solución facil viene de una vez que después de hacer todas las páginas JSP, con el método Absolute() y funcionando correctamente en mi oficina, se me ocurrio instalarlos en la máquina del cliente, el resultado fue catastrófico ya que el motor de JDBC que debía interpretar mi código no funcionaba correctamente ya que tenian instalado Oracle en Linux y yo lo tenía en Windows NT, la verdad , es que no funcionó porque el driver JDBC no estaba actualizado y no entendía el método Absolute.
Bueno y después de haberte aburrido con mis experiencias profesionales, pasamos a relizar una paginación con el método absolute que funciona correctamente con Oracle sobre Windows NT. Solo te sugiero que tengas cuidado con Oracle sobre Linux porque el JDBC puede darte problemas.

Es el momento de hacer algo que de verdad vale para algo, aqui te muestro como hacer paginación de registros de una consulta con JSP.

```
<%@ page language="java"
import="java.sql.*,java.io.*,java.util.*,com.coolservlets.foru
m.database.*" %>
<html>
          <head>
          <title>paginacion en JSP</title>
<body bgcolor="#FFFFFF">
<%
          int registros = 10;
```

```
           int RegistroActual = 1;
           String
dbURL="jdbc:oracle:thin:@111.24.80.2:1010:INSTANCIA";
           String usuario = "USUARIO";
           String pwd = "PWD";
           //Class.forName("oracle.jdbc.driver.OracleDriver");
           // Connection cn =
DriverManager.getConnection(dbURL,usuario,pwd);
           // POOL DE CONEXIONES
           Connection cn = null;
           cn = DbConnectionManager.getConnection();
           String LaDireccion =
request.getParameter("Direccion");

           if (LaDireccion==null) LaDireccion="";
           if ( LaDireccion.equals(""))    RegistroActual = 1;
 %>
           <p>
           <%
           // LOS PARAMETROS DE SIG. O ANT. Y EL REGISTRO
ACTUAL
           String empezar = request.getParameter("Actual");
           if (empezar==null) empezar="";
           //VARIABLE PARA EL REGISTRO ACTUAL DONDE SE
ENCUENTRA
           // SI SE LE HA PASADO PARAMETRO DE BUSQUEDA SE LE
PONE AL RegistroActual
           if (!(empezar.equals(""))){
               if (Integer.parseInt(empezar) <= 0)
                   RegistroActual = 1;
               else
                   RegistroActual = Integer.parseInt(empezar);
           }
           // ****************************************
           // SACA EL NUMERO DE REGISTROS DE LA TABLA
           // ****************************************
           int intTotalReg = 0;
           Statement s1 = cn.createStatement();
           ResultSet rsTotalRegistros =
s1.executeQuery("select count(CLIENTES.COD_CLIENTE) AS
TOTAL_REGISTROS FROM CLIENTES " );
           if (rsTotalRegistros != null){
           rsTotalRegistros.next();
           intTotalReg =
rsTotalRegistros.getInt("TOTAL_REGISTROS");
           }
           rsTotalRegistros.close();
           s1.close();

           // FIN DEL NUMERO DE REGISTRO DE LA TABLA
           //************************************


           StringBuffer strsql = new StringBuffer();
           strsql.append(" ");
           strsql.append(" SELECT NOMBRE,APELLIDOS,DIRECCION
FROM CLIENTES ");
           strsql.append(" ORDER BY NOMBRE,APELLIDOS ");
           Statement
s=cn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultS
et.CONCUR_READ_ONLY);
           ResultSet rs = s.executeQuery(strsql.toString());
      if (intTotalReg >0 && (rs.next() || RegistroActual >=1)
```

```
)
           {
           rs.absolute(RegistroActual);

           if (!rs.isFirst()) rs.previous();
           int contador = 1;
           %>
           <table width="508" border="0" cellspacing="1"
cellpadding="3" align="center">
           <tr bgcolor="#0099CC">
           <td><b>listado de Clientes</b>
           </td>
           </tr>
           <tr bgcolor="#54DB00">
           <td>
           <div align="left">Nombre</div>
           </td>
           <td>
           <div align="left">Apellidos</div>
           </td>
           <td>
           <div align="left">Dirección</div>
           </td>
           </tr>
           <%
           int i =1 ;
           do {%>
                   <tr>
                   <td><%=rs.getString("NOMBRE")%></td>
                    <td> <%=rs.getString("APELLIDOS")%></td>
                  <td><%=rs.getString("DIRECCION")%></td>
               </tr>
            <%
             if (rs.isLast()) break;
             i++;
             contadorr++;
           } while (i<=registros && rs.next());
           if (contador == registros)
                RegistroActual =rs.getRow();
           else
                RegistroActual = rs.getRow() + (registros-
contador);

           %>
           </table>
           <table width="100%" align="center" border="0">
           <tr align="center">
           <% if (RegistroActual <= registros) %>
                        <td height="11"> </td>
           <% else {%>
                      <td height="11">
                      <div align="right"><a
href="esta_pagina.jsp?Direccion=ANT&Actual=<%=RegistroActual-
19  %>">
                       &lt;-- Anteriores</a> </div>
                     </td>
           <% } %>
           <td height="11"> </td>
           <% if (RegistroActual >= intTotalReg)%>
                         <td height="11"> </td>
           <% else {
                     if (contador >= registros) {%>
                         <td height="11">
```

```
                              <div align="left"><a
href="esta_pagina.jsp?Direccion=SIG&Actual=<%=RegistroActual+1
%>">
                            Siguientes --&gt;</a></div>
                          </td>
                    <%}
               } %>
               </tr>
               </table>
   <%} else
               out.println("<CENTER>No hay datos en la
consulta</CENTER> ");
               %> </td>
               </tr>
               </table>
               <br>
<%
               rs.close();
               cn.close();
               %>
               </body>
               </html>
```

Autor: ***Miguel Angel García***
E-Mail: webmaster@verextremadura.com

Edición y distribución: **javaHispano**
http://www.javahispano.com