© by Scheme, the API man (*jmrequena@larural.es*)

La API Servlets 2.1

En este capitulo aprenderemos como se realiza la programación del lado del servidor usando la API Servlets 2.1.Como primer punto importante señalaremos que este paquete se trata de una extensión del API Core del JDK y por tanto ha de ser descargado de forma independiente de aquel.La URL para la descarga es http://java.sun.com/products/servlet/index.html, donde podremos encontrar el JSDK 2.1 (Java Servlets Development Kit 2.1).

Debido también a que se trata de una extensión el paquete debe ser indicado en el CLASSPATH de la máquina a ejecutar, concretamente los archivos *server.jar* y *servlet.jar.*

¿Qué es un servlet?

Un servlet de forma intuitiva se puede definir como un programa independiente de plataforma que aporta la misma funcionalidad a la programación en el lado del servidor que tradicionalmente han realizado la interfaz CGI.Con respecto a esta tecnología aporta numerosas ventajas que citaremos a continuación:

- Independencia de la plataforma.(La tan anelada premisa del “write once run everywhere” aun no totalmente conseguida).Esto proporciona un menor esfuerzo de codificación con respecto a soluciones dependientes del servidor web y de la plataforma como ISAPI o NSAPI.

- Ejecución en paralelo de multiples peticiones por una sola instancia del servlet.Tradicionalmente en los programas CGI se ejecuta un proceso distinto para cada petición lo que conlleva una gradual degradación del rendimiento y una necesidad de recursos muy elevada.En un servlet todas las peticiones se atienden en el mismo proceso por distintos hilos y una vez que se ha cargado el servlet este permanece en memoria hasta que se reinicie el servidor o hasta que se le diga lo contrario con lo cual las subsiguientes peticiones son mas rapidas al encontrarse el programa ya cargado en memoria.

- Un servlet puede ejecutarse(incido en esto *puede* no es necesario) en una *sandbox* o recinto de seguridad parecido al modelo que se se sigue con los applets.Debido a esto pueden colocarse servlets en servidores dedicados a hosting sin que la empresa tema por la integridad del servidor y la seguridad de las aplicaciones.

Historicamente el rechazo al uso de los servlets se ha debido a la injustificada leyenda de la falta de velocidad de ejecución del lenguaje Java.Sin embargo si observamos las lenguajes de script que tradicionalmente se han usado para escribir aplicaciones CGI´s (Perl,PHP..) nos encontramos con que tambien son interpretados lo que unido a la necesidad de lanzar un proceso por petición provocan un rendimiento considerablemente menor.Esto no pasa si el lenguaje de implementación del CGI es uno compilado como puede ser C pero la obligatoriedad de cumplir la ecuación 1 peticion = 1 proceso situan el rendimiento en niveles parecidos con un menor consumo de recursos del servlet.

¿Dónde puedo ejecutar Servlets y qué necesito?

En la actualidad la mayoria de servidores web tanto comerciales como de licencia libre tienen la capacidad de ejecutar servlets a traves de plug-ins o modulos.Señalaremos unos cuantos:

- Apache 1.1.3
- Nestcape FastTrack 2.0,Enterprise 2.0,Enterprise 3.0
- Microsoft IIS
- WebLogic Tengah
- Lotus Domino Go Web Server
- IBM Interner Conecction Server
- Java Web Server

Con respecto a este ultimo cabe destacar que ejecuta servlets de forma nativa sin necesidad de modulos adicionales.Señalaremos dos modulos de ejecución de servlets Allaire´s JRun y Yakarta´s Tomcat ambos gratuitos y descargables desde su pagina web si no es para usos comerciales.

Como dato adicional el JSDK 2.1 incluye una herramienta llamada *servletrunner* análoga a *appletviewer* para la ejecución y depuración de servlet con unas capacidades muy limitadas por lo que solo se debe usar para comprobar la exactitud del servlet.

Estructura de un servlet

El API Servlet consiste básicamente en dos paquetes:

- **`javax.servlet`** En este paquete se definen 6 interfaces y 3 clases para la implementación de servlets genericos,sin especificación de protocolo.Hoy en dia no tienen utilidad practica mas que para servir de base en la jerarquia de clases de los servlets.Conforme pase el tiempo se supone que constituiran la base para la implementacion de otros protocolos distintos de http.

- **`javax.servlet.http`** Ofrece la implementación especifica de servlets para el protocolo http.

En estos paquetes se definen todas las clases e interfaces necesarias para la escritura de applets.De hecho cuando se usen los servlets(y hoy en dia no hay otra utilidad) para gestionat conexiones http usaremos las clases del paquete **`javax.servlet.http`**.

El ciclo de ejecución de un servlet es análogo al de un applet con ligeras diferencias.Inicialmente el servlet debe extender a la clase **`HttpServlet`**:

```
import javax.servlet;
import javax.servlet.*;
import javax.servlet.http.*;

public class MiServlet extends HttpServlet{
....
}
```

Para dotar de funcionalidad a un servlet se han de redefinir una seria de metodos que guardan una analogía con los metodos de funcionamiento de un applet(**init(),start(),stop(),destroy()**).

- **public void init(ServletConfig config)**

Cada vez que se inicia el servlet el servidor web llama a este metodo pasando un parámetro de la clase **ServletConfig** que guarda información de la configuración del servlet y del contexto del servidor web en el que se ejecuta. A traves de **ServletConfig** se accede a los parámetros de inicialización del servlet que se establecieron al configurar el servlet y a traves de la interfaz **ServletContext** (obtenido a partir del metodo **getServletContext()** de **ServletConfig**)se accede a la información del servidor web.

A modo de ejemplo desarrollaremos un ejemplo simple de un servlet que escribe información en un fichero de registro(el formato,ubicación y nombre de este es dependiente del servidor web):

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
public class MiServlet extends HttpServlet{
  public void init(ServletConfig config){
    config.getServletContext().log("Iniciado MiServlet a las" +new Date());
  }
}
```

En este metodo se han de realizar todas las operaciones unicas en el ciclo de vida del servlet tal como conexion a BD de forma persistente y otras tareas de inicialización.Dado que el servlet se carga en memoria al iniciar el sevidor web o al recibir la primera petición(dependiendo de la configuración) el metodo init() es llamado *solo* una vez,no cada vez que se realice una petición.

public void destroy()

Este metodo es analogo al metodo **init()** solo que sera llamado por el servidor web cuando el servlet esta a punto de ser descargado de memoria(repito e insisto: no cuando termina una petición).En este metodo se han de realizar las tareas necesarias para conseguir una finalizacion apropiada como cerrar archivos y flujos de entrada de salida externos a la petición,cerrar conexiones persistentes a bases de datos...etc etc..

Un punto importante es que se puede llamar a este método cuando todavía esta ejecutándose alguna petición por lo que podría producirse un fallo del sistema y una inconsistencia de datos tanto en archivos como en BD. Por eso debe retrasarse la desaparición del servlet hasta que todas las peticiones hayan sido concluidas(con respecto a la peticiones que lleguen en ese intervalo el servidor web sencillamente las desestimará).

public void service(HttpServletRequest request,

HttpServletResponse response) throws ServletException, IOException.

En este metodo se encuentra la mayor parte de la funcionalidad del servlet.Cada vez que se realice una petición se llamará a este metodo pasándole dos parámetros que nos permite obtener información de la petición y un flujo de salida para escribir la respuesta.Análogamente tenemos otra serie de metodos que realizan la implementación

de respuesta a metodos de comunicación del protocolo http 1.1 como son GET y POST.Estos son respectivamente:

```
public void doGet(HttpServletRequest request,HttpServletResponse response)

public void doPost(HttpServletRequest request,HttpServletResponse response)
```

Los dos parámetros que recibe **service()** son esenciales para el funcionamiento del servlet por lo que pasaremos a verlos con mas profundidad:

Los dos parámetros que recibe **service()** son **HttpservletRequest** y **HttpServletResponse**

HttpServletRequest

Esta interfaz derivada de **ServletRequest** proporciona los metodos para recuperar la informacion de la petición del usuario asi como del propio usuario.Señalaremos los mas importantes:

public abstract String getRemoteHost()
Devuelve el nombre del ordenador que realizó la petición

public abstract String getParameter(String parameter)
Devuelve el valor del parámetro *parameter* o null si dicho parámetro no existe.

public abstract String[] getParameterValues(String parameter)
Devuelve un array con los valores del parametro especificado por parameter o null si dicho parametro no existe.

public abstract Enumeration getParameterNames()
Devuelve una Enumeration de los nombre de los parametros empleados en la petición.

HttpServletResponse

Se trata de un interfaz derivada de ServletResponse que proporciona los metodos para realizar la respuesta al cliente que originó la petición.Señalaremos los mas importantes:

public abstract PrintWriter getWriter()
Permite obtener un objeto **PrintWriter** para escribir la respuesta.

public abstract setContentType(String)
Permite establecer el tipo MIME de la respuesta

public abstract setContentLenght(int)
Permite especificar la longitud de la respuesta.Si no se indica el propio motor de servlets calculará la longitud de la respuesta.A efectos practicos el autor de este tutorial no encuentra otra utilidad a este metodo mas que permitir al navegador mostrar una barra de progreso que muestre el estado de descarga de la solicitud.

Ejemplo de servlet

A continuación realizaremos un sencillo de ejemplo de un servlet que recibirá como parámetro un nombre y saludará al cliente que realizo la petición. Para ello construiremos una pagina web con un formulario que nos servirá para enviar la petición al servlet.

```
<html>
  <head>
```

```
    <title>Ejemplo de servlet</title>
  </head>


<!-- Se da por supuesto
que el servlet se esta ejecutando en la maquina local,en el puerto 8080 que
es
el que por defecto usan los servlets y el servlet se encuentra situado en
el
directorio /servlet a partir de la raiz del servidor.Encaso contrario
habría
que cambiar estos parámetros del servlet-->


  <body>

    <h1>Introduzca su nombre y pulse el boton de enviar</h1>

    <FORM ACTION="http://localhost:8080/servlet/HolaServlet" METHOD="post">
      Nombre:<INPUT TYPE="text"  NAME="nombre" size="30">
      <INPUT TYPE="submit" NAME="enviar" VALUE="Enviar">
    </form>
  </body>
</html>
```

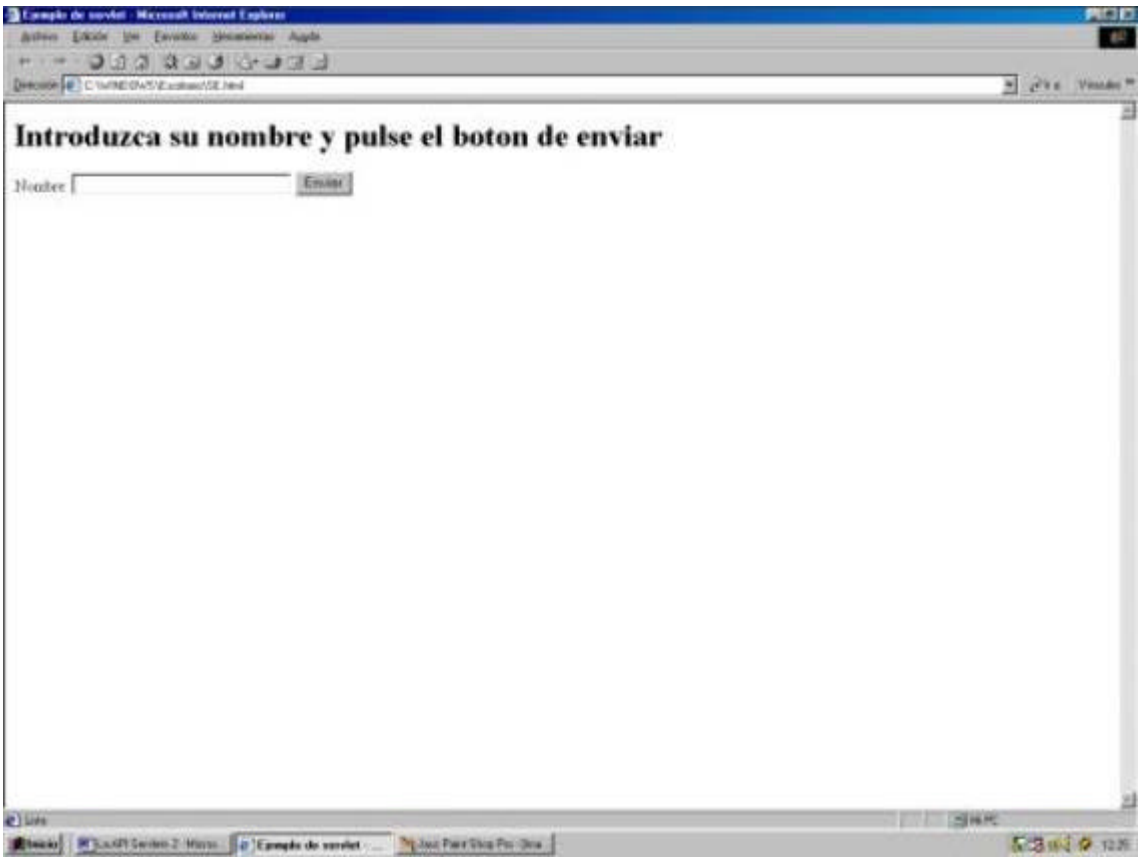

Figura 1.1 Aspecto de la pagina de petición.

A continuación se muestra el codigo del servlet profusamente comentado.Este codigo fuente se compilaría y se situaría en el directorio configurado en el servido web para la ejecución de servlets(en nuestro caso sera /servlet):

```
import java.io.*;
import javax.servlets.*;
import javax.servlets.http.*;
```

```
public class HolaServlet extends HttpServlet {

  /*
  *  Redefinición del metodo init y configuración de los parámetros de
inicio
  */
  public void init(ServletConfig config) throws ServletException {

    /*
     * Llamada la metodo init() de la superclase.Esto es imprescindible
     * para la correcta inicialización del servlet y debe realizarse
     * antes que cualquier otra acción
    */

    super.init(config);
    System.out.println("HolaServlet arrancado a las " + new Date());
  }

    /*
     * Redefinición del metodo destroy sin tareas a realizar en este caso
     */
    public void destroy() {
      System.out.println("HolaServlet detenido a las " + new Date());
    }

    /*
     *  En este caso se ha optado por redefinir el metodo doPost(),
pudiendose
     *  igualmente haberse optado por redefinir service().Lo que seria
incorrecto
      * es redefinir doGet() ya que la peticion se realizará por el método
post
     */
    public void doPost(HttpServletRequest request,
              HttpServletResponse response) throws ServletException {

      /*
       * Se obtiene el valor del parametro enviado
       */
      String name = request.getParameter("nombre");
      /*
       * Se establece el contenido MIME de la respuesta
       */
      response.setContentType("text/html");

      /*
       * Se obtiene un flujo de salida para la respuesta
       */
      PrintWriter out;

      try {
        out = response.getWriter();
      }catch (IOException e){
        System.out.println("Error en el canal de salida: "+e.toString());
      }

      /*
       *Se escribe la respuesta en HTML estandar
       */
      out.println("<html>");
      out.println("<head>");
      out.println("<title> Respuesta de HolaServlet</title>");
      out.println("<head>");
```

```
        out.println("<body>");
        out.println("<h1>¡Funcionó!:El servlet ha generado la pagina</h1>");
        out.println("<br>");
        out.println("<font color="red">");
        out.println("<h2>Hola " + name + "</h2>");
        out.println("</font>");
        out.println("</body>");
        out.println("</html>");

        /*
         * Se fuerza del volcado del buffer de la salida y se cierra el canal
         */
        out.flush();
        out.close();
    } //fin doPost()
}//fin clase
```

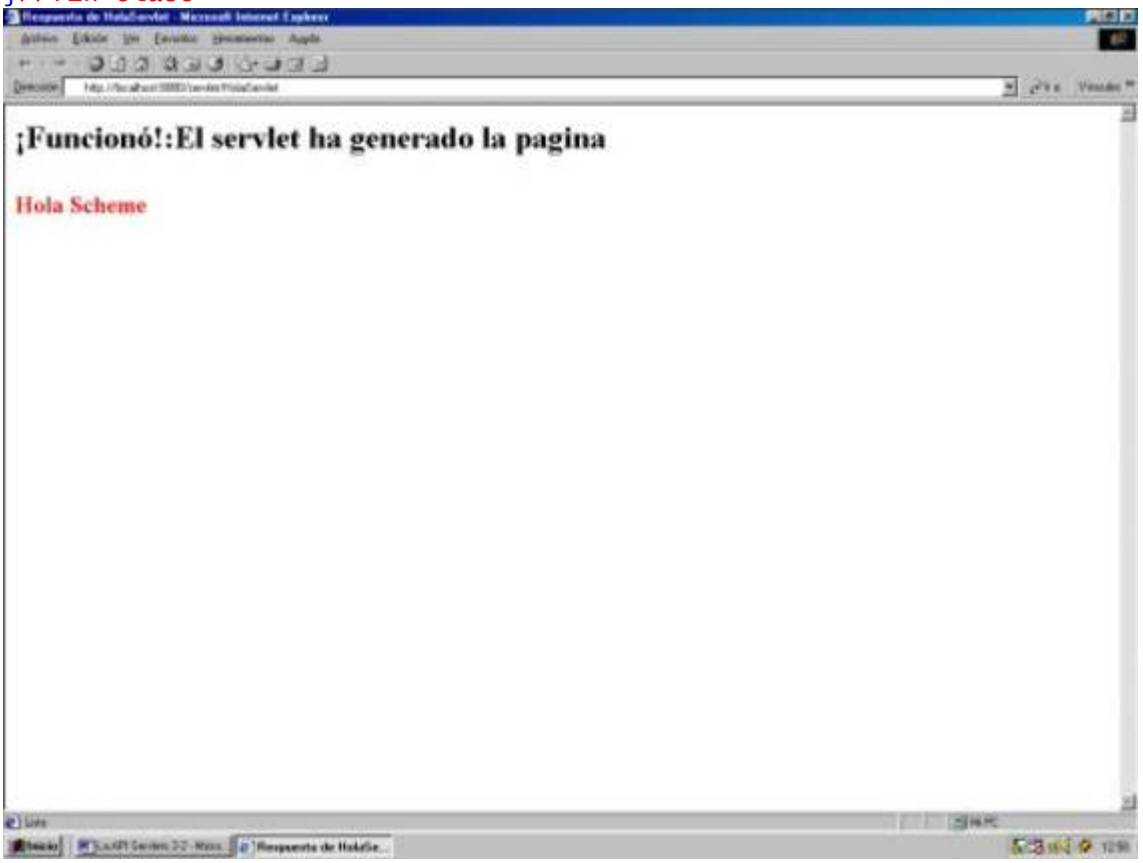

Figura 1.2.Respuesta del servlet.

La API JDBC 1.2

La API JBDC es una interfaz de acceso a RDBMS(Relational Database Management System) independiente de la plataforma y del gestor de bases de datos utilizado.Se relaciona muy a menudo con el acronimo ODBC por lo que se suele expresar como *Java Database Connectivity* pero oficialmente,según Javasoft,JDBC no significa nada ni es acronimo de nada.

El API consiste en una serie de interfaces Java implementadas por un controlador.Este programa de gestión se encarga de la traducción a las llamadas estandar que requiere la base de datos compatible con el.De esta manera el programador puede abstraerse de la programación especifica de la base de datos creando codigo que funcionará para todas los RDBMS que cuenten con un driver JDBC con solo cambiar tal driver.

En la actualidad se encuentran drivers JDBC para todos los sistemas de gestión de bases de datos mas populares(e incluso podriamos decir existentes) como Informix,Oracle,SQL-Server,DB2,InterBase, SyBase... y otros productos de indole no comercial como mSql,mySql y PostGress,etc,...

Aun asi existe un tipo especial de drivers denominados puentes JDBC-ODBC que traducen las llamadas en JDBC a llamadas en el estandar de comunicación con bases de datos desarrollado por Microsoft ODBC por lo que en ultimo termino siempre se podrá utilizar uno de estos drivers ya que la totalidad de los sistemas de gestión de bases de datos cuentan con un driver de este ultimo tipo.Esto nos lleva a la clasificación de los distintos drivers que cumplen la especificación JDBC 1.2.

Tipos de drivers JDBC

A continuación se señalan y describen los cuatro tipos distintos de drivers JDBC existentes:

- **Nivel 1: Puente JDBC:ODBC.**
  Esta categoría de controladores se remite al controlador de puente JDBC-ODBC original.Este ultimo utiliza el controlador ODBC de Microsoft para comunicarse con los servidores de base de datos.Se implementa tanto en codigo binario como en Java y debe ser instalado previamente en la computadora cliente antes de poder usarlo.

- **Nivel 2: Controlador JDBC de protocolo nativo en Java y binario.**
  Esta categoría esta compuesta por controladores que hablan con los servidores de bases de datos en el protocolo nativo de la base de datos.Se implementan como una combinación de codigo binario y Java y deben ser instalados en el cliente antes de poder ser usados.

- **Nivel 3:Controlador JDBC de protocolo no nativo en Java puro.**
  Esta categoría esta formada por controladores de Java puro(no hay codigo binario) que hablan un protocolo de red estandar(como http) con un servidor de acceso a bases de datos.Este servidor traduce el protocolo de red a uno de base de datos específicos de la marca.No necesita instalación en cliente.

- **Nivel 4:Controlador JDBC de protocolo nativo en Java puro**
  Está formada por controladores de Java puro que hablan el protocolo de la base de datos especifico de la marca edl servidor de bases de datos que se haya designado para servir de interfaz.No necesita instalción en cliente.

Como se observa la clasificación de los niveles de controladores JDBC se hace en funcion de que hablen el protocolo nativo de la base de datos y de que se encuentren implementados en Java puro o parcialmente esten escritos en codigo binario dependiente de la plataforma.Con respecto a lo primero la eliminación de las capas de traducción en el caso de que la llamada del driver no sea nativa pueden degradar el rendimiento pero hay ocasiones en que no hay otra opcion.Por otra parte el hecho de que el driver esté escrito en Java puro consigue,aparte de independencia de la plataforma de ejecución,que no sea necesaria instalación ni configuración(como el caso del DSN del ODBC) en el cliente lo que facilita la gestión y siendo la unica solución en casos en que el cliente sea un applet.Si encontramos los 4 niveles de drivers el mas adecuado en el 95% de las ocasiones será el de nivel 4.

El paquete java.sql Consta de una seria de clases e interfaces de las cuales pasaremos a discutir las mas importantes:

- **Driver**

Se trata de una clase que implementa el controlador JDBC especifico de la base de datos y es suministrado por el proveedor de bases de datos.Junto a la clase *DriverManager* permite cargar y descargar los controladores de forma dinamica.El controlador de sirve de una cadena para localizar y acceder a recursos dentro la base de datos con una sintaxis muy parecida a una URL.En todo caso esta cadena será de la forma:

```
Jdbc:<controlador>://<servidor>:<puerto>/<base de datos>
```

Antes de realizar la conexión con la base de datos se debe haber cargado en memoria el controlador para lo que se usa el metodo estatico de la clase Class *forName(String).*

- *Connection*

  Esta interfaz representa una sesión persistente con la base de datos que es devuelta por el Driver. Nos permite utilizar transacciones(si el DBMS lo admite) asi como obtener una interfaz para la ejecución de instrucciones SQL.

- *Statement*

  Esta interfaz se trata de un vehículo para la ejecución de sentencias SQL a la base de datos y la extracción de resultados.A este respecto hay que señalar que JDBC acepta el estandar SQL-92 como minimo exigible por lo que implementaciones nuevas y/o dependientes del DBMS pueden no estar admitidas.

- *ResultSet*

  Representa un conjunto de resultados de forma abstracta(esto es una "tabla").Dependiendo de su creación permite acceso secuencial o aleatorio y presenta una serie de metodos para obtener información de los resultados y para movernos por el conjunto.

Una vez vistas las clases e interfaces para la gestion de consultas JDBC veremos los pasos a seguir para realizar una consulta a la base de datos.

Inicialmente se debe cargar en memoria el controlador JDBC que vayamos a usar:

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

Esta sentencia hace que la JVM busque en todas las rutas especificadas por el CLASPATH la clase correspondiente al driver y la cargue en memoria de tal manera que este lista para posteriores usos. Seguidamente se debe realizar la conexión con la base de datos:

```
/*
 * Se usa ahora un driver Oracle para acceder a la maquina local y a la
tabla Ejemplo:
 */
String Url = "jdbc:oracle://localhost:8080/Ejemplo";
Connection conn = DriverManager.getConnection(url);
```

Tambien existen versiones de este ultimo metodo que permiten realizar la conexion con la BD especificando un nombre de usuario y una contraseña.

Creamos una sentencia para poder interactuar con la BD mediante el uso de SQL:

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

Ahora se deberian usar algunos metodos de la interfaz Statement dependientes del tipo de sentencia SQL que queramos realizar,si devuelve datos o no,o si no sabemos si devuelve datos.

```
/*
 * La ejecución de la instrucción SQL devuelve resultados
 */
ResultSet rs = stm.executeQuery("SELECT * FROM Ejemplo");
int numRowsUpdated = stm.executeUpdate("INSERT INTO Ejemplo VALUES
(`Pepe`,`Sanchez`,`45598652`)");

/*
 * No se sabe si devuelve resultados.Util cuando no se sabe que tipo
 * de instruccion SQL se esta ejecutando como cuando el usuario introduce
directamente
 * el SQL. Posteriormente si devuelve resultados se recupera con el metodo
getResultSet()
 */

Boolean hasResults = stm.execute("SELECT * FROM Ejemplo");
if(hasResults) {
  ResultSet rs = stm.getResultSet();
}
```

La interfaz ResultSet presenta metodos para obtener un tipo SQL convertido a un tipo Java a partir del nombre de la columna de la forma ***getXXX(String nombreColumna)*** *y se desplaza a traves de las filas usando el metodo bolean next() que desplaza el indicador de posición del ResultSet a la siguiente columna y devuelve un booleano indicando si hay mas filas(inicialmente se encuentra en la primer fila).Las XXX representan algun tipo Java como int,String,flota,double...obteniéndose metodos como getInt(string),getString(String),...*

Ahora si disponemos de un objeto Resulset podemos usar sus metodos para desplazarnos por el de la siguiente manera:

```
while(rs.next()) {
  System.out.print(rs.getString("Nombre")+ "-");
  System.out.print(rs.getString("Apellidos")+ "-");
  System.out.println(rs.getInt("DNI"));
}
```

Un ejemplo para ponerlo todo junto

A continuación desarrollaremos un programa que mediante la utilización de Servlets permita actualizar y consultar los registros de una base de datos de ejemplo.El ejemplo constará de dos paginas,una que dará acceso a un servlet que permitirá introducir datos en una tabla y otra que permitirá consultar todos los registros introducidos hasta la fecha.La estructura de la tabla será la siguiente:

Nombre	Apellidos	**Dni**

El driver utilizado será el puente JDBC:ODBC dado su amplia difusión y el carácter introductorio de este tutorial y se supondrá que la maquina que ejecuta las peticiones es la misma que corre el servidor web y el servidor de bases de datos.Dada la utilización del ODBC debe configurarse el DSN(de usuario o de sistema indistintamente) previamente a la utilización de servlets.A continuación se presentan las paginas HTML usadas y un captura de su apariencia:

Pagina de introducción de datos:

```
<HTML>
  <HEAD>
     <TITLE>Pagina de introducción de datos</TITLE>
```

```
  </HEAD>

  <BODY>
    <H1>INTRODUZCA SUS DATOS</H1>

     <FORM ACTION="localhost:80/servlets/InServlet METHOD="post">
       NOMBRE:<INPUT TYPE="text" NAME="Nombre" SIZE="15" >
       <BR>
       APELLIDOS:<INPUT TYPE="text" NAME="Apellidos" SIZE="30">
       <BR>
       DNI: <INPUT TYPE="text" NAME="Dni" SIZE="15">
       <BR>
       <INPUT TYPE="submit" NAME="Enviar" VALUE="Enviar">
       <INPUT TYPE="reset" NAME="Limpiar" VALUE="Limpiar">
       <BR>
     </FORM>

  </BODY>
</HTML>
```

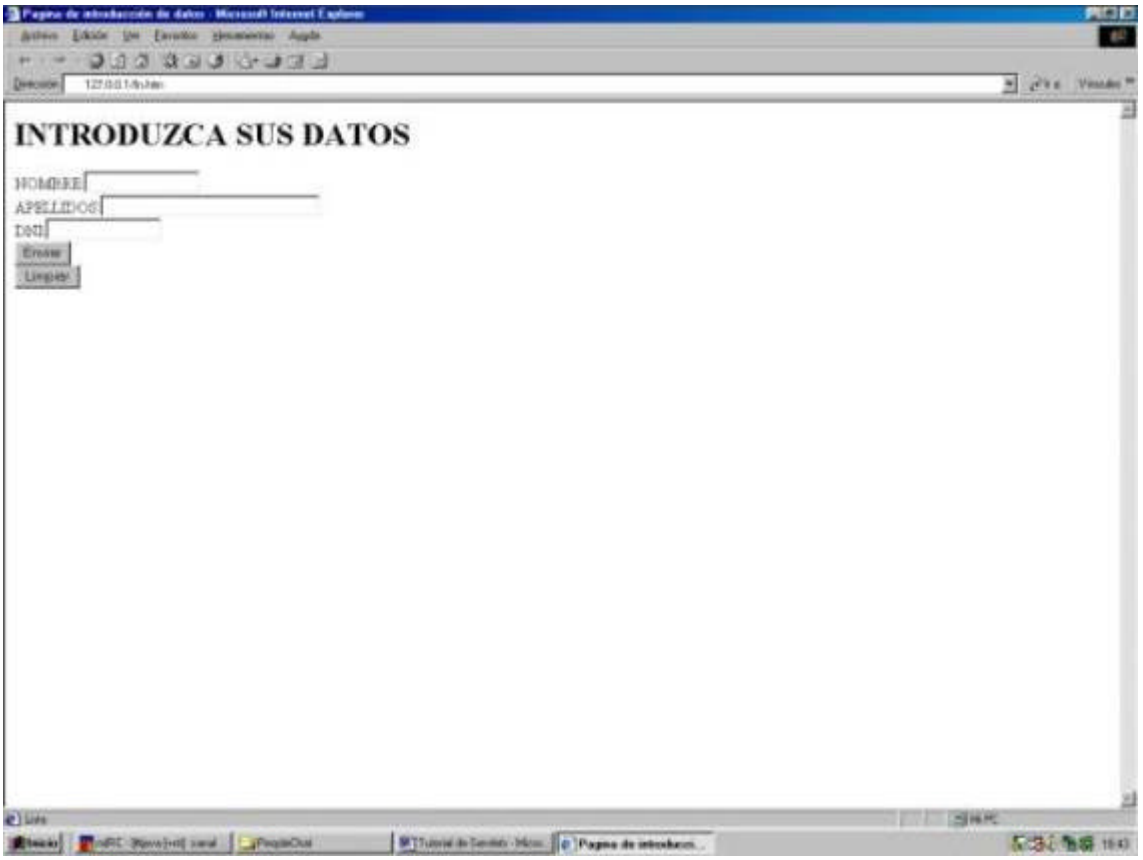

Figura 2.1 *Página de introducción de un registro en la base de datos*

Página de consulta de resultados

```
<-- Las consultas se realizan por DNI-->
<HTML>
  <HEAD>
    <TITLE>Pagina de recuperación de datos</TITLE>
  </HEAD>


<BODY>
  <H1>INTRODUZCA EL DNI DE LA PERSONA A BUSCAR</H1>
```

```
  <FORM ACTION="localhost:80/servlets/OutServlet" METHOD="post">
    DNI:<INPUT TYPE="text" NAME="Dni" SIZE="15">
    <BR>
    <INPUT TYPE="submit" NAME="Enviar" VALUE="Enviar">
    <BR>
    <INPUT TYPE="reset" NAME="Limpiar" VALUE="Limpiar">
    <BR>
  </FORM>
</BODY>
</HTML>
```

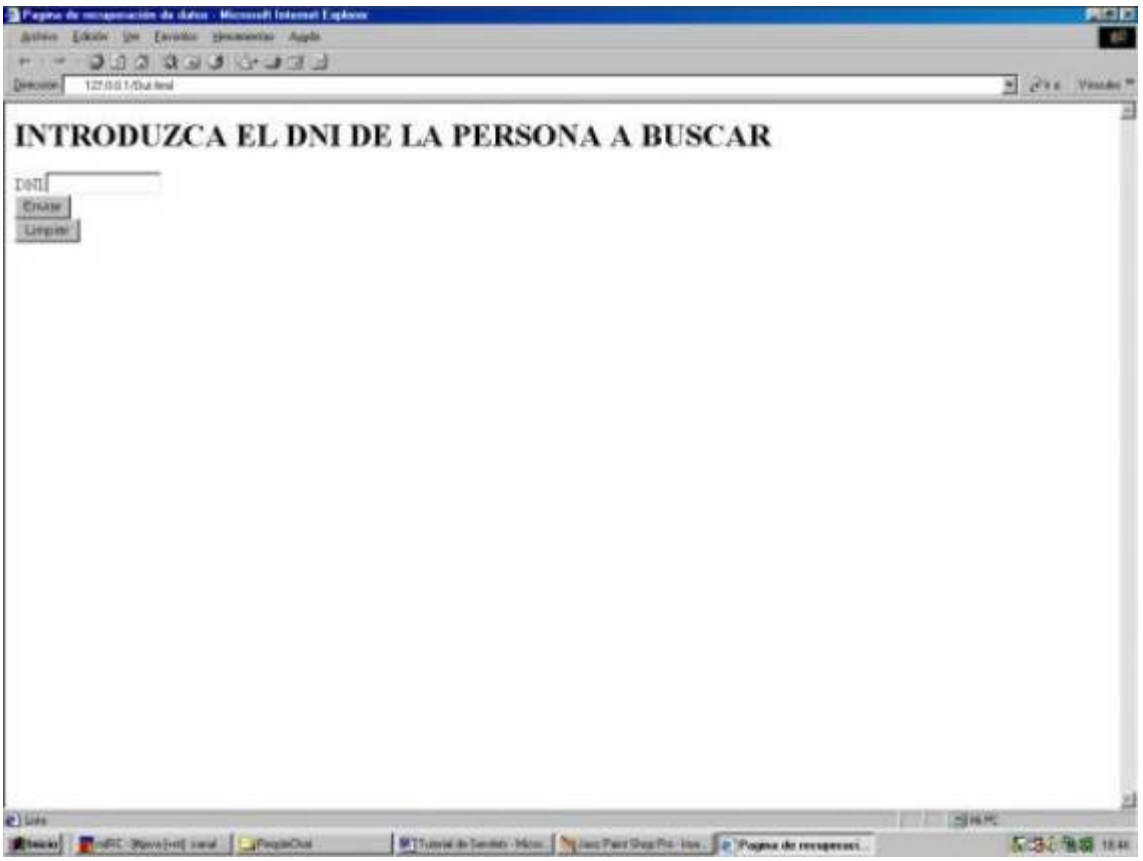

Figura 2.2 Pagina de introducción del DNI para consulta de ese registro

Como se puede observar la aplicación consta de dos servlets que operarán sobre la misma base de datos Ejemplo.El primero InServlet permitirá introducir datos en la BD mientras que el segundo OutServlet permitirá realizar consultas basándose en el DNI de la persona.A continuación se presenta el codigo del primer servlet:

```
/*
 *InServlet.java
 */
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;

public class InServlet extends HttpServlet {
  Connection conn;

  /*
   * En el metodo Init se realizará la conexión a la BD
   * por lo que las peticiones no producirán ningun retardo de conexión
```

```
   */
  public void init(ServletConfig cf) throws ServletException {
    super.init(cf);

    // Se carga el driver a continuación
    try {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    }catch (ClassNotFoundException e) {
      System.out.println("Imposible     encontrar     el     driver:"     +
e.getMessage());
    }

    //Se intenta realizar la conexión a la base de datos ejemplo
    try {
      conn = DriverManager.getConnection("jdbc:odbc://localhost/Ejemplo");
    }catch (SQLException e) {
      Systen.out.println("Imposible crear conexion:" + e.getMessage());
    }
  }// fin init

  /*
   * En el metodo destroy() una vez que el servlet se esta apagando,
desconectaremos
   * de la base de datos
   */
  public void destroy() {
    super.destroy();
    //Llamada al destroy de la superclase
    try {
      conn.close();
    }catch (SQLException e){
      Systen.out.println("Imposible cerrar conexion:" + e.getMessage());
    }
  } //fin destroy()

  /*
    *Redefinimos el metodo doPost ya que las peticiones se realizaran por
ese metodo
   */
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response) throws
                                          ServletException {
    String nombre;
    String apellidos;
    int dni;

    //Obtenemos los valores de los parámetros
    nombre = request.getParameter("Nombre");
    apellidos = request.getParameter("Apellidos");
    dni = request.getParameter("Dni");

    //Comprobamos que no hay ninguno vacio
    //Si es asi enviamos una pagina informando del error y si no
    //actualizamos la BD e informamos del éxito de la operación
    if (nombre==null || apellidos == null || dni == null ) {
      PrintWriter out = response.getWriter();
      //Devolvemos una pagina de error
      out.println("<HTML>");
      out.println("<HEAD>");
      out.println("<TITLE>Error en la actualización de datos</TITLE>");
      out.println("</HEAD>");
      out.println("<BODY>");
```

```
      out.println("<H1>¡¡Esto no rula si no pones todos los parametros
chaval!!</H1>");
      out.println("</BODY>");
      out.println("</HTML>");
      out.flush();
      out.close();
    }else {// no hay errores
      try {
        //Creamos una sentencia para la introducción de valores
        Statement stm = conn.createStatement();
        int nunRowsUpdated = stm.executeUpdate("INSERT INTO Ejemplo
VALUES"+"(`"+nombre+"`,`"+apellidos+"`,`"+dni+"`)");
        PrintWriter out = response.getWriter();

       //Devolvemos una pagina de exito de operación

       out.println("<HEAD>");
       out.println("<TITLE>Registro actualizado</TITLE>");
       out.println("</HEAD>");
       out.println("<BODY>");
       out.println("<H1>Operación realizada con exito</H1>");
       out.println("<H3>Actualizadas”+numRowsUpdated + “filas</H3>");
       out.println("</BODY>");
       out.println("</HTML>");
       out.flush();
       out.close();
     }catch(Exception e){
       System.out.println("Error en la actualización" + e.getMessage());
     }finally{
       try{
         stm.close();
       }catch(SQLException e){}
     }//fin finally
   }//fin else
  }//fin doPOst()
}//fin servlet
```

A continuación OutServlet.java

```
/*
 *OutServlet.java
 */
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;

public class OutServlet extends HttpServlet {
  Connection conn; //Objeto de la conexion

  //En el metodo Init se realizará la conexión a la BD
  //por lo que las peticiones no producirán ningun retardo de conexión
  public void init(ServletConfig cf) throws ServletException {
    super.init(cf);
    //Se intenta cargar el driver de puente JDBC-ODBC
    try {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    } catch (ClassNotFoundException e) {
      System.out.println("Imposible encontrar el driver:" +
e.getMessage());
    }

    //Se intenta realizar la conexión a la base de datos ejemplo
```

```
    try {
      conn = DriverManager.getConnection("jdbc:odbc://localhost/Ejemplo");
    }catch (SQLException e) {
      Systen.out.println("Imposible crear conexion:" + e.getMessage());
    }

    //En el metodo destroy() una vez que el servlet se esta apagando,
    //desconectaremos de la base de datos
  }// fin init()

  public void destroy() {
    super.destroy();
    try {
      conn.close();
    } catch (SQLException e) {
      Systen.out.println("Imposible cerrar conexion:" +e.getMessage());
    }
  }//fin destroy()

  //Redefinimos el metodo doPost ya que las peticiones se realizaran por
ese metodo
  public void doPost(HttpServletRequest request,
             HttpServletResponse response) throws ServletException {
    int dni;
    //Obtenemos los valores de los parámetros
    dni = request.getParameter("Dni");
    //Comprobamos si esta vacio
    //Si es asi enviamos una pagina informando del error y si no
    //consultamos la BD e informamos de los resultados

    if (dni == null ) {
      PrintWriter out = response.getWriter();
      //Devolvemos una pagina de error
      out.println("<HTML>");
      out.println("<HEAD>");
      out.println("<TITLE>Error en la actualización de datos</TITLE>");
      out.println("</HEAD>");
      out.println("<BODY>");
      out.println("<H1>¡¡Va   a   ser   difícil   si   no   pones   un   DNI
chaval!!</H1>");
      out.println("</BODY>");
      out.println("</HTML>");
      out.flush();
      out.close();
    } else { // no hay errores
      try {
        //Creamos una sentencia para la recuperación de valores
        Statement stm = conn.createStatement();
         ResultSet rs = stm.executeQuery("SELECT * FROM Ejemplos WHERE Dni
="+ dni);
        PrintWriter out = response.getWriter();
        //Devolvemos una pagina de exito de operación
        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>Datos de la persona solicitada</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");
        out.println("<TABLE BORDER=3 WIDTH=75%>");
        out.println("<TR>");
        out.println("<td   width=33%><font   color=\"#FF0000\">Nombre</font>
</td>");
         out.println("<td width=33%><font color=\"#FF0000\">Apellidos</font>
</td>");
```

```
        out.println("<td  width=\"33%\"><font  color=\"#FF0000\">Dni</font>
</td>");
        out.println("</TR>");
        while(rs.next()) {
          out.println("<tr>");
          out.println("<td    width=\"33%\">"+    rs.getString(\"Nombre\")+
"</td>");
          out.println("<td  width=\"33%\">"+  rs.getString(\"Apellidos\")+"
</td>");
          out.println("<td width=\"33%\">"+ rs.getInt(\"Dni\")+"</td>");
          out.println("</tr>");
        }
        out.println("</TABLE>");
        out.println("</BODY>");
        out.println("</HTML>");
        out.flush();
        out.close();
      }catch (Exception e) {
        System.out.println("Error en la actualización" + e.getMessage());
      }finally {
        try {
          stm.close();
        }catch(SQLException e){ }
      }//fin finally
    }//fin else
  }//fin doPost()
}//fin servlet
```

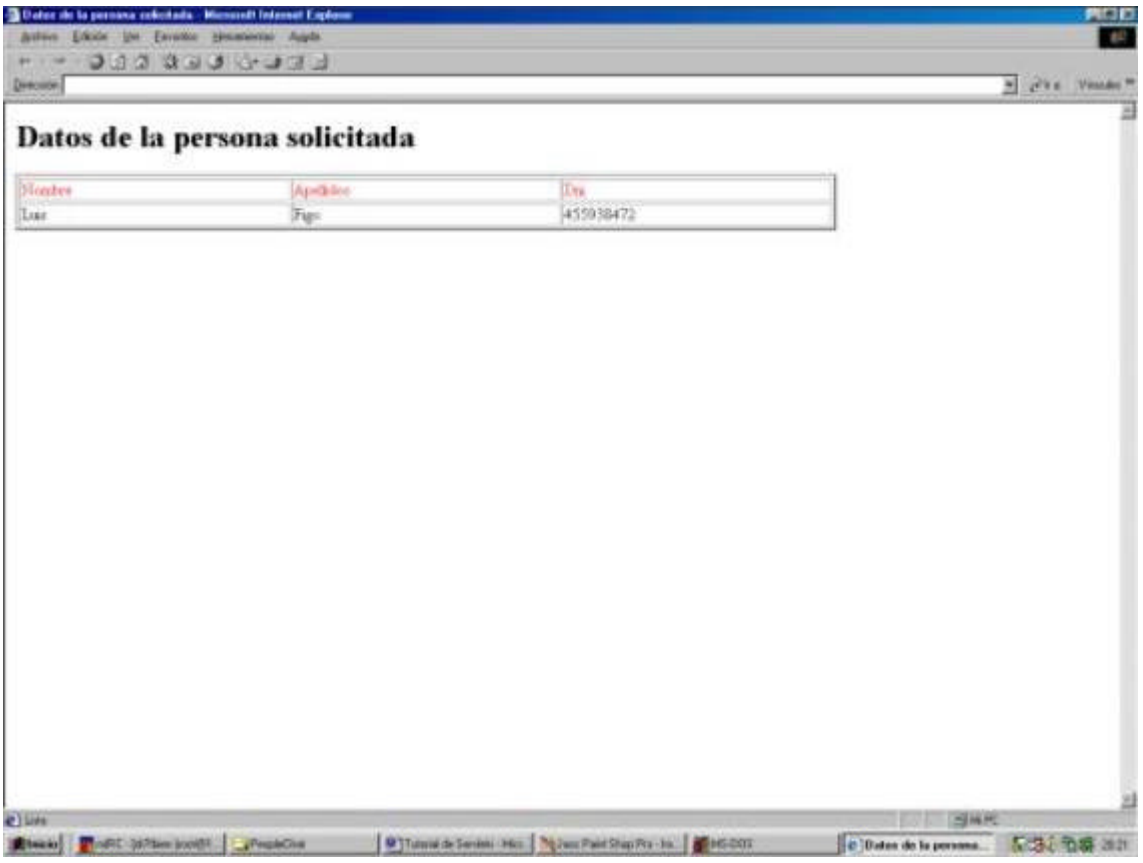

***Figura 2.3**.- Resultado de una consulta por DNI.*

Apéndice

BIBLIOGRAFÍA

Titulo: *Java Unleashed 1.2*
Autor: *Jaime Jaworski*
Editorial: *Prentice Hall*
ISBN: *84-8322-061-X*

Titulo: *Mastering Java 2*
Autor: *John Zukowski*
Editorial: *Sybex*
ISBN: *84-415-0948-4*

Titulo: *Using Java 1.2*
Autor: *Mike Morgan*
Editorial: *Prentice Hall*
ISBN: *0-7897-1627-5*

Titulo: *Thinking In Java*
Autor: *Bruce Eckel*
Editorial: *descargable desde www.bruceEckel.com*
ISBN: *-----*

También se puede consultar mas información desde www.servlets.com y las páginas de java.sun.com y www.javasoft.com.

Para cualquier duda o matización ponerse en contacto con el autor en jmrequena@larural.es.

Editado y distribuido por www.javahispano.com