

## Aplicaciones web con Tomcat y MySQL en Linux

En este artículo se muestra como usar las tecnologías JDBC y Java Servlet para la creación de aplicaciones web que interactúen con una base de datos MySQL cuando se trabaja principalmente sobre una plataforma Linux.

Por MySQL hispano

## Introducción

Una aplicación web es cualquier aplicación que utiliza el Protocolo de Transferencia de Hipertexto o HTTP como principal protocolo de comunicación y de intercambio de información entre un cliente y un servidor.

El web se diseñó originalmente como un medio para suministrar contenidos por medio de páginas HTML estáticas. Cuando un navegador web envía una petición a un servidor web, éste último se encarga de buscar la página solicitada en su sistema de archivos y devolverla al navegador. Sin embargo, lo que devuelve el servidor web no tiene por qué ser siempre una página HTML estática almacenada en el servidor, ya que puede tratarse de la salida de un programa que se ejecuta en el entorno del servidor web. Esto es lo que se ha hecho por mucho tiempo con el uso de la interfaz CGI (Common Gateway Interface).

CGI es una opción válida ya que permite generar contenido dinámicos, sin embargo presenta una serie de limitaciones, en especial lo relacionado con la eficiencia. Cada vez que se hace una petición a un programa CGI, se arranca un nuevo proceso. Si el programa CGI es relativamente pequeño, el costo de arrancar el proceso puede superar el tiempo de ejecución del mismo. Con los servlets, la Máquina Virtual de Java permanece en ejecución y administra cada petición mediante un ligero subproceso de Java, no con un pesado proceso del sistema operativo. En resumen, los servlets son la respuesta de la tecnología Java a la programación CGI tradicional, y nos proporciona un ambiente de desarrollo más eficiente, seguro, portable y robusto del lado del servidor, a través del API Java Servlet.

En este artículo se muestra como combinar las tecnologías JDBC y Java Servlet para la creación de aplicaciones web que interactúen con una base de datos MySQL. Se recomienda leer un artículo publicado con anterioridad titulado [MySQL con Java en Linux](#) en el que se explica el uso del driver JDBC para MySQL, ya que en este artículo no se darán demasiados detalles acerca del API JDBC, y más bien nos centraremos en el uso del API Java Servlet.

Dado que los Java Servlets forman la base de otra tecnología para el desarrollo de aplicaciones web conocida como JSP (JavaServer Pages), es importante conocer los principios básicos de éstos para realizar un desarrollo efectivo con esta otra tecnología de Java. Cabe mencionar que muchas de las aplicaciones en sitios reales combinan ambas tecnologías, en lugar de usar sólo una de ellas.

Aunque se da por hecho que no se tienen los conocimientos necesarios del desarrollo de aplicaciones sobre un servidor web, sí se espera que se tengan los conocimientos básicos de la programación con Java.

## Herramientas necesarias

- » Un ambiente de desarrollo para Java, tal como el Java 2 SDK. La versión estándar del SDK 1.4 ya incluye el API JDBC.
- » Un servidor de bases de datos MySQL al que se tenga acceso con un nombre de usuario y una contraseña.
- » El driver JDBC para MySQL, [Connector/J](#).
- » El servidor de aplicaciones [Tomcat](#) que nos permitirá trabajar con los programas en Java. Con éste se incluye el API Java Servlet.

### Creación de la base de datos

Para nuestro ejemplo necesitamos crear una base de datos que nombraremos **agendita** en la cual guardaremos una lista de contactos. Los datos que vamos a manejar son únicamente nombre, email y teléfono. El usuario que tendrá acceso total a esta base de datos es llamado **bingo**, este usuario tendrá una contraseña **holahola**, y además se le permitirá acceso a esta base de datos únicamente cuando se conecte de manera local (**localhost**).

\* Nótese el uso de la sentencia GRANT que se ejecuta a continuación, y la relación que tiene con los datos que se mencionaron anteriormente para permitir el acceso a la base de datos agendita.

```
[eduardo@casita]$ mysqladmin create agendita

[eduardo@casita]$ mysql agendita
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3525 to server version: 3.23.54

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> CREATE TABLE contactos
-> (id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
-> nombre varchar(80), telefono varchar(20), email varchar(60));
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO contactos VALUES
-> (0, 'Pepe Pecas', '8282-7272', 'pepe@hotmail.net');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO contactos VALUES
-> (0, 'Laura Zarco', '2737-9212', 'lauris@micorreo.com');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO contactos VALUES
-> (0, 'Juan Penas', '7262-8292', 'juan@correo.com.cx');
Query OK, 1 row affected (0.00 sec)

mysql> GRANT ALL on agendita.* TO bingo@localhost IDENTIFIED by 'holahola';
Query OK, 0 rows affected (0.06 sec)
```

### Qué son los servlets

Un servlet es un programa en Java usado para extender las capacidades de un servidor cuyas aplicaciones son accedidas vía un modelo petición-respuesta (request-response). A pesar de que un servlet podría ser usado en cualquier servidor de este tipo, actualmente su uso se ha extendido básicamente a las aplicaciones que se ejecutan bajo un servidor web. En este caso, los servlets se ejecutan dentro del entorno del servidor web, y por lo tanto amplían la funcionalidad del mismo al permitir la generación de páginas HTML de manera dinámica. Estos servlets reciben y responden a peticiones de clientes web a través del protocolo HTTP, el Protocolo de Transferencia de Hypertexto.

Para que se puedan ejecutar los servlets es preciso tener un contenedor de servlets que trabaje en conjunción con el servidor web. Este contenedor se encarga entre otras cosas de administrar la carga y descarga de los servlets dentro del servidor web.

Nota: Es muy común usar el término *servlet engine* (algo así como "motor de servlets"), para referirse a dicho contenedor.

Tomcat proporciona el contenedor de servlets necesario para el trabajo con los Java Servlets, y además puede ser usado en conjunción con algunos otros servidores web, tales como Apache, sin embargo, se recomienda en principio utilizar el propio servidor web que se incluye con Tomcat, ya que es la forma más sencilla en la que se puede comenzar de inmediato a trabajar con Java Servlets y JSPs.

Para la versión de Tomcat que se va a utilizar (4.1.27) se tiene soporte para las especificaciones Servlet 2.3 y JSP 1.2 propuestas por Sun Microsystems.

### Preparación del ambiente de desarrollo

Se supone que se tiene ya instalado el Java 2 SDK, y que además se ha obtenido el driver JDBC para MySQL, por lo tanto se indicará únicamente como instalar y configurar el servidor de aplicaciones Tomcat. El procedimiento de instalación que explicaremos a continuación se basa en un sistema operativo Linux RedHat 8.0, sin embargo, debe funcionar en cualquier otra distribución de Linux, incluso en algún otro sistema Unix.

Como se mencionó anteriormente, se usará la versión 4.1.27 de Tomcat, la versión estable más reciente al momento de escribir este artículo. El archivo correspondiente debe descargarse del sitio web de Tomcat.

A continuación se resume el proceso de instalación:

- » Obtener el archivo de distribución de Tomcat para Linux (jakarta-tomcat-4.1.27.tar.gz).
- » Decomprimir y desempaquetar el archivo de distribución de Tomcat.
- » De manera opcional, renombrar el directorio base de Tomcat.

```
[eduardo@casita]$ pwd
/home/eduardo

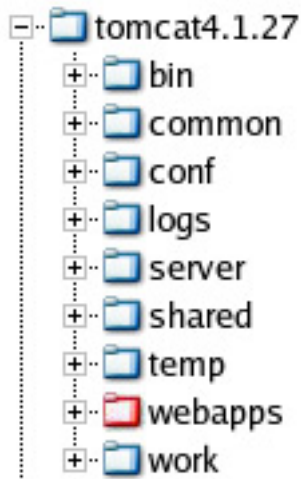
[eduardo@casita]$ ls -lF
jakarta-tomcat-4.1.12.tar.gz*

[eduardo@casita]$ tar xzf jakarta-tomcat-4.1.12.tar.gz

[eduardo@casita]$ ls -lF
jakarta-tomcat-4.1.12/
jakarta-tomcat-4.1.12.tar.gz*

[eduardo@casita]$ mv jakarta-tomcat-4.1.12 tomcat4.1.27
```

Al desempaquetar y descomprimir el archivo con el software de Tomcat, obtenemos una estructura de directorios como la que se muestra en la siguiente figura. En ésta se resalta en color rojo el directorio **webapps** sobre el que trabajaremos posteriormente.



A continuación se describe brevemente cada uno de estos directorios.

Directorio	Descripción
bin/	Incluye los archivos binarios y scripts de Tomcat.
common/	Para clases disponibles tanto para Catalina como para las aplicaciones web.
conf/	Guarda los archivos de configuración.
logs/	Guarda los archivos de registros (logs).
server/	Contiene clases para uso interno y exclusivo de Catalina.
shared/	Contiene clases compartidas por todas las aplicaciones web.
temp/	Guarda archivos temporales creados por la Máquina Virtual de Java.
webapps/	Directorio base para las aplicaciones web.
work/	Directorio de trabajo para archivos y directorios temporales usado por Tomcat.

El nombre de **Catalina** se refiere al contenedor de servlets en sí que se incluye con Tomcat.

En algunos de estos directorios se encuentran los subdirectorios **classes** y **lib**. En el primero se colocan las clases "sueltas", sin empaquetar, y en el segundo se colocan clases empaquetadas en archivo JAR.

Muy bien, ahora vamos a ver como iniciamos y detenemos el servidor Tomcat.

```
[eduardo@casita]$ export CATALINA_HOME=/home/eduardo/tomcat4.1.27
[eduardo@casita]$ export JAVA_HOME=/usr/java/j2sdk1.4.1_01
[eduardo@casita]$ cd tomcat4.1.27
[eduardo@casita]$ ls -l bin/*.sh
bin/catalina.sh
bin/jasper.sh
bin/setclasspath.sh
bin/shutdown.sh
bin/startup.sh
bin/tool-wrapper.sh
```

```
[eduardo@casita]$ ./bin/startup.sh
Using CATALINA_BASE:   /home/eduardo/tomcat4.1.27
Using CATALINA_HOME:   /home/eduardo/tomcat4.1.27
Using CATALINA_TMPDIR: /home/eduardo/tomcat4.1.27/temp
Using JAVA_HOME:       /usr/java/j2sdk1.4.1_01
```

Dentro del directorio **bin** de Tomcat se tienen dos scripts con los cuales podemos iniciar y detener nuestro servidor en el momento que lo queramos. Estos son **startup.sh** y **shutdown.sh** respectivamente. Es muy importante que se fijen las variables de ambiente CATALINA\_HOME y JAVA\_HOME para evitar tener problemas con la ejecución de Tomcat. CATALINA\_HOME corresponde al directorio base o principal de Tomcat, y JAVA\_HOME al directorio base o principal de Java.

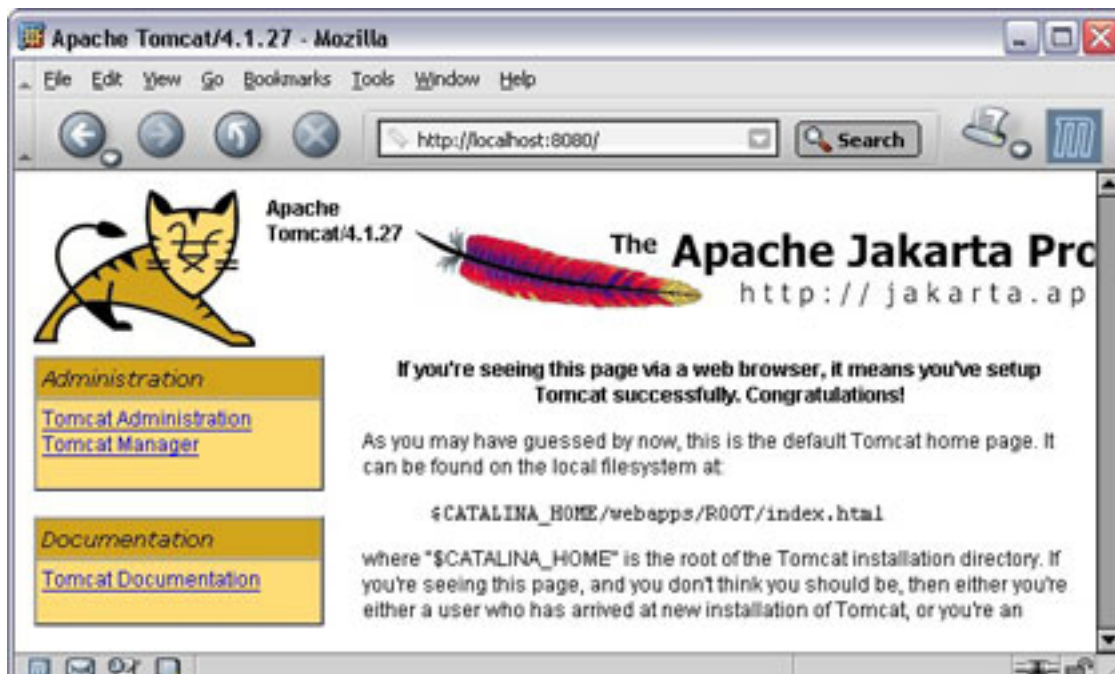
Si no hubo ningún problema, en este momento debe estar ejecutándose en nuestra máquina el servidor web que se incluye con Tomcat, y que como se explicó anteriormente, es el que nos permitirá ejecutar los servlets que escribiremos más adelante.

Un servidor web escucha peticiones en un puerto concreto, normalmente el puerto 80, aunque puede usar prácticamente cualquier otro número de puerto. Si un servidor web está escuchando en otro puerto distinto del 80, las peticiones dirigidas al servidor deben especificar el número de puerto en concreto. En el archivo de configuración de Tomcat se fija por default el puerto con el valor 8080.

Desde el navegador podemos acceder a Tomcat con el siguiente URL:

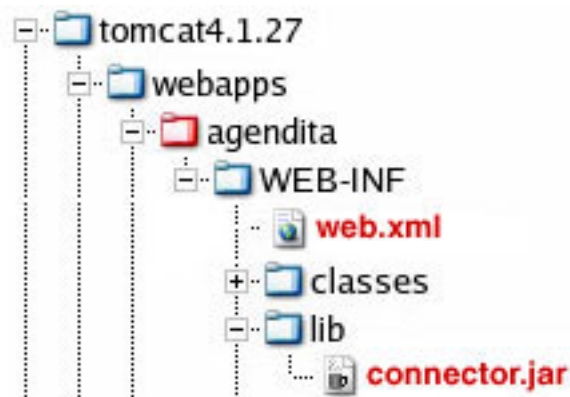
```
http://localhost:8080/
```

Veremos un página como la mostrada en la siguiente figura:



Ahora que hemos verificado que Tomcat está trabajando sin problemas, lo que haremos a continuación es crear una nueva aplicación web. Para que una aplicación web pueda ser ejecutada, ésta debe ser desplegada en el contenedor de servlets. Existen varias maneras en las que una aplicación web puede ser desplegada en Tomcat, nosotros hablaremos únicamente de una de ellas, la que se basa en la creación de una estructura de directorios preestablecida bajo el directorio **webapps**, ya que es la más sencilla y rápida durante la etapa de desarrollo.

La estructura de directorios está basada en la especificación del API Java Servlet, y cualquier contenedor de servlets debería de seguir estas reglas. El nivel superior, o directorio raíz de la aplicación (agendita para nuestro ejemplo) contiene documentos HTML, scripts JSP, imágenes, y algunos otros archivos. A partir de éste, se pueden tener todos los subdirectorios que se requieran. Observar la siguiente figura.



El directorio raíz contiene un directorio especial llamado **WEB-INF**. Este directorio no es visible para los usuarios de la aplicación, sin embargo, aquí se guardan todas las clases, servlets y archivos JAR de los que conste una aplicación web. Dentro del directorio WEB-INF hay dos subdirectorios y un archivo que son de especial interés:

- » **classes** - Este directorio contiene servlets y otras clases. Estas clases se hayan automáticamente por el cargador de servlets como si estuvieran en la ruta de clases (CLASSPATH). Puede incluir subdirectorios que correspondan a la estructura de paquetes.
- » **lib** - Es similar al directorio anterior, pero contiene únicamente archivos JAR.
- » **web.xml** - Es un documento XML llamado *descriptor de despliegue*. Tiene una estructura rigurosamente definida que se usa para configurar los servlets y otros recursos que forman parte de una aplicación web.

De acuerdo con esta estructura de directorios, el archivo connector.jar que contiene el driver JDBC para MySQL se colocará en el directorio agendita/WEB-INF/lib. De hecho, todos los archivos JAR que vayan a ser usados por nuestra aplicación web deben colocarse en este mismo directorio.

Todos los servlets y otras clases de nuestra aplicación que no estén en archivos JAR deben ser colocados en el directorio agendita/WEB-INF/classes.

Cualquier otro archivo (HTML, GIF, JPG, etc) se debe colocar en el directorio agenda.

El archivo agenda/WEB-INF/web.xml debe contener las siguientes líneas:

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

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

<web-app>
  <display-name>
    Agendita
  </display-name>
  <description>
    Es un ejemplo que muestra el funcionamiento de los Java Servlets con
MySQL
  </description>
  <servlet-mapping>
    <servlet-name>invoker</servlet-name>
    <url-pattern>/servlet/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Aunque no es necesario, es recomendable detener y volver a iniciar Tomcat para que los cambios tengan efecto, y reconozca sin problemas la nueva aplicación web después de haber creado la estructura de directorios.

```
[eduardo@casita]$ ./bin/shutdown.sh
[eduardo@casita]$ ./bin/startup.sh
```

Para verificar que todo marcha bien y explicar la forma en como tenemos que compilar los servlets y posteriormente invocarlos desde el navegador (nuestro cliente web), mostraremos el famoso ejemplo "hola mundo", versión Java Servlet.

Vamos a crear un archivo HolaMundo.java que contenga el siguiente código:

```
import java.io.*;

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

public class HolaMundo extends HttpServlet
{
  public void doGet (HttpServletRequest req, HttpServletResponse res)
  throws ServletException, IOException
  {
    PrintWriter out = res.getWriter();

    out.println("Hola Mundo");
    out.close();
  }
}
```

Para compilarlo, bajo el directorio `agendita/WEB-INF/classes` ejecutamos los siguientes comandos:

- Para que se encuentre el comando `javac`

```
[eduardo@casita]$ export PATH=/usr/java/j2sdk1.4.1_01/bin:$PATH
```

- Para que se encuentre el API Java Servlet

```
[eduardo@casita]$ export CLASSPATH=$CATALINA_HOME/common/lib/servlet.jar:.
```

- Verificamos que existe el archivo `HolaMundo.java`

```
[eduardo@casita]$ ls -l
HolaMundo.java
```

- Compilamos el programita

```
[eduardo@casita]$ javac HolaMundo.java
```

- Verificamos que se compiló con éxito el programa

```
[eduardo@casita]$ ls -l
HolaMundo.class
HolaMundo.java
```

Para ver la salida de este servlet, lo invocamos desde el navegador con el URL mostrado a continuación:

```
http://localhost:8080/agendita/servlet/HolaMundo
```

Examinando este URL se observa que la sintaxis usada para invocar un servlet es: `/NombreDeLaAplicaciónWeb/servlet/NombreDelServlet`.

Es muy importante señalar que por default Tomcat no vuelve a cargar los servlets si han sufrido alguna modificación y se han vuelto a compilar. Por ejemplo, si modificamos el programita `HolaMundo.java` y lo compilamos nuevamente, al invocarlo desde el navegador no veremos ningún cambio. Dado que esto es sumamente importante y necesario cuando se están desarrollando y depurando los servlets, a continuación mostraremos como hacer para que Tomcat vuelva a cargar los servlets que hayan sufrido algún cambio.

Tenemos que editar el archivo `server.xml` que está bajo el directorio **conf** de Tomcat y agregar la siguiente directiva apróximadamente en la línea 268, entre el contexto `ROOT` y el contexto `Examples`.

```
<DefaultContext reloadable="true"/>
```

De esta manera, para cada aplicación web que se cree de la misma manera que `agendita`, se tendrá habilitada la recarga de servlets. Cabe mencionar que por razones de eficiencia, esta característica de Tomcat debería de usarse únicamente para sistemas que no se encuentran en producción, es decir, que se encuentran en la etapa de desarrollo.

### El API Java Servlet

Esta API consiste básicamente de dos paquetes:

» **javax.servlet**

En este paquete se definen las clases e interfaces que permiten implementar servlets genéricos, sin algún protocolo en específico.

» **javax.servlet.http**

Este paquete contiene las clases e interfaces que describen y definen el modo de funcionamiento de un servlet que usa el protocolo HTTP.

Por lo tanto, para desarrollar Java Servlets, debemos importar las clases e interfaces contenidas en estos paquetes.

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

La interfaz **Servlet** es la parte central del API Java Servlet. Todos los servlets implementan esta interfaz, ya sea directamente, o más comúnmente, al extender una clase que implemente esta interfaz. Las dos clases en esta API que implementan esta interfaz son **GenericServlet** y **HttpServlet**.

Cuando se está implementando un servicio genérico se puede extender la clase `GenericServlet` que se incluye en el paquete `javax.servlet`, sin embargo, como se mencionó anteriormente, el uso principal de los servlets es cuando se ejecutan bajo el entorno de un servidor web y por lo tanto trabajan con el protocolo HTTP. Es por esta razón que la mayoría de las ocasiones lo que se hará, es crear una clase que extienda a la clase `HttpServlet` del paquete `javax.servlet.http` y así se proporcionen los métodos apropiados para recibir y responder a peticiones específicas de este protocolo.

### El ciclo de vida de un Servlet

Un servlet es manejado a través de un ciclo de vida que especifica como es cargado, instanciado, inicializado, como se manejan las peticiones del cliente, y como deja de prestar servicio el servlet.

El ciclo de vida de un servlet es controlado por el contenedor de servlets del servidor web. Cuando una petición es enviada al servlet, el contenedor ejecuta las siguientes acciones:

1. Si no existe una instancia del servlet, el contenedor
  - » Carga la clase del servlet.
  - » Crea una instancia de la clase del servlet.
  - » Inicializa la instancia del servlet al invocar a su método `init()`.
2. Invoca a su método `service()` pasándole un objeto `Request` y un objeto `Response`.
3. Si el contenedor necesita remover el servlet, éste finaliza la ejecución del mismo al llamar a su método `destroy()`.

### Carga, instanciación e inicialización del servlet

El contenedor de servlets es el responsable de cargar e instanciar servlets. La carga e instanciación puede ocurrir cuando el contenedor de servlets es iniciado, o hasta que el contenedor determina que el servlet es necesario para atender una petición.

Después de que el contenedor carga e instancia la clase del servlet, y antes de que se atiendan las peticiones de los clientes, el contenedor inicializa el servlet. Un servlet que no puede completar su proceso de inicialización lanzará una excepción del tipo `UnavailableException`.

El contenedor inicializa una instancia del servlet al llamar a su método **init()** y pasarle un objeto del tipo **ServletConfig**.

En este método se realizan todas las operaciones únicas en el ciclo de vida del servlet, tales como abrir conexiones a bases de datos, y leer datos de configuración almacenados de manera persistente. Este método es llamado sólo una vez, inmediatamente después de que se ha instanciado el servlet. El método `init()` tiene la siguiente forma:

```
public void init(ServletConfig config)
```

El objeto `config` del tipo **ServletConfig** que recibe el método `init()` es usado por el contenedor de servlets para pasarle información al servlet durante su inicialización. A través de este objeto se disponen de varios métodos, sin embargo quizás uno de los más importantes es el método `getServletContext()` que se describe a continuación:

```
public ServletContext getServletContext()
```

Este método regresa una referencia a un objeto del tipo `ServletContext` que le permite al servlet interactuar con el contenedor de servlets y de esta manera acceder a la información del contexto web en el cuál se está ejecutando el servlet, tal como parámetros de inicialización y recursos asociados con dicho contexto.

### Los métodos de servicio del servlet

La interfaz `Servlet` requiere la definición de un método **service()** para manejar las peticiones de los clientes. Este método es llamado cada vez que el servlet recibe una petición. Generalmente el contenedor de servlets es el encargado de manejar las peticiones concurrentes hacia un mismo servlet al ejecutar el método `service()` en diferentes threads.

El servicio proporcionado por un servlet será entonces definido por el método `service()` de un `GenericServlet` o un `HttpServlet`. Por su parte, la clase abstracta `HttpServlet` agrega algunos métodos adicionales que son llamados automáticamente por el método `service()` para procesar los datos de acuerdo al tipo de petición HTTP (GET, POST, PUT, etc) que hizo el cliente. Estos métodos son **doGet()**, **doPost()** y **doPut()** respectivamente.

Las peticiones GET son el tipo usual de peticiones, y se generan cuando el usuario teclea directamente un URL en la barra de direcciones del navegador, cuando presiona una liga (un link), o cuando usa un formulario HTML en el que no se especifica el método de envío.

Las peticiones POST son generadas cuando se crea un formulario en el que se hace uso explícito del método POST (`method = "post"`).

Aunque esto pudiera parecer algo complejo, la realidad es que cuando se desarrollan servlets, básicamente nos tenemos que concentrar en la definición del método `doGet()` o `doPost()`.

Estos métodos tienen la siguiente forma:

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

El funcionamiento de ambos métodos es prácticamente el mismo, esto es, con el objeto **HttpServletRequest** el servlet extrae la información que es enviada por el cliente cuando éste hace una petición, entonces se procesa dicha petición y posteriormente se envía una respuesta al cliente a través del objeto **HttpServletResponse**.

Algunos de los métodos más importantes de la interfaz `HttpServletRequest` son:

```
public String getParameter(String nombre)
```

Regresa el valor de un parámetro como una cadena, o un valor nulo si el parámetro no existe. Los parámetros contienen información extra que es enviada al servlet cuando se hace una petición. Para los servlets HTTP, los parámetros comúnmente se envían a través de formularios HTML.

Este método debe usarse cuando se está completamente seguro de que el parámetro tiene únicamente un valor. Si el parámetro pudiera tener más de un valor, debemos usar el método `getParameterValues()`.

Si el método `getParameter()` se usa con un parámetro con múltiples valores, el valor regresado será igual al primer valor en el arreglo regresado por `getParameterValues()`.

```
public String[] getParameterValues(String nombre)
```

Regresa un arreglo de cadenas que contienen todos los valores de un parámetro, o un valor nulo si el parámetro no existe. Si el parámetro tiene un único valor, el arreglo tendrá sólo un elemento.

```
Enumeration getParameterNames()
```

Regresa una enumeración de cadenas con los nombres de los parámetros que se enviaron con la petición.

Algunos de los métodos más importantes de la interfaz `HttpServletResponse` son:

```
public abstract PrintWriter getWriter()
```

Permite obtener un objeto `PrintWriter` para escribir la respuesta.

```
public void setCharacterEncoding(String codigoCaracteres)
```

Este método permite especificar el código de caracteres que será utilizado al enviar la respuesta al cliente, por ejemplo, ISO-8859-1, ISO-8859-8-I, o UTF-8. Al llamar al método `setContentType()` con la cadena "text/html" y a este método con la cadena "UTF-8" se obtiene el mismo resultado que al llamar a `setContentType()` y pasarle el valor de la cadena "text/html; charset=UTF-8".

Este método no tiene ningún efecto si es llamado después de que `getWriter()` ha sido llamado, o después de que la respuesta ha sido enviada al cliente.

```
public void setContentType(String tipoContenido)
```

Permite establecer el tipo de contenido que será enviado al cliente.

La mayoría de los servlets generan HTML, en lugar de sólo texto como en nuestro servlet de ejemplo `HolaMundo`. Si necesitamos generar HTML, tendremos que indicarle al navegador que los datos serán enviados en formato HTML. Esto se logra al enviar un encabezado de respuesta `Content-Type` con el valor **text/html**.

En general los encabezados se establecen con el uso del método `setHeader()` del objeto `HttpServletResponse`, aunque se puede hacer uso directamente del método `setContentType()`.

### Finalización del servlet

Cuando el contenedor de servlets determina que un servlet debe dejar de prestar su servicio, se llama al método `destroy()` del servlet. Este método se usa comúnmente para liberar todos los recursos usados por el servlet y ejecutar algunas otras operaciones para una correcta finalización del servlet.

### Desarrollo de la aplicación

Ahora que se han explicado a grandes rasgos cuáles son las características generales de los servlets, vamos a desarrollar una pequeña aplicación en la que se realizan operaciones de consulta, inserción y eliminación de registros en una base de datos MySQL usando el API JDBC.

```
/*
  MostrarAgendita.java

  Este programa realiza una consulta SELECT a la base de datos para mostrar en
  una tabla HTML los contactos de la Agendita.

  - Se da la opción de ver el registro completo de un contacto.
  - Se da la opción de agregar un nuevo contacto.
*/

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

```
public class MostrarAgendita extends HttpServlet
{
    static String login = "bingo";
    static String password = "holahola";
    static String url = "jdbc:mysql://localhost/agendita";

    Connection conn;

    public void init(ServletConfig config) throws ServletException
    {
        try
        {
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            conn = DriverManager.getConnection(url,login,password);
        }
        catch(Exception ex)
        {
            System.out.println(ex.toString());
        }
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
    {
        PrintWriter out ;
        try
        {
            String id, nombre, liga, qry, htmlPage;
            Statement stmt;
            ResultSet rs;

            qry = "SELECT id,nombre FROM contactos";
            stmt = conn.createStatement();
            rs = stmt.executeQuery(qry);

            htmlPage = "<h3 align='center'>Lista de contactos</h3>";
            htmlPage += "<table align='center' border='1' width='60%'>";
            htmlPage += "<tr>";
            htmlPage += "<th>Id</th>";
            htmlPage += "<th>Nombre</th>";
            htmlPage += "</tr>";

            while(rs.next())
            {
                id = rs.getString("id");
                nombre = rs.getString("nombre");
                liga = "<a href='VerContacto?id="+id+"'>"+nombre+"</a>";

                htmlPage += "<tr>";
                htmlPage += "<td>" + id + "</td>";
                htmlPage += "<td>" + liga + "</td>";
                htmlPage += "</tr>";
            }

            htmlPage += "</table>";
            htmlPage += "<br><br>";
            htmlPage += "<div align='center'>";
            htmlPage += "<a href='/agendita/AgregarContacto.html'>";
            htmlPage += "Agregar contacto";
            htmlPage += "</a>";
        }
    }
}
```

```
        response.setContentType("text/html");
        out = response.getWriter();
        out.println(htmlPage);

        rs.close();
        stmt.close();
    }
    catch(Exception ex)
    {
        System.out.println(ex.toString());
    }
}

public void destroy()
{
    try
    {
        conn.close();
    }
    catch(Exception ex)
    {
        System.out.println(ex.toString());
    }
}
}
```

/\*

VerContacto.java

Este programa realiza una consulta SELECT para obtener todos los datos de un contacto conociendo el ID de su registro.

- Se da la opción de eliminar el registro.

\*/

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

public class VerContacto extends HttpServlet
{
    static String login = "bingo";
    static String password = "holahola";
    static String url = "jdbc:mysql://localhost/agendita";

    Connection conn;

    public void init(ServletConfig config) throws ServletException
    {
        try
        {
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            conn = DriverManager.getConnection(url, login, password);
        }
    }
}
```

```
        catch(Exception ex)
        {
            System.out.println(ex.toString());
        }
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
    {
        PrintWriter out ;
        try
        {
            String id, qry, htmlPage;
            Statement stmt;
            ResultSet rs;

            id = request.getParameter("id");
            qry = "SELECT * FROM contactos WHERE id="+id;

            stmt = conn.createStatement();
            rs = stmt.executeQuery(qry);

            htmlPage = "<table align='center' width='60%' border='1'>";

            while(rs.next())
            {
                String nombre = rs.getString("nombre");
                String email = rs.getString("email");
                String telefono = rs.getString("telefono");

                htmlPage += "<tr>";
                htmlPage += "<td>Id</td><td>" + id + "</td>";
                htmlPage += "</tr>";
                htmlPage += "<tr>";
                htmlPage += "<td>Nombre</td><td>" + nombre + "</td>";
                htmlPage += "</tr>";
                htmlPage += "<tr>";
                htmlPage += "<td>Email</td><td>" + email + "</td>";
                htmlPage += "</tr>";
                htmlPage += "<tr>";
                htmlPage += "<td>Telefono</td><td>" + telefono + "</td>";
                htmlPage += "</tr>";

            }

            htmlPage += "</table>";

            htmlPage += "<br><br>";
            htmlPage += "<div align='center'>";
            htmlPage += "<a href='EliminarContacto?id="+id+"'>Eliminar
contacto</a>";
            htmlPage += "<br>";
            htmlPage += "<a href='MostrarAgendita'>Lista de contactos</a>";
            htmlPage += "</div>";

            response.setContentType("text/html");
            out = response.getWriter();
            out.println(htmlPage);
        }
    }
}
```

```
        rs.close();
        stmt.close();
    }
    catch(Exception ex)
    {
        System.out.println(ex.toString());
    }
}

public void destroy()
{
    try
    {
        conn.close();
    }
    catch(Exception ex)
    {
        System.out.println(ex.toString());
    }
}
}

<!--
AgregarContacto.html

Presenta un formulario que es utilizado para llenar los datos de un
contacto y agregarlo a la Agendita.

Los datos son enviados al servlet AgregarContacto.java
-->

<html>
<head>
</head>
<body>
    <form action="/agendita/servlet/AgregarContacto" method="post">
        <table align="center" width="70%" border="0">
            <tr>
                <th colspan="2" align="center">
                    Agregar un contacto a mi Agendita
                </th>
            </tr>
            <tr>
                <td>
                    Nombre:
                </td>
                <td>
                    <input name="Nombre" type="text" size="40">
                </td>
            <tr>
                <td>
                    Email:
                </td>
                <td>
                    <input name="Email" type="text" size="40">
                </td>
            <tr>
            </tr>
        </table>
    </form>

```

```
        <tr>
            <td>
                Teléfono:
            </td>
            <td>
                <input name="Telefono" type="text" size="20">
            </td>
        </tr>
        <tr>
        <tr>
            <td colspan="2" align="center">
                <input name="Enviar" type="submit" value="Guardar">
            </td>
        </tr>
    </table>
</form>
</body>
</html>
```

```
/*
AgregarContacto.java

Este programa recibe los datos de un contacto que llegan del formulario y
los guarda en un nuevo registro en la Agendita con el uso de una sentencia
INSERT.
*/

import java.io.*;
import java.sql.*;
import javax.servlet.http.*;
import javax.servlet.*;

public class AgregarContacto extends HttpServlet
{
    String login = "bingo";
    String password = "holahola";
    String url = "jdbc:mysql://localhost/agendita";

    Connection conn;

    public void init(ServletConfig config) throws ServletException
    {
        try
        {
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            conn = DriverManager.getConnection(url,login,password);
        }
        catch(Exception ex)
        {
            System.out.println(ex.toString());
        }
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
    {
        PrintWriter out ;
```

```
try
{
    String nombre, email, telefono, qry, htmlPage, msg;
    Statement stmt;
    int affectedRows;

    nombre    = request.getParameter("Nombre");
    email     = request.getParameter("Email");
    telefono  = request.getParameter("Telefono");

    qry = "INSERT INTO contactos VALUES ";
    qry += "(0," + "'" + nombre+ "'," + email+ "'," + telefono+ "' )";

    stmt = conn.createStatement();
    affectedRows = stmt.executeUpdate(qry);

    if(affectedRows > 0)
        msg = "El registro ha sido agregado";
    else
        msg = "No se ha podido agregar el registro";

    htmlPage = "<div align='center'>";
    htmlPage += msg;
    htmlPage += "<br><br>";
    htmlPage += "<a href='/agendita/servlet/MostrarAgendita'>";
    htmlPage += "Lista de contactos";
    htmlPage += "</a>";
    htmlPage += "</div>";
    response.setContentType("text/html");
    out = response.getWriter();
    out.println(htmlPage);

    stmt.close();
}
catch(Exception ex)
{
    System.out.println(ex.toString());
}
}

public void destroy()
{
    try
    {
        conn.close();
    }
    catch(Exception ex)
    {
        System.out.println(ex.toString());
    }
}
}
```

```
/*
  EliminarContacto.java

  Se utiliza una sentencia DELETE para eliminar un contacto de la Agendita
  cuando se conoce el ID de su registro.
*/

import java.io.*;
import java.sql.*;
import javax.servlet.http.*;
import javax.servlet.*;

public class EliminarContacto extends HttpServlet
{
    static String login = "bingo";
    static String password = "holahola";
    static String url = "jdbc:mysql://localhost/agendita";

    Connection conn;

    public void init(ServletConfig config) throws ServletException
    {
        try
        {
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            conn = DriverManager.getConnection(url,login,password);
        }
        catch(Exception ex)
        {
            System.out.println(ex.toString());
        }
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
    {
        PrintWriter out ;
        try{
            String id, qry, htmlPage, msg;
            Statement stmt;
            int affectedRows;

            id = request.getParameter("id");
            qry = "DELETE FROM contactos WHERE id="+id;

            stmt = conn.createStatement();
            affectedRows = stmt.executeUpdate(qry);

            if(affectedRows > 0)
                msg = "El registro ha sido eliminado";
            else
                msg = "No se ha podido eliminar el registro";

            htmlPage = "<div align='center'>";
            htmlPage += msg;
            htmlPage += "<br><br>";
            htmlPage += "<a href='/agendita/servlet/MostrarAgendita'>";
            htmlPage += "Lista de contactos";
            htmlPage += "</a>";
            htmlPage += "</div>";
        }
    }
}
```

```
        response.setContentType("text/html");
        out = response.getWriter();
        out.println(htmlPage);

        stmt.close();
    }
    catch(Exception ex)
    {
        System.out.println(ex.toString());
    }
}

public void destroy()
{
    try
    {
        conn.close();
    }
    catch(Exception ex)
    {
        System.out.println(ex.toString());
    }
}
}
```

### Comentarios finales

Después de leer la explicación acerca de los servlets y ver el código de los programas de ejemplo, se supone que debemos ser capaces de entender lo que son los servlets, y como interactúan con un servidor de aplicaciones, así como también identificar el papel que juega el API JDBC para la interacción con una base de datos MySQL, así que lo único que falta es poner a funcionar nosotros mismos la aplicación web, revisarla, examinarla y estudiarla para un completo y total entendimiento.

El código fuente de los servlets y un respaldo de la base de datos Agendita puede descargarse de la siguiente liga:

Archivos del [artículo](#)