Material descargable

egg

Java Database Connectivity (JDBC)

JDBC

Java™ Database Connectivity (JDBC) es la especificación JavaSoft de una interfaz de programación de aplicaciones (API) estándar que permite que los **programas Java accedan a sistemas de gestión de bases de datos**. La API JDBC consiste en un conjunto de interfaces y clases escritas en el lenguaje de programación Java. Con estas interfaces y clases estándar, los programadores pueden escribir aplicaciones que se conecten con bases de datos, envíen consultas escritas en el lenguaje de consulta estructurada (SQL) y procesen los resultados. Puesto que JDBC es una especificación estándar, un programa Java que utilice la API JDBC puede conectar con cualquier sistema de gestión de bases de datos (DBMS), siempre y cuando haya un driver para dicho DBMS en concreto.

Componentes de JDBC

En general, hay dos componentes principales de JDBC a través de los cuales puede interactuar con una base de datos. Son los que se mencionan a continuación:

JDBC Driver Manager: carga el driver específico de la base de datos en una aplicación para establecer una conexión con una base de datos. Se utiliza para realizar una llamada específica de la base de datos a la base de datos para procesar la solicitud del usuario.

API JDBC: Es un conjunto de interfaces y clases, que proporciona varios métodos e interfaces para una fácil comunicación con la base de datos. Proporciona dos paquetes de la siguiente manera que contiene las plataformas java SE y java EE para exhibir capacidades WORA (write once run everything).

Estos paquetes son:

- java.sql.*;
- javax.sql.*;

Las clases e interfaces principales de JDBC son:

- java.sql.DriverManager
- java.sql.Connection
- java.sql.Statement
- java.sql.ResultSet
- java.sql.PreparedStatement
- javax.sql.DataSource

Acceso a Base de Datos con JDBC

JDBC nos permitirá acceder a bases de datos desde Java. Para ello necesitaremos contar con un SGBD (sistema gestor de bases de datos) además de un driver específico para poder acceder a este SGBD. La ventaja de JDBC es que nos permitirá acceder a cualquier tipo de base de datos, siempre que contemos con un driver apropiado para ella.

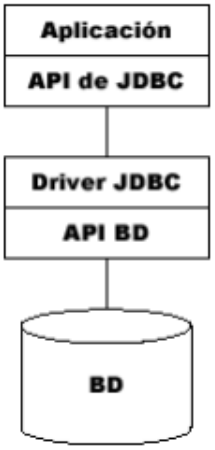


Figura 1: Arquitectura de JDBC

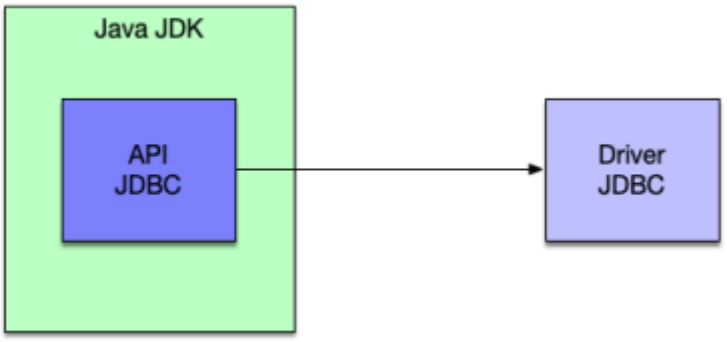
Como se observa en la Figura 1, cuando se construye una aplicación Java utilizando JDBC para el acceso a una base de datos, en la aplicación siempre se utiliza la API estándar de JDBC, y la implementación concreta de la base de datos será transparente para el usuario.

¿Qué es un driver JDBC?

Vimos que dentro de los componentes de JDBC existe el Driver Manager que es el encargado de cargar el driver, pero que es el driver exactamente.

La API JDBC define las interfaces y clases Java™ que utilizan los programadores para conectarse con bases de datos y enviar consultas. Un driver JDBC implementa dichas interfaces y clases para un determinado proveedor de DBMS.

Un programa Java que utiliza la API JDBC carga el controlador especificado para el DBMS particular antes de conectar realmente con una base de datos. Luego la clase JDBC DriverManager envía todas las llamadas de la API JDBC al controlador cargado.



Cada base de datos debe aportar sus propias implementaciones y es ahí donde el Driver JDBC realiza sus aportes.

El concepto de Driver hace referencia al conjunto de clases necesarias que implementa de forma nativa el protocolo de comunicación con la base de datos, en un caso será Oracle y en otro caso será MySQL.

Por lo tanto, para cada base de datos deberemos elegir su Driver. ¿Cómo se encarga Java de saber cuál tenemos que usar en cada caso? Muy sencillo, Java realiza esta operación en dos pasos. En el primero registra el driver con la instrucción:

```java
Class.forName("com.mysql.cj.jdbc.Driver");
```

Una vez registrado el Driver , este es seleccionado a través de la propia cadena de conexión que incluye la información sobre cuál queremos usar, en la siguiente línea podemos ver que una vez especificado el tipo de conexión define el Driver “MySQL”.

```java
String url= "jdbc:mysql://localhost:3306/biblioteca";*
```

* En este ejemplo, la URL contiene el HOST (localhost), el PUERTO (3306) y la BASE DE DATOS (biblioteca) con la que se desea establecer la conexión. Estos son los elementos mínimos e indispensables necesarios para conectarse.

Componentes del API de JDBC

Anteriormente mencionamos los componentes del API de JDBC; ahora profundizaremos en cada uno de ellos.

Driver: Es el enlace de comunicaciones de la base de datos que maneja toda la comunicación con la base de datos. Normalmente, una vez que se carga el controlador, el desarrollador no necesita llamarlo explícitamente.

Connection: Es una interfaz con todos los métodos para contactar una base de datos. El objeto de conexión representa el contexto de comunicación, es decir, toda la comunicación con la base de datos es solo a través del objeto de Connection.

Statement: Encapsula una instrucción SQL que se pasa a la base de datos para ser analizada, compilada, planificada y ejecutada.

PreparedStatement: Es una extensión de Statement que permite precompilar instrucciones SQL. Esto mejora el rendimiento y la seguridad, ya que evita las inyecciones SQL y reutiliza las mismas consultas SQL con diferentes parámetros.

ResultSet: Los ResultSet representan un conjunto de filas recuperadas debido a la ejecución de una consulta.

Conexión con la Base de Datos

Para comunicarnos con una base de datos utilizando JDBC, se debe en primer lugar establecer una conexión con la base de datos a través del driver JDBC apropiado.

La API JDBC especifica la conexión en la interfaz java.sql.Connection. La clase DriverManager permite obtener objetos Connection con la base de datos.

Para conectarse es necesario proporcionar:

- URL de conexión, que incluye: o Nombre del host donde está la base de datos. o Nombre de la base de datos a usar.
- Nombre del usuario en la base de datos.
- Contraseña del usuario en la base de datos.

El siguiente código muestra un ejemplo de conexión y obtención de datos en JDBC a una base de datos MySQL:

```java
Connection connection = null;

try { Class.forName("com.mysql.jdbc.Driver");

       String url = “jdbc:mysql://hostname/database-name”;

       connection = DriverManager.getConnection(url, “user”, “password”); }

} catch (SQLException ex) {

       connection = null;

       ex.printStackTrace();

       System.out.println(“SQLException:      ”      +      ex.getMessage());
       System.out.println(“SQLState:       ”       +       ex.getSQLState());
       System.out.println(“VendorError: ” + ex.getErrorCode());

}
```

En este ejemplo, primero se revisa el driver con la sentencia ClassForName. Después, la clase DriverManager intenta establecer una conexión con la base de datos databasename utilizando el driver JDBC que proporciona MySQL. Para poder acceder al RDBMS MySQL es necesario introducir un username y un password válidos. En el API JDBC, hay varios métodos que pueden lanzar la excepción SQLException.

Conexión a la Base de Datos (Objeto Connection)

Una vez cargado el driver apropiado para nuestro SGBD se debe establecer la conexión con la BD.

Para ello se utiliza el siguiente método:

```java
Connection con = DriverManager.getConnection(url, login, password);
```

La conexión a la BD está encapsulada en un objeto Connection, y para su creación se debe proporcionar la url de la BD y el username y password para acceder a ella. El formato de la url variará según el driver que se utilice.

El objeto Connection representa el contexto de una conexión con la base de datos, es decir:

- Permite obtener objetos Statement para realizar consultas SQL.
- Permite obtener metadatos acerca de la base de datos (nombres de tablas, etc.)
- Permite gestionar transacciones.

Para quienes deseen explorar más sobre los componentes del API de JDBC y su funcionamiento, recomendamos consultar la **documentación oficial** proporcionada por Oracle. Allí encontrarán información detallada, ejemplos y guías actualizadas para implementar JDBC en sus proyectos.