JDBC

Bryan Ivan Montiel Ortega

Noviembre 2023

Índice

Why JDBC?

JDBC es una API de Java para ejecutar sentencias SQL. (Como punto de interés, JDBC es nombre de una marca registrada y no es un acrónimo, a pesar de todo, JDBC es a menudo interpretado como “Java DataBase Connectivity”). Consta de un conjunto de clases e interfaces escrito en lenguaje de programación Java.

Usando JDBC es fácil enviar sentencias SQL a virtualmente cualquier ‘base de datos relacional’. En otras palabras, con la API JDBC no es necesario escribir un programa para acceder a una base de datos tipo. Access, otro programa para acceder a una base de datos tipo Oracle y así para cada tipo de base de datos.

Uno puede escribir un solo programa usando la API JDBC y el programa será capaz de enviar sentencias SQL a la base de datos apropiada. Y, con una aplicación escrita en Java, uno no tiene por qué preocuparse por escribir diferentes programas para diferentes plataformas. La combinación de JDBC permite al programador escribir una vez y ejecutar en cualquier sitio.

Java, siendo robusto, seguro, fácil de usar, fácil de entender, y automáticamente descargable en una red, es un excelente lenguaje base para aplicaciones con bases de datos. Lo que es necesario es una forma para que las aplicaciones Java puedan entenderse con bases de datos de diferentes tipos. JDBC es el mecanismo para hacer esto.

JDBC extiende lo que puede hacerse con Java. Por ejemplo, con Java y la API de JDBC, es posible publicar una página web que usa información obtenida de una base de datos remota. O una compañía puede usar JDBC para conectar todos sus empleados (incluso si estos están usando un conglomerado de máquinas Windows, Macintosh y UNÍS) a una o más bases de datos internas vía una intranet.

De una forma simple, JDBC posibilita hacer tres cosas:

- Establecer una conexión con una base de datos
- Enviar sentencias SQL
- Procesar los resultados

Driver Manager

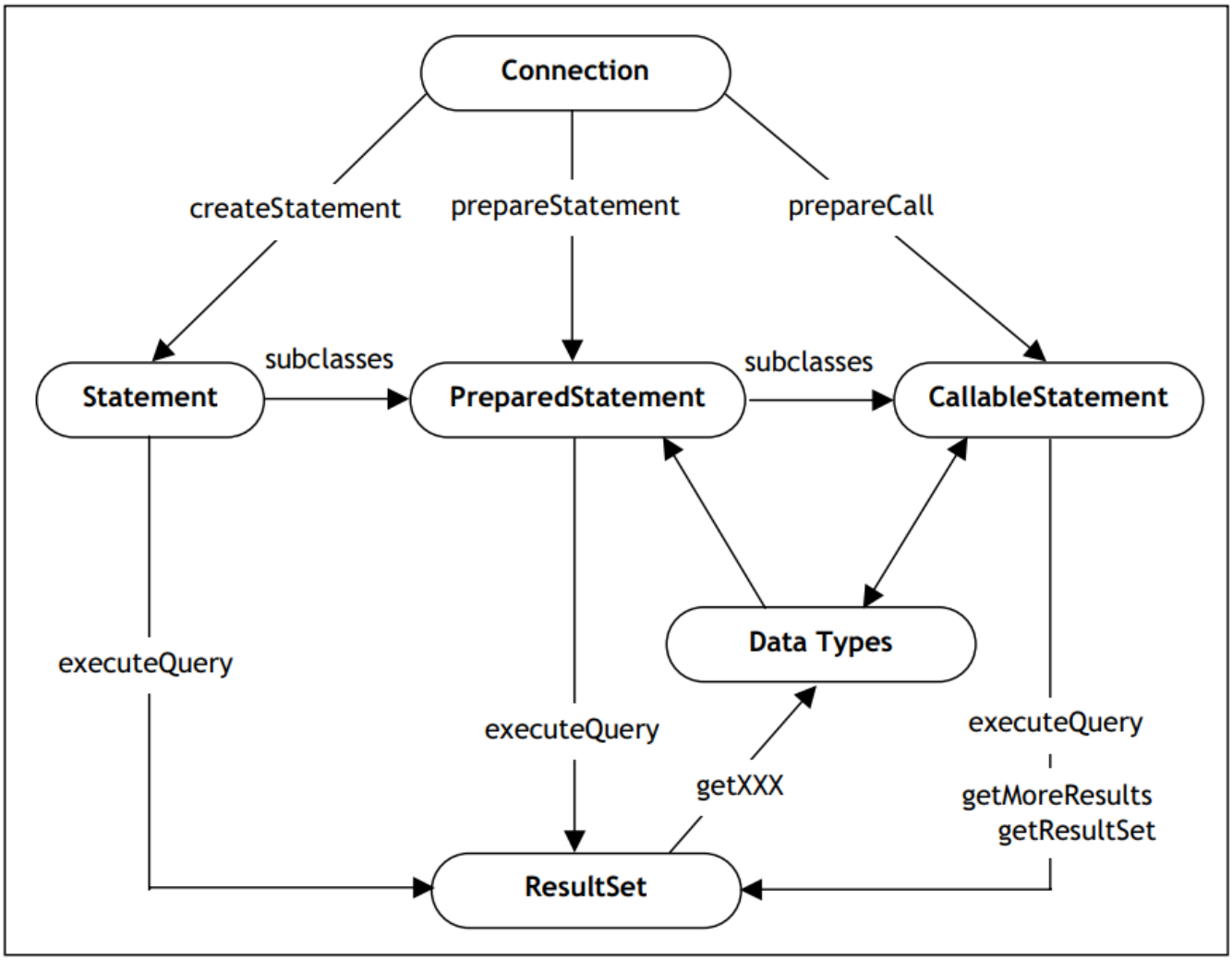
La clase `DriverManager` es la capa gestora de JDBC, trabajando entre el usuario y el controlador (driver). Se encarga de seguir el rastro de los controladores que están disponibles y establecer la conexión entre la base de datos y el controlador apropiado.

La clase `DriverManager` mantiene una lista de clases Driver que se han registrado llamando al método `DriverManager.registerDriver`. Un usuario normalmente no llamará al método `DriverManager.registerDriver` directamente, sino que será llamado automáticamente por el controlador (driver) cuando este se carga. El usuario lo que hace es forzar que se cargue el driver, lo cual puede hacerse de dos formas, aunque la recomendada es llamando al método `Class.forName()`.

Esto carga la clase driver explícitamente. La siguiente sentencia carga la clase `sun.jdbc.odbc.JdbcOdbcDriver`, que permite usar el Puente JDBC-ODBC:

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

La clase `sun.jdbc.odbc.JdbcOdbcDriver` ha sido escrita de forma que al cargarla crea una instancia de ella y llama a DriverManager.registerDriver con esa instancia como parámetro y entonces es añadida a la lista de drivers de DriverManager y está disponible para crear una conexión a una base de datos.



Relación entre las principales clases e interface en el paquete `java.sql`

Statements

Un objeto Statement se usa para enviar sentencias SQL a una base de datos. Una vez que se ha establecido una conexión con una base de datos particular, esa conexión puede ser usada para enviar sentencias SQL. Un objeto Statement se crea con el método creatStatement de Connection como en el siguiente fragmento de código:

```
Connection con = DriverManager.getConnection(url);

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

La sentencia SQL que será enviada a la base de datos es proporcionada como argumento a uno de los métodos para ejecutar un objeto Statement:

```
ResultSet rs = stmt.executeQuery(SELECT a, b, c FROM Table2);
```

Inserting Rows

El siguiente código funciona para insertar el código SQL nativo de a continuación:

```
insert into employees
      (last_name, first_name, email, department, salary)
      values
      'Wright', 'Eric', 'eric.wright@foo.com', 'HR', 33000.00
```

Para insertar se deben seguir los siguientes pasos (Development Process):

1. GetConnection
2. Create Statement
3. Execute SQL insert

En el siguiente fragmento de código se llama a statement.executeUpdate() y se proporciona dentro el string SQL que se va a insertar.

```
String dbUrl = "jdbc:mysql://localhost:3306/demo";
Spring user = "student";
String pass = "student";

Connection myConn = DriverManager.getConnection(dbUrl, user, password);

Statement myStmt = myConn.createStatement();

int rowsAffected = myStmt.executeUpdate(
      "insert into employee " +
      "(last_name, first_name, email, department, salary) " +
      "values " +
      "('Wright', 'Eric', 'eric.wright@foo.com', 'HR', 33000.00)" );
```

Cabe mencionar que el statement.`executeUpdate()` es un método genérico que se puede utilizar para cualquier actualización a la base de datos así que puede usarse para hacer SQL `'inserts'`, `'updates'`, y `'deletes'`.

El método también regresará el número de filas afectadas, y en el caso del código anterior, regresará el valor de 'uno' ya que solo se ha agregado una fila.

Updating Rows

El siguiente código funciona para insertar el código SQL nativo de a continuación:

```
update employees set email='john.doe@luv2code.com'
                 where last_name='Doe'
                 and first_name='John';
```

Para insertar se deben seguir los siguientes pasos (Development Process):

4. GetConnection
5. Create Statement
6. Execute SQL insert

En el siguiente fragmento de código se llama a statement.executeUpdate() y se proporciona dentro el string SQL que se va a insertar.

```
String dbUrl = ...;
Spring user = "student";
String pass = "student";

Connection myConn = DriverManager.getConnection(dbUrl, user, password);

Statement myStmt = myConn.createStatement();

int rowsAffected = myStmt.executeUpdate(
      "update employee " +
      "set email='john.doe@luv2code.com' " +
      "where last_name='Doe' and first_name='John'" );
```

Cabe mencionar que el statement.`executeUpdate()` es un método genérico que se puede utilizar para cualquier actualización a la base de datos así que puede usarse para hacer SQL `'inserts'`, `'updates'`, y `'deletes'`.

Deleting Rows

El siguiente código funciona para insertar el código SQL nativo de a continuación:

```
delete from employees
       where last_name='Doe'
       and first_name='John'
```

En el siguiente fragmento de código se llama a statement.executeUpdate() y se proporciona dentro el string SQL que se va a insertar.

```
String dbUrl = ...;
Spring user = "student";
String pass = "student";

Connection myConn = DriverManager.getConnection(dbUrl, user, password);

Statement myStmt = myConn.createStatement();

int rowsAffected = myStmt.executeUpdate(
     "delete from employees " +
     "where last_name='Doe' and first_name='John'" );
```

De igual manera, es de este modo ya que se ejecutan con el mismo método.

Other Modifying Statements

Se refiere a las declaraciones SQL que modifican los datos en una base de datos, aparte de las operaciones básicas de inserción, actualización y eliminación.

Estas declaraciones incluyen comandos como ALTER TABLE (para modificar la estructura de una tabla), CREATE INDEX (para crear un índice), DROP TABLE (para eliminar una tabla) y otros.

Result Set

Un `ResultSet` contiene todos los registros (filas) que satisfacen las condiciones impuestas en una sentencia SQL y proporciona acceso a los datos en dichos registros a través de un conjunto de métodos get que permiten acceder a los diferentes campos o atributos (columnas) del registro actual. Un `ResultSet` mantiene un cursor que apunta al registro actual. El método `ResultSet.next()` se usa para moverse al siguiente registro del `ResultSet`, haciendo el siguiente registro el registro actual.

Los métodos getXXX proporcionan los medios para obtener los valores de los campos, atributos o columnas del registro actual. Para cada registro, los valores de las columnas pueden ser obtenidos en cualquier orden, pero para la mayor portabilidad, se debe hacer de izquierda a derecha y leer el valor de la columna sólo una vez. Tanto el nombre de la columna como el número de esta puede ser usado para designar la columna de la cual se quiere obtener el valor. Si en el ejemplo anterior la columna "a" es de

tipo entero, la “b” del tipo cadena de caracteres y la “c” de tipo coma flotante, la siguiente porción de código imprimiría los valores de todos los registros:

```
while(rs.next()){
      int i = rs.getInt("a");
      String s = rs.getString("b");
      Float f = rs.getFloat("c");
      System.out.println("ROW= " +i+ " " +s+ " " + f);
}
```

Mapping Between SQL & Java Data Types

El controlador JDBC convierte el tipo de datos Java al tipo JDBC apropiado, antes de enviarlo a la base de datos. Utiliza una asignación predeterminada para la mayoría de los tipos de datos. Por ejemplo, un int de Java se convierte en un INTEGER de SQL. Se crearon asignaciones predeterminadas para proporcionar coherencia entre los controladores.

La siguiente tabla resume el tipo de datos JDBC predeterminado al que se convierte el tipo de datos Java cuando se llama al método setXXX () del objeto `PreparedStatement` o `CallableStatement` o al método `ResultSet.updateXXX ()`.

SQL	JDBC / Java	setXXX	updateXXX
VARCHAR	java.lang.String	setString	updateString
CARBONIZARSE	java.lang.String	setString	updateString
LONGVARCHAR	java.lang.String	setString	updateString
POCO	booleano	setBoolean	updateBoolean
NUMÉRICO	java.math.BigDecimal	setBigDecimal	updateBigDecimal
TINYINT	byte	setByte	updateByte
PEQUEÑO	corto	setShort	updateShort
ENTERO	En t	setInt	updateInt
EMPEZANDO	largo	setLong	updateLong
REAL	flotador	setFloat	updateFloat
FLOTADOR	flotador	setFloat	updateFloat
DOBLE	doble	setDouble	updateDouble
VARBINARIO	byte []	setBytes	updateBytes
BINARIO	byte []	setBytes	updateBytes
FECHA	java.sql.Date	define la fecha	fecha de actualización
HORA	java.sql.Time	fijar tiempo	tiempo de actualizacion
TIMESTAMP	java.sql.Timestamp	setTimestamp	updateTimestamp
CLOB	java.sql.Clob	setClob	updateClob
GOTA	java.sql.Blob	setBlob	updateBlob
FORMACIÓN	java.sql.Array	setARRAY	updateARRAY
ÁRBITRO	java.sql.Ref	SetRef	updateRef

PreparedStatement

Es un SQL statement pre compilado, este tipo de 'statements' tienen los siguientes beneficios:

- Hace más fácil establecer parámetros SQL
- Previene contra ataques de inyección SQL
- Pueden mejorar el rendimiento de la aplicación
    - SQL 'statement' es pre compilado

En lugar de establecer códigos con valores fijos de SQL

```
select * from employees
where salary > 80000 and department='legal'
```

Se pueden colocar 'placeholders', para ello se usa el signo de interrogación '?'

```
select * from employees
where salary > ? and department=?
```

Con lo anterior, se prepara un código como el siguiente (prepared statement)

```
PreparedStatement myStmt =

      myConn.prepareStatement("select * from employees" +
                     " where salary > ? and department=?");
```

Y ahora se puede hacer el uso para setear valores como sigue:

```
myStmt.setDouble(1, 8000);
myStmt.setString(2, 'Legal')

// now execute the query
ResultSet myRs = myStmt.executeQuery();
```

SQLException

Cuando JDBC encuentra un error durante una interacción con una fuente de datos (como una base de datos), lanza una instancia de SQLException en lugar de simplemente Exception.

La instancia de SQLException contiene información útil para determinar la causa del error:

- Descripción del error: Puedes obtenerla llamando al método `SQLException.getMessage()`.
- Código SQLState: Puedes obtenerlo llamando al método `SQLException.getSQLState()`.
- Código de error: Es un valor entero que identifica el error específico. Puedes obtenerlo llamando al método `SQLException.getErrorCode()`.

- Causa: Una instancia de SQLException puede tener una relación causal con otras excepciones. Puedes navegar por esta cadena llamando al método `SQLException.getCause()`.

Referencias

1. Darby, C. (n.d.). Java Database Connection: JDBC and MySQL. Udemy.com. Retrieved November 2, 2023, from https://www.udemy.com/course/how-to-connect-java-jdbc-to-mysql/learn/lecture/2081056#overview
2. Introducción, 8. 1. (n.d.). 8. JDBC: acceso a bases de datos. Ehu.Es. Retrieved November 2, 2023, from http://www.vc.ehu.es/jiwotvim/ISOFT2009-2010/Teoria/BloqueIV/JDBC.pdf
3. JDBC: tipos de datos. (n.d.). Edu.lat. Retrieved November 2, 2023, from https://tutoriales.edu.lat/pub/jdbc/jdbc-data-types/jdbc-tipos-de-datos
4. Tutorial de JDBC. (n.d.). Edu.lat. Retrieved November 2, 2023, from https://tutoriales.edu.lat/pub/jdbc?alias=tutorial-de-jdbc