JDBC

Sitio:	Aula virtual do IES Pazo da Mercé	Impreso por:	Antonio de Andrés Lema
Curso:	Programación (24-25)	Data:	luns, 7 de abril de 2025, 9:40 AM
Libro:	JDBC		

Táboa de contidos

1. JDBC

JDBC (*Java DataBase Connectivity*) é a **API de Java que permite o acceso a bases de datos relacionais**, utilizando a linguaxe SQL. As clases que forman a API de JDBC englóbanse dentro dos paquetes *java.sql* e *javax.sql* (este último para aplicacións executadas no contorno dun servidor). Proporciona unha arquitectura independente do xestor de bases de datos utilizado.

É importante destacar que o xestor de bases de datos pode ser un servidor como MySQL, MariaDB, PostgreSQL ou moitos outros, pero tamén é moi utilizada en aplicacións para dispositivos móbiles as bases de datos SQLite, que non precisan dun servidor de bases de datos e simplemente almacenan todas as táboas nun ficheiro. Todos eles poden ser xestionados con JDBC.

Arquitectura da API JDBC

A arquitectura da API de JDBC baséase nunha serie de interfaces que definen o modo de acceso a calquera base de datos relacional.

Realmente o paquete *java.sql* define fundamentalmente interfaces, e contén poucas clases que implementen funcionalidades propias. Isto é así debido a que a idea consiste en que JDBC simplemente establece como son as clases e funcións que se deben implementar para permitir a manipulación dunha base de datos dende Java, e os distintos fabricantes de xestores de bases de datos poden implementar o seu ***driver JDBC*** que conterá o código necesario para poder acceder a ese xestor de base de datos particular. Esas clases son empaquetadas nun ficheiro *.jar* e serán necesarias para poder acceder a ese xestor de base de datos.

Nos seguintes apartados describiremos as funcións das clases máis relevantes de JDBC, e que nos permitirán levar a cabo as operacións básicas sobre unha base de datos relacional.

2. Conexión coa base de datos

JDBC define unha interface para manexar a conexión coa base de datos, que teremos que establecer en primeiro lugar para logo poder realizar operacións

A interface *Connection*

Esta interface representa unha conexión xenérica cunha base de datos. Ofrece os métodos para executar consultas, manexar transaccións e interactuar coa base de datos.

O problema é: como obtemos esta conexión coa base de datos? A resposta témola na clase que veremos a continuación.

A clase *DriverManager*

Esta clase proporciona o método estático *getConnection(String url, String username, String password)* que permite obter unha conexión a unha base de datos, devolvéndoa en forma dun obxecto da clase *Connection*:

- Os parámetros *username* e *password* permiten indicar o nome de usuario e contrasinal necesarios para conectarse á base de datos respectivamente.
- O parámetro *url* indica a url que especifica a base de datos coa que queremos conectar, que ten a forma *jdbc:nomexestorBD:rutaBD* onde a forma da ruta da base de datos depende do tipo de xestor de base de datos utilizado, pero o usual é que especifique o nome ou IP do servidor, o número de porto e o nome da base de datos coa que queremos conectar.

Vexamos un exemplo de como se obtería unha conexión a unha base de datos MySQL no equipo local (porto 3306) chamada “bd_curso”:

```
try {
  // Obtemos unha conexión coa base de datos
  Connection c = DriverManager.getConnection("jdbc:mysql://localhost:3306/bd_curso","", "");
  System.out.println("Conexion realizada con exito");
} catch (Exception e) {
  e.printStackTrace();
}
```

A clase *DataSource* tamén permite obter conexións a orixes de datos ocultando ao programa os detalles do seu funcionamento interno. Esta clase é utilizada sobre todo en aplicacións que se executan no contorno do servidor.

Cando rematemos de realizar as operacións sobre a base de datos deberemos pechar a conexión, usando o método "***close()***".

3. Execución de consultas

JDBC define varias interfaces para poder executar distintos tipos de consultas sobre unha base de datos relacional. Imos ver a continuación as máis destacables e a súa utilidade.

A interface *Statement*

Esta interface é a que permite executar unha instrución SQL e devolver o seu resultado.

Crearemos un obxecto *Statement* invocando o método *createStatement()* do obxecto *Connection*, e destacamos os seguintes métodos:

- O método *executeUpdate(String sql)* desta interface executa a instrución SQL indicada como parámetro sobre a conexión actual. Devolve como resultado un número enteiro que indica o número de filas afectadas pola execución da instrución.
- O método *close()* permítenos liberar o *Statement* cando xa non o necesitemos.

A continuación podemos ver un exemplo da execución dunha consulta utilizando esta interface:

```
String sql = "INSERT INTO CLIENTE VALUES(3, 'juan')";
Statement st = c.createStatement();
st.executeUpdate(sql);
st.close();
```

A interface *PreparedStatement*

Esta interface estende a interface *Statement* para permitir consultas preparadas, que son consultas nas que podemos introducir valores de certos parámetros, de forma que así podemos executar moitas consultas de forma máis rápida. Por exemplo, nunha consulta de *INSERT* teremos que introducir os distintos valores para a fila dentro da consulta *("VALUES(2,'pepe')")* e se queremos executar varias consultas teremos que ir cambiando estes valores.

Con *PreparedStatement* poderemos introducir unha consulta do tipo *"INSERT INTO CLIENTE VALUES(? , ?)"*, de forma que logo cada interrogante pode ser substituído polo seu valor co método *setTipo(int numeroParámetro, tipo valor)*, onde o tipo pode ser *String*, *int*, *Boolean*, etc.

Ademais de ser máis rápido se executamos varias veces a mesma consulta, é unha forma mais segura de introducir datos nunha consulta xa que os método *setTipo()* fan a transformación correcta entre o tipo de Java e o tipo SQL da base de datos, e protexemos o noso código contra a inxección de código SQL nas entradas da aplicación que pretendan realizar accións non desexadas sobre a BBDD, así que é moi recomendable empregar esta opción en lugar de *Statement*.

Para crear un *PreparedStatement* utilizaremos o método *prepareStatement(string sql)* de *Connection*:

```
PreparedStatement pst = c.prepareStatement("INSERT INTO CLIENTE VALUES(?,?)");
pst.setInt(1, 5);
pst.setString(2, "julian");
pst.executeUpdate();
pst.close();
```

A interface *CallableStatement*

Esta interface é moi similar ao *PreparedStatement*, pero será o que utilicemos se queremos executar un procedemento almacenado nunha base de datos. Para obter un *CallableStatement* utilizaremos o método *prepareCall(String sql)* de *Connection* (neste enlace pódese ver un exemplo de execución dun procedemento almacenado).

4. Procesamento da información obtida da BBDD

Os métodos *execute()* e *executeUpdate()* das interfaces *Statement* e *PreparedStatement* que acabamos de ver, son útiles para a execución de instrucións *INSERT*, *UPDATE* ou *DELETE*, que non devolven ningún tipo de resultado. Pero se executamos unha instrución *SELECT* e queremos procesar a táboa devolta como resultado desa consulta deberemos utilizar o método *executeQuery(String sql)*. Este método devolve un obxecto de tipo *ResultSet*.

A interface *ResultSet*

A interface *ResultSet* representa un conxunto de resultados froito dunha consulta sobre a base de datos.

Proporciona numerosos métodos para poder percorrer as distintas filas do resultado e obter os valores dos distintos campos das filas:

- O método *next()* permite que nos vaiamos movendo a seguinte fila do conxunto de resultados.
- Un gran número de métodos *getTipo()* (*getString(), getInt(), getBoolean()*, etc.) permiten obter os valores dos distintos campos da fila indicando o número do campo dentro da fila (se se pasa como parámetro un número) ou o nome do mesmo (se se pasa como parámetro un String). Java transforma de forma automática os tipos de datos da base datos a tipos de datos de Java.
- Contén tamén distintos métodos para moverse a unha fila determinada, á primeira, á última, á anterior, etc.
- O método *close()* permite pechar o conxunto de resultados liberando os seus recursos.

A continuación podemos ver un exemplo do procesamento do resultado dunha consulta a unha BBDD usando o *ResultSet*.

```
String sql = "SELECT * FROM CLIENTE";
Statement st = c.createStatement();
ResultSet rst = st.executeQuery(sql);

while (rst.next()) {
  System.out.println("Cliente Código:" + rst.getInt("Codigo") + ". Nome:" + rst.getString("Nome"));
}
rst.close();
st.close();
```

Un ResultSet tamén ofrece o método *getMetaData()*, que devolve un obxecto da clase *ResultSetMetaData* con métodos que permiten obter metadatos (datos acerca dos datos) da información obtida da base de datos. A modo de exemplo, o método *getColumnCount()* devolve o número de columnas do ResulSet, e o método *getColumnName(int col)* obtén o nome dunha columna determinada.