Introduction à JDBC

Valère VIANDIER
v.viandier@consultant.com

Utilisation de JDBC

Les accès à une base de données doivent être effectués en sept étapes consécutives.

1. Chargement d'un pilote JDBC
2. Définition de l'URL de connexion.
3. Etablissement de la connexion.
4. Création d'une instruction.
5. Exécution de la requête.
6. Traitement des résultats.
7. Fermeture de la connexion.

Chacune de ces étapes à un coût en terme de temps et de performances. Imaginez des dizaines d'utilisateur effectuant le même scénario sur un site…
Mais concentrons-nous avant tout expliquer et mettre en œuvre chacune des étapes.

Chargement d'un pilote JDBC

Le pilote JDBC est un « logiciel » qui a connaissance des méthodes d'accès à votre base de données. Il existe 4 types de pilotes JDBC.

Ce pilote est disponible sous forme de classe java et généralement dans un pacquage JAR.
La première étape consiste donc à charger cette classe. Pour cela, l'utilisation de la méthode statique Class.forName() nous est d'une aide précieuse. En effet, grâce à l'utilisation de cette classe, votre programme a la possibilité de rester totalement indépendant de la base de données utilisée en conservant le nom du pilote dans un fichier de propriétés.
L'utilisation de la méthode Class.forName() peut lever une exception de type ClassNotFoundException, il convient donc de placer le chargement du pilote dans un bloc sécurisé.

Exemple :

```
Try {
        Class.forName(« oracle.jdbc.driver.OracleDriver ») ;
}
catch(ClassNotFoundException e) {
        System.err.println(« Erreur de chargement du driver : + e) ;
}
```

Pour information, le pilote à charger lors de l'utilisation d'une base de données de type ODBC est « sun.jdbc.odbc.JdbcOdbcDriver ».
Pour obtenir plus d'informations sur les pilotes JDBC, référez-vous à l'adresse suivante :
http://java.sun.com/jdbc/drivers.html

L'utilisation du chargement d'une classe au moment de l'exécution par la méthode Class.forName() requiers que celle-ci se trouve dans votre CLASSPATH.

Définition de l'URL de connexion.

Afin de localiser votre serveur ou votre base de données, il est indispensable de spécifier une adresse sous forme d'URL de type « jdbc: ».
Le format exact de cette URL est dépendant du pilote JDBC utilisé. Je ne peux que vous inviter à consulter la documentation du pilote que vous devez mettre en œuvre. Cependant, pour une connexion à une base de données en utilisant un driver JDBC, l'URL se compose comme suit :
« jdbc:odbc:nomdupiloteodbc »
Par exemple, « jdbc:odbc:comptoir » vous permettra de vous connecter à la base de données Access livrée avec le produit après avoir définis un alias ODBC nommé « comptoir » par le panneau de configuration.

Etablissement de la connexion

La troisième étape doit nous permettre de rentrer en contact, pour la première fois, avec notre base de données. C'est la connexion à proprement parler.
Lors de la connexion, il est fort probable que vous deviez spécifier un certain nombre de paramètres tels que le nom de l'utilisateur et son mot de passe.
Pour l'établissement de la connexion, nous allons utiliser une classe du pacquage java.sql, la classe DriverManager. Celle-ci dispose d'une méthode statique permettant d'obtenir une connexion à notre URL, la méthode getConnection() qui retourne un objet de type Connexion. Cette méthode peut, si la connexion échoue ou si aucun pilote ne prend en charge l'URL spécifiée, une exception de type SQLException.

Exemple :

```
Import java.sql.* ;
…
…

try {
   Connection con = DriverManager.getConnection(url,userId,password) ;
}
catch(SQLException sqle) {
   System.err.println(« Erreur lors de la connexion : » + sqle) ;
}
```

Une fois la connexion établie, vous pouvez obtenir un certain nombre d'informations en utilisant la classe DatabaseMetaData. Une instance de cette classe vous est retournée par la méthode Connexion.getDatabaseMetaData().

Exemple :

```
import java.sql.* ;
…
…

try {
   Connection con = DriverManager.getConnection(url,userId,password) ;
   DatabaseMetaData metaData = con.getMetaData();

   System.out.println(metaData.getDatabaseProductName());
   System.out.println(metaData.getDatabaseProductVersion());
}
catch(SQLException sqle) {
   System.err.println(« Erreur lors de la connexion : » + sqle) ;
}
```

Création d'une instruction.

Afin d'accéder ou de modifier les informations contenues dans votre base de données, il convient d'utiliser un objet de type Statement. Une instance de cet objet est retournée par la méthode Connexion.createStatement() comme ceci :

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

Exécution d'une requête

Après avoir obtenu notre Statement, nous allons nous en servir pour effectuer une requête sur notre base de données. Les requêtes de type sélection doivent utiliser un objet de type ResultSet afin de pouvoir en traiter le résultat.

Exemple :

```
String query = "SELECT * FROM Employés";
ResultSet resultset = statement.executeQuery(query);
```

Si vous devez utiliser une requête de type commande (INSERT, DELETE ou UPDATE), il sera préférable de passer par la méthode executeUpdate() qui, en retour, vous fournira le nombre de lignes affectées par votre requête.

Exemple :

```
String query = "DELETE FROM Employés WHERE Région = 'WA'";
int result = statement.executeUpdate(query) ;
```

Traitement du résultat

L'objet précédemment utilisé nous permet d'avoir un accès aux données résultantes de notre requête en mode enregistrement par enregistrement. Un certain nombre d'accesseurs ont été défini afin de récupérer unitairement les données de chacune des colonnes de notre enregistrement. Chaque accesseur nous permet de récupérer un résultat en spécifiant le numéro de la colonne désirée ou bien son nom. En ce qui concerne la numérotation des colonnes, elle ne commence pas à 0 comme pour les tableaux JAVA, mais à 1.
L'objet ResultSet dispose aussi d'un certain nombre de méthodes permettant de naviguer d'un enregistrement à un autre. Ainsi, la méthode ResultSet.next() nous positionne sur l'enregistrement suivant et nous indique s'il existe un autre enregistrement à suivre ou bien si nous sommes positionnés sur le dernier.
Voici un exemple de code permettant de parcourir l'ensemble des résultats d'une requête.

```
while(resultset.next()) {
  System.out.println(resultset.getString(1)) ;
}
```

Comme vous pouvez le constater, lors de l'exécution de la requête, l'objet ResultSet ne semble pas positionné sur le premier enregistrement mais avant, dans une zone que l'on nomme le GAP.

Fermeture de la connexion

Dernière étape enfin, la fermeture de la connexion à votre base de données. Sans cela, vous risquez de maintenir inutilement des ressources dont vous n'avez plus utilité. Pour fermer explicitement une connexion, utilisez la méthode

```
Connexion.close().
```

Voici un exemple complet, basé sur la base de données « Les comptoirs » d'Access et mettant en œuvre l'ensemble des points évoqués précédemment.

Test.java

```
import java.sql.*;

public class Test {

  public Test() {
    try {
      // Chargement du pilote JDBC
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

      // URL de connexion
      String url = "jdbc:odbc:comptoir";
      // Connexion
      Connection con = DriverManager.getConnection(url);
```

```
      // Création d'une instruction
      Statement statement = con.createStatement();

      // Exécution d'une requete
      String query = "SELECT * FROM Employés";
      ResultSet resultset = statement.executeQuery(query);

      // Traitement des résultats
      while(resultset.next()) {
        System.out.println(resultset.getString(2) + " " + resultset.getString(3));
      }

      // fermeture de la connexion
      con.close();
    }
    catch( ClassNotFoundException e) {
      System.err.println("Erreur lors du chargement du pilote : " + e);
    }
    catch(SQLException sqle) {
      System.err.print("Erreur SQL : " + sqle);
    }
  }


  public static void main(String[] args) {
    Test test = new Test();
  }
}
```