Master MIAGE FC / FA 2e année
Développement d'Applications et Nouvelles Technologies

JDBC

Java Database Connectivity

Loïc Nogues
Novembre 2011

Présentation générale

- Depuis 1996, JDBC est un package contenant des classes et interfaces permettant d'accéder à une base de données à partir d'un programme Java.

- Le point fort :

  Quelle que soit le type de base de données, les programmes écrits avec JBDC fonctionneront, quasiment sans modification du code. Un programme écrit pour accéder aux données sur un serveur *SQL Server de Microsoft fonctionnera* également sur base *Oracle*

Le gestionnaire de pilotes

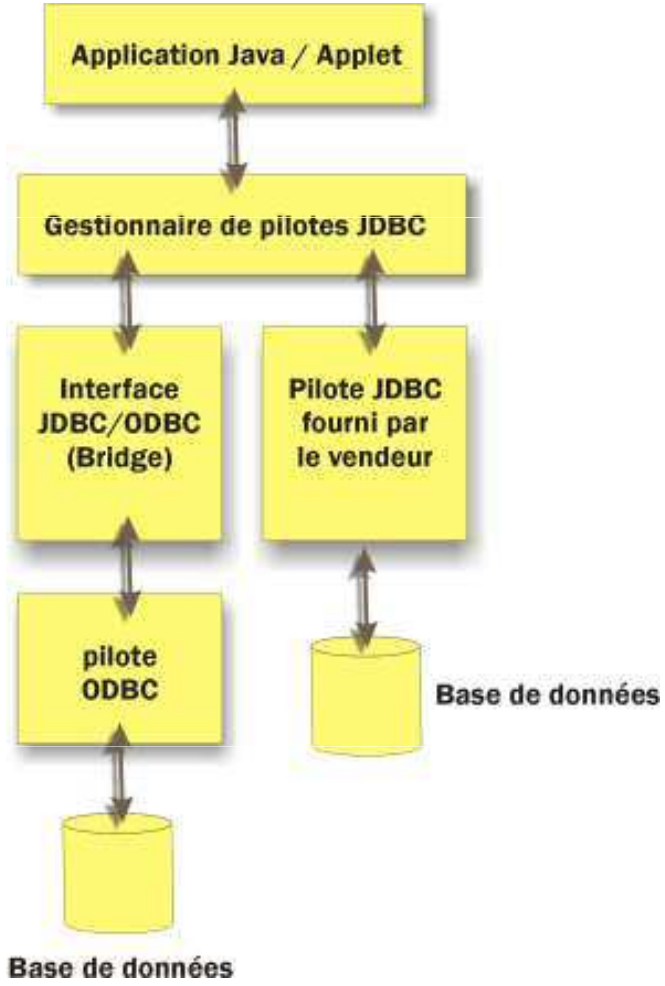
- JDBC Driver Manager
- Il s'agit de l'organe principal de JDBC.
- Il coordonne les différents pilotes, ce qui permet à un même programme d'accéder à plusieurs bases de marque différente.
- La classe DriverManager fournie dans le package java.sql s'exécute dans un environnement statique, ce qui lui permet de gérer efficacement les connexions aux bases de données. C'est également lors de l'utilisation de cette classe, que les différentes pilotes sont enregistrés auprès du gestionnaire.

Le gestionnaire de pilotes

- Deux facons de charger le driver
  - Class.forName("Driver Name").newInstance() ;
  - DriverManager.registerDriver(new Driver Name()) ;

- DriverDifférentes valeurs possibles pour Driver Name
  - JDBC-ODBC sun.jdbc.odbc.JdbcOdbcDriver
  - MySQL com.mysql.jdbc.Driver
  - PostgreSQL org.postgresql.Driver
  - FireBird org.rebirdsql.jdbc.FBDriver
  - Oracle oracle.jdbc.driver.OracleDriver
    oracle.jdbc.OracleDriver

Les pilotes



Les 4 types de drivers :

type 1 : passerelle vers une base de donnees grâce a une autre technologie (ex :JDBC-ODBC via ODBC).

type 2 : mélange de pilotes natifs et de pilotes Java. Les appels JDBC sont convertis en appels natifs.(ex : Oracle)

type 3 : convertit les appels JDBC en un protocole indépendant de la base de données. Un serveur convertit ensuite ceux-ci dans le protocole requis.

type 4 : convertit les appels JDBC directement en une protocole exploite par la base de données. (ex : Oracle,PostgreSQL,Firebird)

Etablir une connexion

Directe

```
Connection con = DriverManager.getConnection(
"jdbc :oracle :thin :@Host Name :Port :DB Name") ;
```

via une DataSource

```
Context initCtx = new InitialContext();
ds = (DataSource) initCtx.lookup("java:comp/env/jdbc/miageBDD");
con = ds.getConnection();
```

Configuration d'un pool de connexion

web.xml :

```
<resource-ref>
<description>
reference a la ressource BDD pour le pool
</description>
<res-ref-name>jdbc/miageBDD</res-ref-name>
<res-type>javax.sql.DataSource</res-type>
<res-auth>Container</res-auth>
  </resource-ref>
```

server.xml :

```
<Context docBase="miagetp4" path="/miagetp4" reloadable="true"
source="org.eclipse.jst.jee.server:miagetp4">
      <Resource name="jdbc/miageBDD" auth="Container"
          type="javax.sql.DataSource" driverClassName="org.postgresql.Driver"
          url="jdbc:postgresql://127.0.0.1:5432/postgres"
          username="postgres" password="postgres" maxActive="8" maxIdle="8"
          maxWait="10000"/>
 </Context>
```

Les transactions

activer / desactiver l'auto commit

con.setAutoCommit(true ou false) ;

faire un commit

con.commit() ;

faire un rollback

con.rollback() ;

Les statements

L'objet connexion est utilisé pour créer des requêtes :

Statement : Permet d'effectuer des requêtes simples comme la lecture, la modification ou l'ajout de quelques enregistrements

```
Connection c = DriverManager.getConnection("…");
Statement s = c.createStatement();
```

PreparedStatement : Permet d'effectuer des requêtes pré compilées , gages de performances plus élevées, surtout lorsque ces requêtes se répètent plusieurs fois. Ces requêtes peuvent également faire passer des paramètres d'exécution

```
PreparedStatement ps = c.prepareStatement("…");
```

CallableStatement : Permet de faire appel à des procédures stockées

```
CallableStatement cs = c.prepareCall("…");
```

Exécution des requêtes (sauf select)

Avec un Statement

```
Statement stmt = conn.createStatement() ;
int result = stmt.executeUpdate(sql) ;
```

Avec un PreparedStatement

```
PreparedStatement pstmt = conn.prepareStatement(sql) ;
int i = 0 ;
pstmt.setString(++i,"V1") ;
pstmt.setObject(++i,"V2") ;
pstmt.setNull(++i, Types.INTEGER) ;
int result = pstmt.executeUpdate() ;
```

Exécution des requêtes select

Avec un Statement

```
Statement stmt = conn.createStatement() ;
ResultSet rs = stmt.executeQuery(sql) ;
```

Avec un PreparedStatement

```
PreparedStatement pstmt = conn.prepareStatement(sql) ;
int i = 0 ;
pstmt.setString(++i,"V1") ;
pstmt.setObject(++i,"V2") ;
pstmt.setNull(++i, Types.INTEGER) ;
ResultSet rs = pstmt.executeQuery() ;
```

Les ResultSet

Parcourir un ResultSet

```
while(rs.next()){
        int j = 0 ;
        ...
        rs.getString(++j) ;
}
```

Atteindre un enregistrement

```
rs.absolute(Num) ;
```

Se deplacer de n enregistrements

```
rs.relative(+/-n) ;
```

Connaître l'enregistrement courant

```
rs.getRow() ;
```

Les ResultSet

Insertion avec un ResultSet

```
rs.moveToInsertRow() ;
rs.updateInt("id",100) ;
rs.updateString("nom","nouveauNom") ;
rs.updateString("prenom","nouveauPrenom") ;
rs.insertRow() ;
if(rs.rowInserted()){
// Enregistrement insere
}
```

Mise a jour avec un ResultSet

```
rs.rst() ;
rs.updateInt("id",100) ;
rs.updateString("nom","nouveauNom") ;
rs.updateString("prenom","nouveauPrenom") ;
rs.updateRow() ;
if(rs.rowUpdated()){
// Enregistrement mis a jour
}
```

Les ResultSet

Suppression avec un ResultSet

```
rs.last() ;
rs.deleteRow;
if(rs.rowDeleted()){
// Enregistrement supprime
}
```

Requêtes groupées

```
PreparedStatement pstmt = conn.prepareStatement(sql) ;
for(int i=0 ;i<10 ;i++){
        pstmt.clearParameters() ;
        pstmt.setXX() ;
        pstmt.addBatch() ;
}
pstmt.executeBatch() ;
```

Les ResultSet

Requêtes groupées

```
Statement stmt = conn.createStatement() ;
for(int i=0 ;i<10 ;i++) {
        stmt.addBatch(sql) ;
}
pstmt.executeBatch() ;
```

Fermeture des connexions

Vous devez fermer toutes les connexions à la base de données

La méthode public void close() est disponible pour les interfaces :

- Statement
- PreparedStatement
- Connection
- ResultSet

Pattern d'utilisation des connexions

```
// 1 - Création de la ressource
try {
    // 2 - Utilisation de la ressource
} finally {
    // 3 - Libération de la ressource
}
```

```
final Connection con = dataSource.getConnection();
try {
    final Statement stmt = cnn.createStatement();
    try {
        final ResultSet rs = cnn.executeQuery("select first_name from customer");
        try {
            while (rst.next()) {
                System.out.println(rst.getString("first_name"));
            }
        } finally {
            rs.close();
        }
    } finally {
        stmt.close();
    }
} finally {
    con.close();
}
```

Questions

?