JDBC : accéder aux bases de données relationnelles depuis Java

Serge Chaumette et Pascal Grange
LaBRI, Laboratoire Bordelais de Recherche en Informatique
Université Bordeaux I
Serge.Chaumette@labri.fr
Pascal.Grange@labri.fr

Copyright

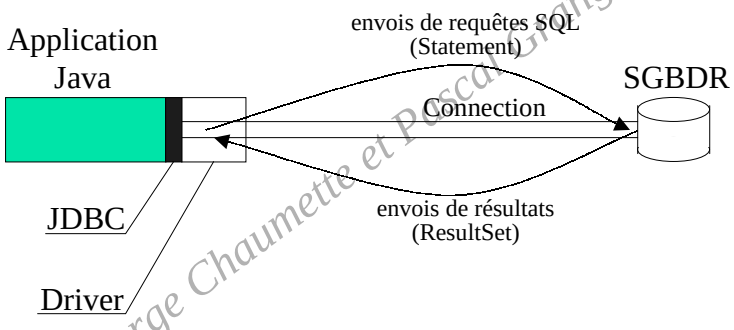
[JDBC]

Qu'est_ce que JDBC ?

- Objectif :
  - accéder aux bases de données relationnelles depuis un programme Java ;
  - conserver une indépendance du programme vis-à-vis du SGBD.
- Moyen : le paquetage java.sql :
  - Connexion à un SGBD distant ;
  - Création – exécution de requêtes SQL.

[JDBC / Architecture]

Fonctionnement général



Les différents types de Drivers

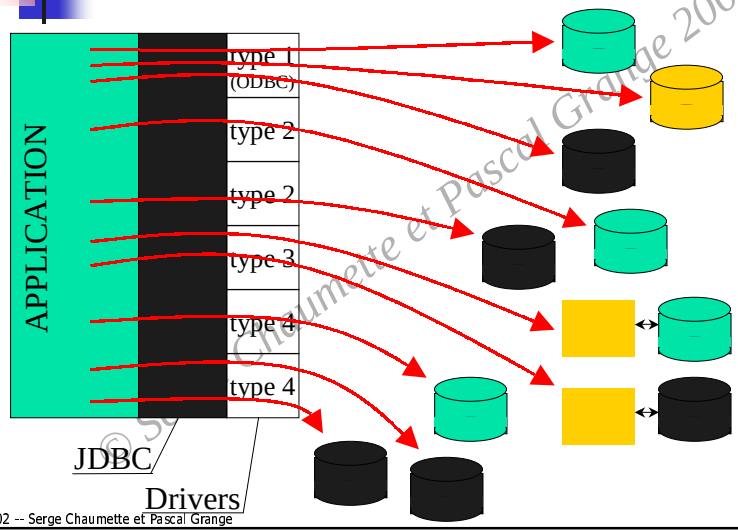
- type 1 : les ponts
  - le pilote JDBC est en fait un pont vers un autre pilote, par exemple ODBC ;
- type 2 : API native
  - le pilote JDBC utilise une API native d'accès à un SGBD spécifique. Fourni par le fabricant du SGBD ;

Les différents types de Drivers (suite)

- type 3 : serveur intermédiaire
  - le pilote JDBC utilise une API réseau générique pour communiquer avec un serveur, et c'est ce dernier qui se charge d'effectuer les requêtes effectives sur le SGBD ;
- type 4 : connexion directe
  - le pilote communique directement avec le SGBD via le protocole réseau spécifié par ce dernier. Fourni par le fabricant du SGBD.

[JDBC / Architecture]

Une application connectée à plusieurs bases de données



[JDBC]

Schéma de fonctionnement d'une application JDBC

1. Enregistrement du pilote du SGBD ciblé
2. Ouverture d'une connexion vers le SGBD
3. Création d'un espace de requêtes
4. Exécution d'une requête SQL
5. Traitement des résultats
6. Fermeture de l'espace de requêtes
7. Fermeture de la connexion

1. Enregistrement du pilote du SGBD ciblé

- Un pilote JDBC est une classe Java ou un ensemble de classes dont le rôle est d'effectuer la correspondance entre les requêtes JDBC et le SGBD ciblé ;
- Lors de l'ouverture d'une connexion JDBC, un pilote approprié doit être choisi. C'est le gestionnaire de pilotes (DriverManager) qui effectue ce choix.

Le gestionnaire de pilotes

- Pour pouvoir être choisi, un pilote doit être enregistré auprès du gestionnaire de pilotes.
- 2 Méthodes d'enregistrement d'un pilote
  - 1. Chargement de la classe qui s'enregistre

```
Class.forName("oracle.jdbc.driver.OracleDriver")
```

  - 2. Déclaration auprès du DriverManager

```
$ java -Djdbc.drivers=oracle.jdbc.driver.OracleDriver:my.sql.Driver …
…
```

2. Ouverture d'une connexion vers le SGBD

- Une connexion vers un SGBD peut être ouverte grâce au gestionnaire de pilotes.
- La méthode DriverManager.getConnection est chargée de trouver le pilote approprié à la connexion souhaitée puis d'effectuer cette connexion.

Les paramètres de la connexion

- Trois paramètres lui sont fournis :
  - l'URL de la base de données à contacter
    - jdbc:subprotocol:subname
    - Exemples
      - jdbc:oracle:thin:@jaguar:1521:base0
      - jdbc:mysql://jaguar:1521/base0
  - le nom d'utilisateur
  - le mot de passe

Exemple

```
import java.sql.*;
public class Connect {
 public static void main(String args[]) throws SQLException {
   // Chargement du pilote...
   try{
     Class.forName("my.sql.Driver");
   } catch(ClassNotFoundException e){
       System.err.println("driver introuvable");
       System.exit(1);
   }
   // Le pilote est bien chargé, ouverture de la connexion...
   Connection c = DriverManager.getConnection("jdbc:mysql:base0",
                                              "moi", "passwd");
   ...
 }
}
```


3. Création d'un espace de requêtes

- Un espace de requêtes est représenté par une instance de la classe Statement
  - associé de façon unique à une connexion
  - créé grâce à la méthode createStatement de la Connection visée.

```
Statement s = c.createStatement();
```

  - c'est ce qui permettra, par la suite, d'effectuer des requêtes sur la base.

4. Exécution d'une requête

- Une fois un Statement créé, une requête peut être exécutée. Deux types de requêtes sont possibles :
  - requêtes de mise à jour de la base

```
int nbCol = s.executeUpdate("INSERT into A
                                    values (1,'a', 3)");
```

  - requêtes de consultation de la base

```
ResultSet r = s.executeQuery("SELECT * from A");
```


5. Traitement des résultats

- Le résultat d'une mise à jour consiste simplement en le nombre de lignes concernées par cette mise à jour ;
- Le résultat d'une consultation de la base est un objet de type ResultSet. Il peut être parcouru ligne à ligne ;
- Lorsque le Statement est réutilisé, le ResultSet devient invalide.

Forme d'un ResultSet

currentRow →

1	2	3	4
NOM	TELEPHONE	PRODUIT	CLE
Dupont	0556984568	chaises	0415
Durand	0142243291	tables	2301
Navarro	0321459012	armoires	6481
Petit	0381058395	buffets	0473
Petit	0381058395	commodes	9473

[JDBC / Étapes d'une application / 5. Traitement des résultats]

Parcours d'un ResultSet

- Le parcours se fait à l'aide d'un curseur se déplaçant de ligne en ligne :
  - la méthode next permet de faire avancer le curseur d'une ligne ;
  - au départ, le curseur se trouve AVANT la première ligne.

Parcours d'un ResultSet (suite)

- Pour une ligne donnée, la récupération de la valeur d'une colonne se fait :
  - soit grâce au nom de la colonne

```
<result set>.getString(<nom de colonne>);
<result set>.getInt(<nom de colonne>);
…
```

  - soit grâce à son indice.

```
<result set>.getString(<indice de colonne>);
<result set>.getInt(<indice de colonne>);
…
```


Exemple de programme parcourant un *ResultSet*

```
...
ResultSet r = s.executeQuery("select * from fournisseurs");
System.out.println("Liste des fournisseurs");
while(r.next()){
  System.out.print(r.getString("NOM") + " ");
  System.out.println(r.getString(2));
}
...
```

```
Liste des fournisseurs
Dupont 0556984568
Durand 0142243291
Navarro 0142243291
Petit 0381058395
```

6. Fermeture de l'espace de requêtes

- Lorsqu'un Statement ne sera plus utilisé pour des requêtes ultérieures, il doit être fermé afin de libérer les ressources de la base de données et de JDBC utilisées.

```
s.close();
```

- Si un ResultSet était disponible, celui-ci est automatiquement fermé :

7. Fermeture de la connexion

- Une connexion JDBC doit être fermée lorsqu'elle n'est plus utilisée dans le programme. Ceci permet de libérer les ressources qui lui sont associées :

```
c.close();
```

[JDBC / Exemple de programme complet]

Exemple de programme complet

```
import java.sql;
public class TestSQL {
  public static void main(String args[])
    throws SQLException{
    //Chargement du pilote...
    try{
      Class.forName("my.sql.Driver");
    }catch(ClassNotFoundException e){
      System.err.println("pilote introuvable");
      System.exit(1);
    }
    // Ouverture de la connexion...
    Connection c = DriverManager.
                   getConnection("jdbc:mysql:base0",
                                 "moi", "passwd");
    // Création d'un espace de requêtes...
    Statement s = c.createStatement();
    // Exécution d'une requête...
    ResultSet r = s.executeQuery("select * from a");
    // Affichage des résultats...
    while(r.next()){
      System.out.println(r.getString("NOM"));
    }
    s.close();
    c.close();
  }
}
```


[JDBC / Les transactions]

Les transactions

- L'objectif est d'assurer l'atomicité d'une suite de requêtes

suite de requêtes

validation annulation

[JDBC / Les transactions]

Mise en oeuvre

- La gestion des transactions se fait à l'aide de la classe Connection.
  - Validation des opérations

```
void commit() throws SQLException;
```

  - Annulation des opérations

```
void rollback() throws SQLException;
```

- *JDBC compliant* ➔ support des transactions

[JDBC / Instructions préparées]

Instructions préparées

- *PreparedStatement* représente une instruction préparée ;
- un paramètre dans la requête SQL se note *?* ;
- les paramètres doivent être spécifiés avant chaque appel ;
- le commit n'est jamais automatique.

Exemple d'utilisation

```
...
PreparedStatement ps = connection.prepareStatement(
                                "select * from A where NOM = ?");
for(int i = 0; i < n; i++){
  //Spécification de la valeur du paramètre (1)...
  ps.setString(1, nom[i]);
  // Appel de l'instruction prédéfinie...
  ps.execute();
}
// Validation de la transaction...
connection.commit();
ps.close();
...
```


Procédures stockées

- CallableStatement représente une procédure stockée ;
- le type des paramètres en sortie doit être spécifié avant le premier appel ;
- les paramètres en entrée doivent être spécifiés avant chaque appel ;
- le commit n'est jamais automatique.

Exemple d'utilisation

```
...
CallableStatement cs = connection.prepareCall(
                                                "{call procStockee[(?,?)]}");
//Enregistrement du paramètre 1 comme paramètre en sortie de type
   float...
cs.registerOutParameter(1, java.sql.Types.FLOAT);
for(int i = 0; i < n; i++){
  //Spécification de la valeur du paramètre 2...
  cs.setInt(2, i);
  // Appel de la procédure stockée...
  cs.execute();
}
// Validation de la transaction...
connection.commit();
cs.close();
...
```


Les méta-données

- *DatabaseMetaData*
  - informations sur la base de données
    - description des tables
    - procédures stockées
    - etc.

```
Connection c;
...
DatabaseMetaData md = c.getMetaData();
...
```

- Voir la documentation de l'API

Les méta-données

- *ResultSetMetaData*
  - informations sur un *ResultSet*
    - type des colonnes
    - nombre de colonnes
    - etc.

```
ResultSet r;
...
ResultSetMetaData md = r.getMetaData();
...
```

  - Voir la documentation de l'API

Références

- le site de Sun sur JDBC :
  - http://java.sun.com/products/jdbc
- livre *officiel* sur JDBC
  - *JDBC Database Access with Java*
  - http://java.sun.com/docs/books/jdbc/summary.html
- les drivers JDBC
  - http://industry.java.sun.com/products/jdbc/drivers