Survol de JDBC

Master Pro SIRES
Bruno MERMET
2008-2009

Plan

- Principes généraux
- Présentation de la notion de Statement
- Requêtes de Consultation
- Requêtes de Mise à jour et requêtes du LDD
- Requêtes paramétrées
- Cas particulier : création d'une base de données
- Approfondissment : requêtes en mode Batch

Présentation Générale

- JDBC = Java DataBase Connectivity
- Principe général
  - De base
    - Un « driverManager » (java.sql.DriverManager)
    - Un ensemble d'interfaces EI dans :
      - java.sql
      - javax.sql
  - Par SGBD
    - Un fichier jar contenant un ensemble de classes implantant les interfaces EI, et notamment un « driver »

Se connecter à une base de données

- Charger le driver
  - Class.forName(« nomDriver »);
  - Effet : Enregistre le driver dans le driverManager
  - Exemple pour Mysql :
    - Class.forName("com.mysql.jdbc.Driver");
- Demander au driverManager une connexion à la base
  - DriverManager.getConnection(« urlBase », « login », « mdp »);
  - Renvoie un objet implantant l'interface java.sql.Connection
- Attention : le driver doit être accessible via le CLASSPATH (ce dernier doit en général contenir le fichier jar fourni par le SGBD)
- Format de urlBase:

  jdbc:nomSGBD://nomMachine:portSGBD/base

Interagir avec une base de données

- Soit cnx un objet implantant java.sql.Connection
- Statement
  - Pour une requête standard à usage unique
  - Exemple : Statement s = cnx.createStatement();
- PreparedStatement (hérite de Statement)
  - Pour une requête paramétrée pour plusieurs usages (la requête sera pré-compilée dans le SGBD)
  - Exemple : PreparedStatement = cnx.prepareStatement(requete);
- CallableStatement (hérite de PreparedStatement)
  - Pour appeler des procédures stockées
  - Non présenté dans ce rapide mémento

Fonctionnement général simplifié d'un Statement

- Fonctionnement général
  - Création de l'objet Statement
  - Utilisation du Statement pour exécuter une requête
  - Analyse éventuelle du résultat de la requête
  - Fermeture du Statement
- Attention :
  - Toute réutilisation du Statement pour une nouvelle requête rend invalide le résultat fourni précédemment par le Statement
  - Par défaut, un *commit* est effectué après toute requête
- 2 types principaux de requête
  - Celles renvoyant des tuples en résultat
  - Les autres

Requêtes de consultation

- Méthode utilisée
  - public ResultSet executeQuery(String req)
- Exemple
  - ResultSet rs;
  - Statement s = cnx.createStatement();
  - rs = s.executeQuery(« select * from clients »);
  - ...
  - rs = s.executeQuery(« select nom from produits »);
- Exception
  - java.sql.SQLException

Exploitation d'un résultat

- Extrait de l'API de l'interface ResultSet (toutes ces méthodes peuvent générer des SQLException)
  - boolean next()
    - Passe à la ligne suivante (la première lors du premier appel)
    - Renvoie faux s'il n'y a plus de ligne
  - String getString(int numCol) / String getString(String nomCol)
  - int getInt(int numCol) / double getInt(String nomCol)
  - double getDouble(int numCol) / double getDouble(String nomCol)
  - …
  - boolean wasNull()
    - Renvoie vrai si le dernier accès à une colonne correspondait à une valeur nulle.

Cas particulier des dates

- To do...

Parcours d'un ResultSet

- Par défaut
  - Seul parcours possible = next()
  - Pas de modification possible
- Options (paramétrables lors de la création du Statement)
  - Déplacement avant/arrière
  - Déplacement relatif/absolu
  - Possibilité de modifier les tuples
  - Existence d'une ligne spéciale pour insérer un tuple

Utilisation d'un ResultSet pour les modifications

- Création :

```
Statement st = cnx.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                   ResultSet.CONCUR_UPDATABLE);
ResultSet rs = st.executeQuery("SELECT nom FROM Produit");
```

- Modification :

```
rs.absolute(2);// on se positionne sur la deuxième ligne
rs.updateString("nom", "clous"); // on modifie le champ « nom » de la ligne courante dans le ResultSet
rs.updateRow();// on valide la modification dans la base de données
```

- Ajout :

```
rs.moveToInsertRow();// on mémorise la ligne courante puis on va à la ligne « spéciale insertion »
rs.updateInt(1, 10); rs.updateInt(2, « clous »); rs.updateInt(3, 200); // on remplit le tuple à insérer
rs.insertRow(); // on insère réellement
rs.moveToCurrentRow(); // on revient à la ligne mémorisée
```

Requêtes de mise à jour

- Méthode utilisée
  - public int executeUpdate(String req)
  - Renvoie le nombre de lignes modifiées
- Exemple
  - int res;
  - Statement s = cnx.createStatement();
  - res = s.executeUpdate(« insert into produits values (10,'clous',200) »);
  - res = s.executeUpdate(« delete from produits where nom='vis' »);
  - res = s.executeUpdate(« update clients set nom= 'martin' where id=10 »);
- Exception
  - java.sql.SQLException

Mises à jour : commit/rollback

- Par défaut
  - Toute mise à jour est suivie d'un commit
- Commit manuel
  - Supprimer les commits automatiques :

    cnx.setAutoCommit(false);
  - Demander un commit (depuis le dernier) :

    cnx.commit();
  - Demander une annulation :

    cnx.rollback() (depuis le dernier commit) ;
  - Utilisation de point de récupération :

    Savepoint s = cnx.setSavePoint(String nom)

    cnx.rollback(Savepoint s)

Requêtes du LDD

- Utilisation de executeUpdate
- Exemple :

```
Statement s = cnx.createStatement();
s.executeUpdate("create table eleve (
        id int primary key,
        nom varchar(30) not null,
        prenom varchar(20) not null,
        groupe varchar(2) not null,
        demigroupe int,
        constraint foreign key (groupe) references groupe(nom))");
```

Requêtes paramétrées

- Peuvent être de mise à jour ou de consultation

  Ici, on n'illustre que sur des requêtes de consultation

- Principe :
  - Création d'un PreparedStatement où les paramètres sont remplacés par des « ? »
  - Avant l'exécution, faire un set*Type*(int numPointInterrog, *Type* val) pour chaque « ? »
  - Exécuter la requête (executeUpdate ou executeQuery suivant le cas)
- Exemple :

```
PreparedStatement ps = cnx.prepareStatement(« select * from produit where id = ? »);
ps.setInt(1, 10);// le 1er point d'interrogation est remplacé par la valeur 10.
ResultSet rs = ps.executeQuery();
```

Créer une base de données (cas MySql)

```
Connection cnx = null;
try {
        Class.forName("com.mysql.jdbc.Driver");
        cnx = DriverManager.getConnection("jdbc:mysql://localhost:3306/", loginRootSGBD, mdpRootSGBD);
} catch (Exception e) {
        System.out.println("connexion ratée : " + e);
        System.exit(1);
}
try {
        Statement stm = cnx.createStatement();
        stm.executeUpdate("create database " + nomBase");
catch (SQLException e) {
        System.err.println("erreur création base");
        System.exit(2);
    }
  }
```

Requêtes en mode Batch

- But
  - Utiliser un même Statement pour exécuter plusieurs requêtes d'un coup
- Méthodes à utiliser
  - Gestion de la liste des requêtes
    - void addBatch(String requête) : ajout d'une requête
    - void clearBatch() : vide la liste des requêtes
  - Exécution
    - int[] executeBatch()
  - Exploitation des résultats
    - boolean getMoreResults()
    - ResultSet getResultSet()

ExecuteBatch()

- Principe

  Exécute les différentes requêtes dans leur ordre d'ajout

- Retour
  - Un tableau d'entier avec dans la $i^{ème}$ case (indice i-1) un entier représentant le résultat de la requête i :
    - N >= 0 si N lignes ont été modifiées par la requête
    - SUCCESS_NO_INFO si nb lignes modifiées inconnu
- Parcours des différents résultats
  - Passage au résultat suivant :
    - Boolean getMoreResults():
      - Renvoie vrai s'il s'agit d'un ResultSet (requête de sélection)

        Faire alors un getResultSet();
      - Renvoie faux sinon (requête de mise à jour par exemple ou plus de résultat)
  - Être sûr qu'il n'y a plus de résultat :

    !getMoreResults() && getUpdateCount() == -1