Introduction à JDBC

Accès aux bases de données en Java

Eric Cariou

Université de Pau et des Pays de l'Adour
Département Informatique

Eric.Cariou@univ-pau.fr

Introduction

- JDBC : Java Data Base Connectivity
- Framework permettant l'accès aux bases de données relationnelles dans un programme Java
  - Indépendament du type de la base utilisée (mySQL, Oracle, Postgres ...)
    - Seule la phase de connexion au SGBDR change
  - Permet de faire tout type de requêtes
    - Sélection de données dans des tables
    - Création de tables et insertion d'éléments dans les tables
    - Gestion des transactions
- Packages : `java.sql` et `javax.sql`

Principes généraux d'accès à une BDD

- Première étape
  - Préciser le type de driver que l'on veut utiliser
    - Driver permet de gérer l'accès à un type particulier de SGBD
- Deuxième étape
  - Récupérer un objet « Connection » en s'identifiant auprès du SGBD et en précisant la base utilisée
- Etapes suivantes
  - A partir de la connexion, créer un « statement » (état) correspondant à une requête particulière
  - Exécuter ce statement au niveau du SGBD
  - Fermer le statement
- Dernière étape
  - Se déconnecter de la base en fermant la connexion

Connexion au SGBD

- Classe `java.sql.DriverManager`
  - Gestion du contrôle et de la connexion au SGBD
- Méthodes principales
  - `static void registerDriver(Driver driver)`
    - Enregistre le driver (objet `driver`) pour un type de SGBD particulier
    - Le driver est dépendant du SGBD utilisé
  - `static Connection getConnection(String url, String user, String password)`
    - Crée une connexion permettant d'utiliser une base
    - `url` : identification de la base considérée sur le SGBD
      - Format de l'URL est dépendant du SGGB utilisé
    - `user` : nom de l'utilisateur qui se connecte à la base
    - `password` : mot de passe de l'utilisateur

Gestion des connexions

- Interface `java.sql.Connection`
- Préparation de l'exécution d'instructions sur la base, 2 types
  - Instruction simple : classe `Statement`
    - On exécute directement et une fois l'action sur la base
  - Instruction paramétrée : classe `PreparedStatement`
    - L'instruction est générique, des champs sont non remplis
    - Permet une pré-compilation de l'instruction optimisant les performances
    - Pour chaque exécution, on précise les champs manquants
  - Pour ces 2 instructions, 2 types d'ordres possibles
    - Update : mise à jour du contenu de la base
    - Query : consulation (avec un select) des données de la base

Gestion des connexions

- Méthodes principales de `Connection`
  - `Statement createStatement()`
    - Retourne un état permettant de réaliser une instruction simple
  - `PreparedStatement prepareStatement(String ordre)`
    - Retourne un état permettant de réaliser une instruction paramétrée et pré-compilée pour un ordre `ordre`
    - Dans l'ordre, les champs libres (au nombre quelconque) sont précisés par des « ? »
      - Ex : `''select nom from clients where ville=?''`
      - Lors de l'exécution de l'ordre, on précisera la valeur du champ
  - `void close()`
    - Ferme la connexion avec le SGBD

Instruction simple

- Classe `Statement`
  - `ResultSet executeQuery(String ordre)`
    - Exécute un `ordre` de type SELECT sur la base
    - Retourne un objet de type `ResultSet` contenant tous les résultats de la requête
  - `int executeUpdate(String ordre)`
    - Exécute un `ordre` de type INSERT, UPDATE, ou DELETE
  - `void close()`
    - Ferme l'état

Instruction paramétrée

- Classe `PreparedStatement`
  - Avant d'exécuter l'ordre, on remplit les champs avec
    - `void set[Type](int index, [Type] val)`
      - Remplit le champ en i^{ème} position définie par `index` avec la valeur `val` de type `[Type]`
      - `[Type]` peut être : String, int, float, long ...
      - Ex : `void setString(int index, String val)`
    - `ResultSet executeQuery()`
      - Exécute un `ordre` de type SELECT sur la base
      - Retourne un objet de type `ResultSet` contenant tous les résultats de la requête
    - `int executeUpdate()`
      - Exécute un `ordre` de type INSERT, UPDATE, ou DELETE

Lecture des résultats

- Classe `ResultSet`
  - Contient les résultats d'une requête SELECT
    - Plusieurs lignes contenant plusieurs colonnes
    - On y accède ligne par ligne puis valeur par valeur dans la ligne
  - Changements de ligne
    - `boolean next()`
      - Se place à la ligne suivante s'il y en a une
      - Retourne `true` si le déplacement a été fait, `false` s'il n'y avait pas d'autre ligne
    - `boolean previous()`
      - Se place à la ligne précédente s'il y en a une
      - Retourne `true` si le déplacement a été fait, `false` s'il n'y avait pas de ligne précédente
    - `boolean absolute(int index)`
      - Se place à la ligne numérotée `index`
      - Retourne `true` si le déplacement a été fait, `false` sinon

Lecture des résultats

- Classe `ResultSet`
  - Accès aux colonnes/données dans une ligne
  - `[type] get[Type](int col)`
    - Retourne le contenu de la colonne `col` dont l'élément est de type `[type]` avec `[type]` pouvant être String, int, float, boolean ...
      - Ex : `String getString(int col)`
  - Fermeture du `ResultSet`
    - `void close()`

Exception SQLException

- Toutes les méthodes présentées précédemment peuvent lever l'exception `SQLException`
  - Exception générique lors d'un problème d'accès à la base lors de la connexion, d'une requête ...
    - Plusieurs spécialisations sont définies (voir API)
  - Opérations possibles sur cette exception
    - `int getErrorCode()` : le code de l'erreur renvoyé par le SGBD (et dépendant du type du SGBD)
    - `SQLException getNextException()` : si plusieurs exceptions sont chaînées entre elles, retourne la suivante ou `null` s'il n'y en a pas
    - `String getSQLState()` : retourne « l'état SQL » associé à l'exception

Exemple

- Accès à une base Oracle contenant 2 tables
  - categorie (codecat, libellecat)
    produit (codprod, nomprod, codecat*)
  - Source de l'exemple : A. Lacayrelle
- Paramètres de la base
  - Fonctionne sur la machine `ladybird` sur le port 1521
  - Base s'appelle « test »
  - Utilisateur qui se connecte : « étudiant », mot de passe : « mdpetud »

Exemple

- Création de la connexion à la base
  - `Connection con;`

```
// chargement du driver Oracle
DriverManager.registerDriver(
    new oracle.jdbc.driver.OracleDriver());

// création de la connexion
con = DriverManager.getConnection(
    ''jdbc :oracle :thin :@ladybird :1521
    :test, ''etudiant'', ''mdpetud'');

//note: la syntaxe du premier argument dépend du type
// du SGBD
```

Exemple

- Exécution d'une instruction simple de type SELECT
  - Lister toutes les caractéristiques de toutes les catégories
  - ```
    Statement req;
    ResultSet res;
    String libelle;
    int code;

    req = con.createStatement();

    res = req.executeQuery(
       ''select codcat, libellecat from categorie'');

    while(res.next()) {
        code = getInt(1);
        libelle = getString(2);
        System.out.println(
                '' produit : ''+code +'',''+ libelle);
    }
    req.close();
    ```
    ```

Exemple

- Exécution d'une instruction simple de type UPDATE
  - Ajouter une catégorie « céréales » de code 5 dans la table catégories
  - ```
    Statement req;
    int nb;

    req = con.createStatement();

    nb = req.executeUpdate(''
     insert into categories values (5, 'cereales')'');

    System.out.println(
         '' nombre de lignes modifiées : ''+nb);

    req.close();
    ```
    ```

Exemple

- Instruction paramétrée de type SELECT
  - Retourne tous les produits de la catégorie céréales
  - ```
    PreparedStatement req;
    ResultSet res;
    String nom;
    int code;

    req = con.prepareStatement(''select codprod, nomprod
       from categorie c, produit p where c.codcat=p.codcat
       and libellecat = ?'');

    req.setString(1, ''cereales'');

    res = req.executeQuery();

    while(res.next()) {
         code = getInt(1);
         libelle = getString(2);
         System.out.println(
                  '' produit : ''+code +'',''+ libelle); }
    req.close();
    ```
    ```

Exemple

- Instruction paramétrée de type UPDATE
  - Ajout de 2 nouvelles catégories dans la table catégorie
  -
```
PreparedStatement req;
int nb;

req = con.prepareStatement(
        ''insert into categories values (?,?)'');

req.setInt(1, 12);
req.setString(2, ''fruits'');
nb = req.executeUpdate();

req.setInt(1, 13);
req.setString(2, ''légumes'');
nb = req.executeUpdate();

req.close();
```

Transaction

- Fonctionnalité avancée
  - Gestion des transactions
  - Transaction
    - Un ensemble d'action est effectué en entier ou pas du tout
  - Voir documentation spécialisée