REINVENT ENGINEERING

CENTRALE NANTES

Qualité, conception, modélisation

Accéder aux données - Java

JY Martin

Novembre 2023

Plan

JDBC

Java DataBase Connectivity

LE mécanisme d'accès aux bases de données

- S'inspire du fonctionnement d'ODBC
- Driver indépendant du système d'exploitation (basé sur java)
- Nécessite l'installation au préalable du driver JDBC adéquat

JDBC

API de connexion aux bases de données

1 driver pour chaque type de base de données

Les drivers sont classés en 4 types

- Type 1 : passerelles JBDC - OBDC
- Type 2 : API natif (connexion directe à la base)
- Type 3 : conversion des appels JDBC en un protocole indépendant du SGBD
- Type 4 : conversion des appels JDBC en un protocole réseau exploité par le SGBD

Il est **vivement** recommandé d'utiliser le type 4.

Mise en œuvre

L'interface java.sql gère le driver JDBC est s'occupe de la partie abstraction de l'accès à la base de données.

- Importer le package java.sql.*
- Charger le driver JDBC (1 seule fois dans le programme)
- Etablir la connexion à la base de données
- Effectuer les requêtes SQL
- Libérer la connexion
- Libérer le driver (1 seule fois dans le programme)

Charger / Libérer le driver JDBC

Charger :

```
try {
        Class.forName(”myDriver.ClassName”);
}
catch (java.lang.ClassNotFoundException e) {
        Logger.getLogger(MaClasse.class.getName()).log(Level.SEVERE, null, ex);
}
```

Remarque : Le fichier JAR correspondant au driver JDBC doit être présent dans votre projet.

Charger / Libérer le driver JDBC

Libérer :

```
Driver theDriver = DriverManager.getDriver(dbURL);
DriverManager.deregisterDriver(theDriver);
```

Les types de driver

Dans le slide précédent, pour monter le driver JDBC en mémoire, on utilise la syntaxe myDriver.ClassName

La syntaxe à utiliser vous est habituellement fournie sur le site sur lequel vous pouvez télécharger le driver.

Concrètement :

- mySQL : com.mysql.jdbc.Driver
- PostgreSQL : org.postgresql.Driver
- Oracle : oracle.jdbc.driver.OracleDriver
- ...

Etablir une connexion / Se déconnecter

Attention : Le driver JDBC doit avoir été monté correctement pour que la connexion puisse s'établir.

On utilise, pour les échanges avec la base de données, un objet **Connection**.

Se connecter :

```
String dbURL= "jdbc:protocol:urlbase";
Connection connect = DriverManager.getConnection(dbURL, "myLogin",
"myPassword");
```

Etablir une connexion / Se déconnecter

protocol.urlbase est de la forme :

- jdbc:mysql://<host>/<bd_name>
- jdbc:postgresql://<host>/<bd_name>
- jdbc:oracle:oci8:@<database>
- ...

Etablir une connexion / Se déconnecter

Se déconnecter :

```
connect.close();
```

Effectuer une requête SQL

Il existe plusieurs classes pour faire des requêtes SQL. Celles utilisées couramment sont :

- Statement
- PreparedStatement

Il est **tres fortement** conseillé d'éviter la première classe qui est sensible aux injections SQL et donc n'offre pas toutes les garanties de sécurité de manipulation de la base de données.

Lors de leur création, ces objets permettent également de préciser des éléments sur ce qu'il sera possible de faire sur le résultat produit (revenir en arrière, modifier le résultat, ...).

Effectuer une requête SQL

```
String query = "...";
PreparedStatement stmt = connect.prepareStatement(query );
```

La requête peut comporter des paramètres qui seront définis par des '?'
Les valeurs des paramètres seront alors définies par les méthodes set...

Effectuer une requête SQL

Si la requête doit retourner un résultat, on utilise la méthode **executeQuery** sur le PreparedStatement.
Si elle ne retourne aucun résulktat (update, delete, ...) on utilise la méthode **executeUpdate**.

La récupération du résultat d'un executeQuery se fait via un objet **ResultSet**.
Les méthodes de ResultSet permettent alors de récupérer les différentes informations.

Effectuer une requête SQL - Les paramètres

Les requêtes définies pour un PreparedStatement peuvent contenir des paramètres.

- Chaque paramètre est défini par un ?.
- Il n'y a pas de limitation au nombre de paramètres.

Pour donner une valeur on utilisera une méthode set... (setString, setInt, setFloat, setDate, ...)
La méthode comporte 2 paramètres :

- Le premier paramètre est un entier indiquant le numéro d'apparition du paramètre dans la requête (1, 2, ...)
- le second paramètre est la valeur du paramètre.

Effectuer une requête SQL - Les paramètres

```
String query = "SELECT Personne_Id FROM Personne WHERE Personne_nom=?";
PreparedStatement stmt = connect.prepareStatement( query );
stmt.setString(1, "ALBAN");
```

- La requête composte un paramètre (le ?).
- On prépare la requête.
- On donne la valeur au 1er paramètre. C'est une chaîne de caractères qui vaut "ALBAN".

Effectuer une requête SQL - Les résultats

- Si la requête ne retourne aucun résultat, on utilise executeUpdate

```
String query = "DELETE FROM Personne WHERE Personne_ID=1";
PreparedStatement stmt = connect.prepareStatement( query );
stmt.executeUpdate();
```

- Si la requête retourne un résultat, on utilise executeQuery

```
String query = "SELECT Personne_Nom FROM Personne WHERE
Personne_ID=1";
PreparedStatement stmt = connect.prepareStatement( query );
ResultSet rs = stmt.executeQuery();
```

CENTRALE NANTES

Effectuer une requête SQL - Parcourir le résultat

executeQuery retourne un **ResultSet**.
Le ResultSet permet de parcourir les lignes du résultat.

- **next**() permet de passer à la ligne suivante. null si elle n'existe pas
- **previous**(), s'il est autorisé lors de la définition du PreparedStatement, permet de revenir à la ligne précédente.
- **get...** (getString, getInt, ...) permet de récupérer une colonne de la ligne. Le paramètre est soit le numéro de la colonne (ordre d'apparition dans la liste des colonnes remontées), soit par son nom.

Exemple

```
import java.sql.*;
public class TestSQL {
        public static void main(String[] argv) {
                try {
                    Class.forName("org.postgresql.Driver");
                    Connection connect = DriverManager.getConnection("jdbc:postgresql://localhost/test", "prweb", "prweb");
                    String query = "SELECT * FROM Personne WHERE nom=?";
                    PreparedStatement stmt = connect.prepareStatement(query );
                    stmt.setString(1, "ALBAN");
                    ResultSet res = stmt.executeQuery();
                    while (res.next()) {
                            System.out.println("Info : "+rs.getString("nom"));
                    }
                    stmt.close();
                    connect.close();
                } catch(java.lang.ClassNotFoundException e) {
                    System.err.println("ClassNotFoundException: " + e.getMessage());
                } catch(SQLException ex) {
                    System.err.println("SQLException: " + ex.getMessage());
                }
        }
}
```

Les DataSource

Certains objects comme les **DataSource** permettent de gérer des pools de connexion.

- Comme pour les connexions, il faut indiquer le driver utilisé, et des éléments tels que login, mot de passe, ...
- La connexion est demandée et rendue au pool de connexion
- Une connexion renduée n'est pas fermée, mais mise de coté. Elle n'est fermée que si elle n'est pas utilisée pendant un certain temps.
- Lors d'une demande de connexion, on commence par vérifier qu'il n'existe pas de connexion en attente, ce qui permet d'éviter de repasser des connexions.

Plan

Java & ORMs

Comme dans la plupart des langages, il existe des bibliothèques de fonctions qui permettent d'assusrer la connexion aux bases de données et le mapping entre objets et tables.

- Hibernate
- JPA

Ces outils reposent généralement sur des fichiers XML de configuration et des classes JAVA correspondant aux objets manipulés.

JPA

Java Persistence API est un outil de gestion de la persistence des données.

- Aux tables de la base de données, on associée des classes, les **Entités (Entity)**
- Les lignes d'une table correspondent à des instances des Entités.
- Faire persister un objet consiste à l'enregistrer dans la base
- Arrêter la persistence d'un objet revient à le retirer de la base.

JPA s'appuie sur un ensemble d'interfaces et des annotations pour fonctionner.

Les Frameworks

¨Pour aller plus loin, vous pouvez également vous appuyer sur des Frameworks qui vont masquer , au moins partiellement,l'utilisation d'une base de données

- Spring
- Springboot
- JSF
- Struts, Tapestry
- Play!


Success
NEXT EXIT