Java : gérer une base de données avec JDBC

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en Programmation par contrainte (IA)
Ingénieur en Génie logiciel

elmouelhi.achref@gmail.com

Plan

JDBC

Pour se connecter à une base de données

- Il nous faut un JDBC (qui varie selon le SGBD utilisé)
- JDBC : Java DataBase Connectivity
- SGBD : Système de Gestion de Bases de données

JDBC

Pour se connecter à une base de données

- Il nous faut un JDBC (qui varie selon le SGBD utilisé)
- JDBC : Java DataBase Connectivity
- SGBD : Système de Gestion de Bases de données

JDBC ?

- Une API (interface d'application) créée par Sun Microsystems
- Il permet d'accéder aux bases de données en utilisant un driver JDBC

JDBC

JDBC 5.1.46

- Aller sur `https://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-java-5.1.46.zip`
- Télécharger et Décompresser l'archive `.rar`

JDBC

JDBC 5.1.46

- Aller sur `https://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-java-5.1.46.zip`
- Télécharger et Décompresser l'archive `.rar`

JDBC 8.0.13

- Aller sur `https://dev.mysql.com/downloads/file/?id=480292`
- Télécharger et Décompresser l'archive `.zip`

JDBC

Intégrer le driver dans votre projet

- Faire un clic droit sur le nom du projet et aller dans `New > Folder`
- Renommer le répertoire `lib` puis valider
- Copier le `.jar` de l'archive décompressée dans `lib`

JDBC

Ajouter JDBC au path du projet

- Faire clic droit sur `.jar` qu'on a placé dans `lib`
- Aller dans `Build Path` et choisir `Add to Build Path`

JDBC

Ajouter JDBC au path du projet

- Faire clic droit sur `.jar` qu'on a placé dans `lib`
- Aller dans `Build Path` et choisir `Add to Build Path`

Ou aussi

- Faire clic droit sur le projet dans `Package Explorer` et aller dans Properties `Properties`
- Dans `Java Build Path`, aller dans l'onglet `Libraries`
- Cliquer sur `Add JARs`
- Indiquer le chemin du `.jar` qui se trouve dans le répertoire `lib` du projet
- Appliquer

JDBC

Ajouter JDBC au path du projet

- Faire clic droit sur `.jar` qu'on a placé dans `lib`
- Aller dans `Build Path` et choisir `Add to Build Path`

Ou aussi

- Faire clic droit sur le projet dans `Package Explorer` et aller dans Properties `Properties`
- Dans `Java Build Path`, aller dans l'onglet `Libraries`
- Cliquer sur `Add JARs`
- Indiquer le chemin du `.jar` qui se trouve dans le répertoire `lib` du projet
- Appliquer

Vérifier qu'une section `Referenced Libraries` a apparu.

JDBC

Avant de commencer, voici le script SQL qui permet de créer la base de données utilisée dans ce cours

```
create database jdbc;

use jdbc;

create table personne(
num int primary key auto_increment,
nom varchar(30),
prenom varchar(30)
);

show tables;
```

JDBC

Trois étapes

- Charger le driver JDBC (pour MySQL dans notre cas)
- Établir la connexion avec la base de données
- Créer et exécuter des requêtes SQL

JDBC

Avant de commencer

Tous les imports de ce chapitre sont de `java.sql.*;`

JDBC

Chargement du driver 5

```
try {
    Class.forName("com.mysql.jdbc.Driver");
}
catch (ClassNotFoundException e) {
    System.out.println(e.getMessage());
}
```

Chargement du driver 8

```
try {
    Class.forName("com.mysql.cj.jdbc.Driver");
}
catch (ClassNotFoundException e) {
    System.out.println(e.getMessage());
}
```

JDBC

Se connecter à la base de données

- Il faut spécifier l'URL (de la forme `jdbc:mysql://hote:port/nombd`)
  - hote : le nom de l'hôte sur lequel le serveur MySQL est installé (dans notre cas localhost ou 127.0.0.1)
  - port : le port TCP/IP utilisé par défaut par MySQL est 3306
  - nombd : le nom de la base de données MySQL
- Il faut aussi le nom d'utilisateur et son mot de passe (qui permettent de se connecter à la base de données MySQL)

JDBC

Connexion à la base

```
String url = "jdbc:mysql://localhost:3306/jdbc";
String user = "root";
String password = "";
Connection connexion = null;
try {
    connexion = DriverManager.getConnection(url, user, password);
} catch (SQLException e) {
    e.printStackTrace();
}
finally {
    if (connexion != null)
        try {
            connexion.close();
        } catch (SQLException ignore) {
            ignore.printStackTrace();
        }
}
```

JDBC

En cas de problème SSL avec JDBC 5, utilisez comme url

```
String url = "jdbc:mysql://localhost:3306/jdbc?
   autoReconnect=true&useSSL=false"
```

En cas de problème d'heure avec JDBC 8, utilisez comme url

```
String url =
"jdbc:mysql://localhost:3306/jdbc?useUnicode=true
&useJDBCCompliantTimezoneShift=true
&useLegacyDatetimeCode=false
&serverTimezone=UTC";
```

JDBC

Préparation et exécution de la requête

```
// création de la requête (statement)
Statement statement = connexion.createStatement();

// Préparation de la requête
String request = "SELECT * FROM Personne;";

// Exécution de la requête
ResultSet result = statement.executeQuery(request);
```

JDBC

Préparation et exécution de la requête

```
// création de la requête (statement)
Statement statement = connexion.createStatement();

// Préparation de la requête
String request = "SELECT * FROM Personne;";

// Exécution de la requête
ResultSet result = statement.executeQuery(request);
```

On utilise

- `executeQuery()` pour les requêtes de lecture : `select`.
- `executeUpdate()` pour les requêtes d'écriture : `insert`, `update`, `delete` et `create`.

JDBC

Récupération des données

```
while (result.next()) {
    // on indique chaque fois le nom de la colonne
       et le type
    int idPersonne = result.getInt("num");
    String nom = result.getString("nom");
    String prenom = result.getString("prenom");

    // pareil pour tous les autres attributs
    System.out.println(nom + " " + prenom );
}
```

JDBC

On peut faire aussi

```
while (result.next()) {
    // on peut indiquer aussi l'index de la colonne
       et le type
    int idPersonne = result.getInt(1);
    String nom = result.getString(2);
    String prenom = result.getString(3);

    // pareil pour tous les autres attributs
    System.out.println(nom + " " + prenom);
}
```

JDBC

Pour faire une insertion

```
Statement statement = connexion.createStatement();
String request = "INSERT INTO Personne (nom,prenom)
    VALUES ('Wick','John');";
int nbr = statement.executeUpdate(request);
if (0!=nbr)
    System.out.println("insertion réussie");
```

JDBC

Pour faire une insertion

```
Statement statement = connexion.createStatement();
String request = "INSERT INTO Personne (nom,prenom)
    VALUES ('Wick','John');";
int nbr = statement.executeUpdate(request);
if (0!=nbr)
    System.out.println("insertion réussie");
```

La méthode `executeUpdate()` retourne

- 0 en cas d'échec de la requête d'insertion, et 1 en cas de succès
- le nombre de lignes respectivement mises à jour ou supprimées

JDBC

Pour récupérer la valeur de la clé primaire auto-générée

```
Statement statement = connexion.createStatement();
String request = "INSERT INTO Personne (nom,prenom)  VALUES ('
   Wick','John');";

// on demande le renvoi des valeurs attribuées à la clé
   primaire
statement.executeUpdate(request,Statement.RETURN_GENERATED_KEYS
   );

// on parcourt les valeurs attribuées à l'ensemble de tuples
   ajoutés
ResultSet resultat = statement.getGeneratedKeys();

// on vérifie s'il contient au moins une valeur
if (resultat.next()) {
    System.out.println("Le numéro généré pour cette personne :
       " +  resultat.getInt(1));
}
```

JDBC

Pour éviter les injections SQL, il faut utiliser les requêtes préparées (pour la consultation, l'ajout, la suppression et la modification)

```
String request = "INSERT INTO Personne (nom,prenom)
   VALUES (?,?);";
PreparedStatement ps =  connexion.prepareStatement(
   request,PreparedStatement.RETURN_GENERATED_KEYS);
ps.setString(1, "Wick");
ps.setString(2, "John");
ps.executeUpdate();
ResultSet resultat = ps.getGeneratedKeys();
if (resultat.next())
    System.out.println("Le numéro généré pour cette
      personne : " +  resultat.getInt(1));
```

JDBC

Pour éviter les injections SQL, il faut utiliser les requêtes préparées (pour la consultation, l'ajout, la suppression et la modification)

```
String request = "INSERT INTO Personne (nom,prenom)
   VALUES (?,?);";
PreparedStatement ps =  connexion.prepareStatement(
   request,PreparedStatement.RETURN_GENERATED_KEYS);
ps.setString(1, "Wick");
ps.setString(2, "John");
ps.executeUpdate();
ResultSet resultat = ps.getGeneratedKeys();
if (resultat.next())
    System.out.println("Le numéro généré pour cette
      personne : " +  resultat.getInt(1));
```

Attention à l'ordre des attributs

JDBC

Les transactions

- ensemble de requête SQL
- appliquant le principe soit tout (toutes les requête SQL) soit rien
- activées par défaut avec MySQL
- pouvant être désactivées et gérées par le développeur

JDBC

Pour désactiver l'auto-commit

```
connection.setAutoCommit(false);
```

Pour valider une transaction

```
connection.commit(false);
```

Pour annuler une transaction

```
connection.rollback(false);
```

JDBC

Exemple avec les transactions

```
// désactiver l'auto-commit
connexion.setAutoCommit(false);

String request = "INSERT INTO Personne (nom,prenom)  VALUES
   (?,?);";
PreparedStatement ps =  connexion.prepareStatement(request,
   PreparedStatement.RETURN_GENERATED_KEYS);
ps.setString(1, "Wick");
ps.setString(2, "John");
ps.executeUpdate();

// valider l'insertion
connexion.commit();

ResultSet resultat = ps.getGeneratedKeys();
if (resultat.next())
    System.out.println("Le numéro généré pour cette personne :
       " +  resultat.getInt(1));
```

JDBC

Organisation du code

- Il faut mettre toutes les données (url, nomUtilisateur, motDePasse...) relatives à notre connexion dans une classe connexion
- Pour chaque table de la base de données, on crée une classe java ayant comme attributs les colonnes de cette table
- Il faut mettre tout le code correspondant à l'accès aux données (de la base de données) dans des nouvelles classes et interfaces (qui constitueront la couche DAO : Data Access Object)

JDBC

La classe `MyConnection`

```
package org.eclipse.config;
public class MyConnection {
  private static String url = "jdbc:mysql://localhost:3306/jdbc";
  private static String utilisateur = "root";
  private static String motDePasse = "";
  private static Connection connexion = null;
  private MyConnection() {
    try {
      Class.forName("com.mysql.jdbc.Driver");
      connexion = DriverManager.getConnection( url, utilisateur,
        motDePasse );
    } catch ( Exception e ) {
      e.printStackTrace();
    }
  }
  public static Connection getConnection() {
    if (connexion == null) {
      new MyConnection();
    }
    return connexion;
  }
```

JDBC

La classe `MyConnection` (suite)

```
  public static void stop() {
    if (connexion != null) {
      try {
        connexion.close();
      } catch (SQLException e) {
        e.printStackTrace();
      }
    }
  }
}
```

JDBC

La classe `Personne`

```
package org.eclipse.model;

public class Personne{

    private int num;
    private String nom;
    private String prenom;

    // + getters + setters + constructeur sans param
       ètre + constructeur avec 2 paramètres nom et
       prénom  + constructeur avec  3 paramètres

}
```

JDBC

L'interface `PersonneDao`

```
package org.eclipse.dao;

import java.util.List;

import org.eclipse.classes.Personne;

public interface PersonneDao {
        int save(Personne personne);
        void remove(Personne personne);
        void update(Personne personne);
        Personne findById(int id);
        List<Personne> getAll();
}
```

La classe `PersonneDaoImpl`

```
package org.eclipse.dao;

public class PersonneDaoImpl implements PersonneDao{
  @Override
  public int save(Personne personne) {
    Connection c = MyConnection.getConnection();
    if (c !=null) {
      try {
        PreparedStatement ps =  c.prepareStatement("insert into
           personne (nom,prenom) values (?,?); ", PreparedStatement.
           RETURN_GENERATED_KEYS);
        ps.setString(1, personne.getNom());
        ps.setString(2, personne.getPrenom());
        ps.executeUpdate();
        ResultSet resultat = ps.getGeneratedKeys();
        if (resultat.next()) {
          return resultat.getInt(1);
        }
      } catch (SQLException e) {
        e.printStackTrace();
      }
    }
    return -1;
  }
```

Exemple de la méthode `save` en utilisant les transactions

```
public class PersonneDaoImpl implements PersonneDao{
  @Override
  public int save(Personne personne) {
    Connection c = MyConnection.getConnection();
    if (c !=null) {
      try {
        c.setAutoCommit(false);
        PreparedStatement ps =  c.prepareStatement("insert into
           personne (nom,prenom) values (?,?); ", PreparedStatement.
           RETURN_GENERATED_KEYS);
        ps.setString(1, personne.getNom());
        ps.setString(2, personne.getPrenom());
        ps.executeUpdate();
        ResultSet resultat = ps.getGeneratedKeys();
        if (resultat.next()) {
          c.commit();
          return resultat.getInt(1);
        }
      } catch (SQLException e) {
        e.printStackTrace();
      }
    }
    return -1;
  }
```

JDBC

La classe `PersonneDaoImpl` (suite)

```
    @Override
    public Personne findById(int id) {
    Connection c = MyConnection.getConnection();
    if (c !=null) {
      try {
        PreparedStatement ps =  c.prepareStatement("select * from
           personne where num = ?; ");
        ps.setInt(1, id);
        ResultSet r =ps.executeQuery();
        r.next();
        Personne p = new Personne(r.getInt("num"),r.getString("nom"),r.
           getString("prenom"));
        return p;
      } catch (SQLException e) {
        e.printStackTrace();
        return null;
      }
    }
    return null;
  }
}
```

Il faut implémenter toutes les méthodes de l'interface `PersonneDao`

JDBC

Le `Main` pour tester toutes ces classes

```
package org.eclipse.classes;

import org.eclipse.dao.PersonneDaoImpl;
import org.eclipse.model.Personne;

public class Main {

  public static void main(String args []) {
    PersonneDaoImpl personneDaoImpl = new PersonneDaoImpl();
    Personne personne = new Personne ("Wick","John");
    int cle = personneDaoImpl.save(personne);
    if (cle != -1)
      System.out.println("personne numéro " + cle + " a été ins
        érée");
    else
      System.out.println("problème d'insertion");
  }
}
```

JDBC

Remarque

N'oublions pas d'implémenter les quatre autres méthodes du DAO

JDBC

Utilisation de la généricité avec les DAO

- Nous devons créer autant d'interfaces DAO que tables de la bases de données
- Pour éviter cela, on peut utiliser une seule interface `Dao` avec un type générique que toutes les classes d'accès aux données doivent l'implémenter.

JDBC

L'interface `Dao`

```
package org.eclipse.dao;

import java.util.List;

public interface Dao <T> {
        void save(T obj);
        void remove(T obj);
        void update(T obj);
        T findById(int id);
        List<T> getAll();

}
```

JDBC

La classe `PersonneDaoImpl`

```
package org.eclipse.dao;

public class PersonneDaoImpl implements Dao<Personne>{

...
```

Rien ne change pour le reste

JDBC

Encore de la restructuration du code

- Mettre les données (url, nomUtilisateur, motDePasse...) relatives à notre connexion dans un fichier de propriétés que nous appelons `db.properties` (utilisé par certain framework comme Spring)
- Créer une nouvelle classe (`DataSourceFactory`) qui va lire et construire les différentes propriétés de la connexion
- Utiliser `DataSourceFactory` dans `MyConnection`

JDBC

Le fichier `db.properties` situé à la racine du projet (ayant la forme `clé = valeur`, le nom de la clé est à choisir par l'utilisateur)

```
url=jdbc:mysql://localhost:3306/jdbc
username=root
password=root
```

Créons la classe `MyDataSourceFactory` dans `org.eclipse.config`

```
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;
import javax.sql.DataSource;
import com.mysql.jdbc.jdbc2.optional.MysqlDataSource;

public class MyDataSourceFactory {
  public static DataSource getMySQLDataSource() {
    Properties props = new Properties();
    FileInputStream fis = null;
    MysqlDataSource mysqlDataSource = null;
    try {
      fis = new FileInputStream("db.properties");
      props.load(fis);
      mysqlDataSource = new MysqlDataSource();
      mysqlDataSource.setURL(props.getProperty("url"));
      mysqlDataSource.setUser(props.getProperty("username"));
      mysqlDataSource.setPassword(props.getProperty("password"));
    } catch (IOException e) {
      e.printStackTrace();
    }
    return mysqlDataSource;
  }
}
```

JDBC

Remarque

Dans `MyDataSourceFactory`, on ne précise pas le driver `com.mysql.jdbc.Driver` car on utilise un objet de la classe `MysqlDataSource` qui charge lui même le driver.

La classe `MyConnection` du package `org.eclipse.config`

```
import java.sql.Connection;
import java.sql.SQLException;
import javax.sql.DataSource;

public class MyConnection {

  private static Connection connexion = null;

  private MyConnection() {

    DataSource dataSource = MyDataSourceFactory.getMySQLDataSource();
    try {
      connexion = dataSource.getConnection();
    } catch (SQLException e) {
      e.printStackTrace();
    }
  }
  public static Connection getConnection() {
    if (connexion == null) {
      new MyConnection();
    }
    return connexion;
  }
```

JDBC

Relançons le `Main` et vérifier que tout fonctionne correctement

```
package org.eclipse.classes;

import org.eclipse.dao.PersonneDaoImpl;
import org.eclipse.model.Personne;

public class Main {

  public static void main(String args []) {
    PersonneDaoImpl personneDaoImpl = new PersonneDaoImpl();
    Personne personne = new Personne ("Wick","John");
    int cle = personneDaoImpl.save(personne);
    if (cle != -1)
      System.out.println("personne numéro " + cle + " a été ins
        érée");
    else
      System.out.println("problème d'insertion");
  }
}
```