Data tier and JDBC

AMT
4. JDBC and JPA

Résumé du document
Definition

Table des matières

1. Résumé de l'API JDBC et des pilotes

L'API JDBC est une interface Java qui permet aux applications Java d'interagir avec une base de données. Un pilote JDBC est une implémentation de cette API pour un SGBD spécifique. C'est une API de bas niveau qui demande beaucoup de code répétitif. Comprendre JDBC est essentiel en cas de problèmes de performance.

1.1. Connexion JDBC

Une connexion JDBC est une session avec une base de données qui permet d'exécuter des requêtes SQL et de récupérer les résultats. Le pilote JDBC est chargé automatiquement si le fichier `.jar` du pilote est sur le classpath.

Exemple de connexion à une base PostgreSQL :

```
Connection connection = DriverManager.getConnection(
    "jdbc:postgresql://localhost:5432/postgres",
    "postgres",
    "postgres"
);
Statement statement = connection.createStatement();
ResultSet resultSet = statement.executeQuery("SELECT 1");
while (resultSet.next()) {
    System.out.println(resultSet.getInt(1));
}
```

2. Source de données JDBC

Une source de données JDBC est une fabrique de connexions JDBC. Elle est recommandée pour se connecter à une base de données.

Exemple de création d'une source de données PostgreSQL :

```
PGSimpleDataSource dataSource = new PGSimpleDataSource();
dataSource.setServerName("localhost");
dataSource.setPortNumber(5432);
dataSource.setDatabaseName("postgres");
dataSource.setUser("postgres");
dataSource.setPassword("postgres");
```

3. Gestion des ressources

Le nombre de connexions étant limité, il est important de libérer les connexions inutilisées. Les classes Connection, Statement et ResultSet implémentent l'interface AutoCloseable, ce qui permet d'utiliser l'instruction try-with-resources.

```
try (Connection connection = dataSource.getConnection();
     Statement statement = connection.createStatement();
     ResultSet resultSet = statement.executeQuery("SELECT 1")) {

    while (resultSet.next()) {
        System.out.println(resultSet.getInt(1));
    }
} catch (SQLException e) {
    // Gestion de l'erreur
}
```

4. Pool de connexions

Le pool de connexions permet de réutiliser des connexions existantes pour éviter les coûts d'ouverture/fermeture fréquents. HikariCP et Apache DBCP sont des bibliothèques populaires pour la gestion de pool de connexions. Voici un exemple de configuration HikariCP :

```
HikariConfig config = new HikariConfig();
config.setJdbcUrl("jdbc:postgresql://localhost:5432/postgres");
config.setUsername("postgres");
config.setPassword("postgres");
config.setMaximumPoolSize(10);

DataSource dataSource = new HikariDataSource(config);
```

Bien que le pool de connexions optimise la gestion des connexions, il ne résout pas tous les problèmes, et dans certains cas, cela peut entraîner des blocages ou des "deadlocks".