L'utilisation de frameworks pour le développement avec Java EE

Module 5 – Spring JDBC

Objectifs

- Comprendre le principe des DataSources
- Savoir utiliser le JdbcTemplate pour des requêtes simples
- Savoir utiliser le BeanPropertyRowMapper afin de travailler sur des objets
- Comprendre la gestion des exceptions
- Savoir injecter des DAO avec @Repository

Spring JDBC

Intérêt

- Simplification de la communication entre l'application et la base de données
- Abstraction des exceptions techniques
- Intégration de la partie JDBC dans Spring

Spring JDBC

Déclaration d'une DataSource

- Qu'est-ce qu'une DataSource ?
  - Fabrique de connexions vers une base de données
  - Généralement déclarée dans un fichier de configuration
  - Propose
    - Des instances de connexion "normales"
    - Des instances de connexion gérées par un pool de connexions

Déclaration d'une DataSource

- La Data Source est gérée par Spring
- Spring JDBC va être responsable de l'ouverture et de la fermeture des connexions JDBC
- En cas d'exception, Spring se charge de fermer correctement la connexion

Déclaration d'une DataSource

- Spring fournit deux DataSources intégrées
  - DriverManagerDataSource
    - Renvoie une nouvelle connexion à chaque demande de l'application
  - SingleConnectionDataSource
    - Crée une seule connexion et renvoie toujours la même connexion pour toute demande

Spring JDBC

Déclaration d'une DataSource

- DriverManagerDataSource
  - Fichier ApplicationContext.xml

```xml
<bean id="datasource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">

    <property name="driverClassName" value="com.mysql.jdbc.Driver" />
    <property name="url" value="jdbc:mysql://localhost:3306/javaavance" />
    <property name="username" value="java" />
    <property name="password" value="avance" />
</bean>
```

Spring JDBC

Déclaration d'une DataSource

- Test du DriverManagerDataSource

```
ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("ApplicationContext.xml");

DataSource ds = context.getBean("datasource", DataSource.class);

for (int i = 0; i < 5; i++) {
    Connection connection = ds.getConnection();
    System.out.println("Connexion numero " + (i+1) + " : " + connection);
    connection.close();
}

context.close();
```

```
Connexion numero 1 : com.mysql.jdbc.JDBC4Connection@631330c
Connexion numero 2 : com.mysql.jdbc.JDBC4Connection@fad74ee
Connexion numero 3 : com.mysql.jdbc.JDBC4Connection@436e852b
Connexion numero 4 : com.mysql.jdbc.JDBC4Connection@9a7504c
Connexion numero 5 : com.mysql.jdbc.JDBC4Connection@6b57696f
```

VOIR DEMO

Spring JDBC

Déclaration d'une DataSource

- SingleConnectionDataSource
  - Fichier ApplicationContext.xml

```
<bean id="datasource" class="org.springframework.jdbc.datasource.SingleConnectionDataSource">

    <property name="driverClassName" value="com.mysql.jdbc.Driver" />
    <property name="url" value="jdbc:mysql://localhost:3306/javaavance" />
    <property name="username" value="java" />
    <property name="password" value="avance" />
</bean>
```

VOIR DEMO

Spring JDBC

Déclaration d'une DataSource

- Test du SingleConnectionDataSource

```
ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("ApplicationContext.xml");

DataSource ds = context.getBean("datasource", DataSource.class);

for (int i = 0; i < 5; i++) {
    Connection connection = ds.getConnection();
    System.out.println("Connexion numero " + (i+1) + " : " + connection);
    connection.close();
}

context.close();
```

```
Connexion numero 1 : com.mysql.jdbc.JDBC4Connection@1a1d6a08
Exception in thread "main" java.sql.SQLException: Connection was closed in
SingleConnectionDataSource. Check that user code checks shouldClose() before
closing Connections, or set 'suppressClose' to 'true'
```

VOIR DEMO

Spring JDBC

Déclaration d'une DataSource

- Test du SingleConnectionDataSource

```
ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("ApplicationContext.xml");

DataSource ds = context.getBean("datasource", DataSource.class);

for (int i = 0; i < 5; i++) {
    Connection connection = ds.getConnection();
    System.out.println("Connexion numero " + (i+1) + " : " + connection);
}

ds.getConnection().close();
context.close();
```

```
Connexion numero 1 : com.mysql.jdbc.JDBC4Connection@1a1d6a08
Connexion numero 2 : com.mysql.jdbc.JDBC4Connection@1a1d6a08
Connexion numero 3 : com.mysql.jdbc.JDBC4Connection@1a1d6a08
Connexion numero 4 : com.mysql.jdbc.JDBC4Connection@1a1d6a08
Connexion numero 5 : com.mysql.jdbc.JDBC4Connection@1a1d6a08
```

VOIR DEMO

Spring JDBC

Déclaration d'une DataSource

- BasicDataSource
  - Proposée par le module DBCP de la bibliothèque Apache Commons
  - Gestion d'un pool de connexions

  - Fichier ApplicationContext.xml

```
<bean id="datasource" class="org.apache.commons.dbcp2.BasicDataSource"
        destroy-method="close">

    <property name="driverClassName" value="com.mysql.jdbc.Driver" />
    <property name="url" value="jdbc:mysql://localhost:3306/javaavance" />
    <property name="username" value="java" />
    <property name="password" value="avance" />

    <property name="initialSize" value="2" />
    <property name="maxTotal" value="8" />

</bean>
```

VOIR DEMO

Spring JDBC

Déclaration des DataSources

Démonstration

Spring JDBC

JdbcTemplate

- JdbcTemplate est la classe principale de Spring JDBC
- Elle est thread-safe
- Permet le requêtage SQL

Spring JDBC

Utilisation de JdbcTemplate

- Création du JdbcTemplate

```
ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("ApplicationContext.xml");

DataSource ds = context.getBean("datasource", DataSource.class);
JdbcTemplate jt = new JdbcTemplate(ds);

// [...]

context.close();
```

Spring JDBC

Utilisation de JdbcTemplate

- Recherche d'une liste de valeurs
  - Requête simple

```
String sql = "Select p.nom from SpringJdbcPersonnes p";
List<String> list = jt.queryForList(sql, String.class);
```

  - Requête avec paramètres

```
String sql = "Select p.prenom from SpringJdbcPersonnes p Where p.prenom Like ?";
List<String> list = jt.queryForList(sql, String.class, "%"+chaine+"%");
return list;
```

VOIR DEMO

Spring JDBC

Utilisation de JdbcTemplate

- Recherche d'une valeur simple

```
String sql = "Select count(p.id) from SpringJdbcPersonnes p";
int nbElements =  jt.queryForObject(sql, Integer.class);
```

```
String sql = "Select count(p.id) from SpringJdbcPersonnes p where p.age > ? And p.age < ?";
int nbElements =  jt.queryForObject(sql, Integer.class, age1, age2);
```

```
String sql = "Select count(p.id) from SpringJdbcPersonnes p where p.age > ? And p.age < ?";
int nbElements =  jt.queryForObject(sql, new Object[] {age1, age2}, Integer.class);
```

Spring JDBC

Utilisation de JdbcTemplate

- Utilisation d'un BeanPropertyRowMapper
  - Permet de mapper un objet Java avec une ligne retournée par une requête SQL

```
public class Personne {

    private int id;
    private String nom;
    private String prenom;
    private int age;

    public Personne() {
    }

    // [...]
```

```
String sql = "select id, prenom, nom, age from SpringJdbcPersonnes";
return jt.query(sql, new BeanPropertyRowMapper(Personne.class));
```

```
String sql = "select id, prenom, nom, age from SpringJdbcPersonnes where age > ? And age <
return jt.query(sql, new Object[] {age1, age2}, new BeanPropertyRowMapper(Personne.class))
```

VOIR DEMO

Spring JDBC

Utilisation de JdbcTemplate

- Utilisation d'un BeanPropertyRowMapper
  - Permet de mapper un objet Java avec une ligne retournée par une requête SQL

```
public class Individu {

    private int pk;
    private String lastname;
    private String firstname;
    private int newAge;


    public Individu() {
    }

    // [...]
```

```
String sql = "select id as pk, prenom as firstname, nom as lastname, age as newAge from SpringJdbcPersonnes";
return jt.query(sql, new BeanPropertyRowMapper(Individu.class));
```

VOIR DEMO

Spring JDBC

Utilisation de JdbcTemplate

- delete, insert, update

```
String req = "delete from SpringJdbcPersonnes where id = ?";
jt.update(req, id);
```

```
String req = "insert into SpringJdbcPersonnes (nom, prenom, age) values (?,?,?)";
jt.update(req, p.getNom(), p.getPrenom(), p.getAge());
```

```
String req = "update SpringJdbcPersonnes set nom=?, prenom=?, age=? where id=?";
jt.update(req, p.getNom(), p.getPrenom(), p.getAge(), p.getId());
```

Spring JDBC

Utilisation de NamedParameterJdbcTemplate

- Utilisation de paramètres nommés

```
String sql = "select id, prenom, nom, age from SpringJdbcPersonnes where age > :ageMin And age < :ageMax";

NamedParameterJdbcTemplate npjt = new NamedParameterJdbcTemplate(ds);

MapSqlParameterSource vParams = new MapSqlParameterSource();
vParams.addValue("ageMin", 35);
vParams.addValue("ageMax", 50);

List<Personne> res =  npjt.query(sql, vParams, new BeanPropertyRowMapper(Personne.class));
```

VOIR DEMO

Les exceptions

- Les exceptions provenant des DAO (erreur de requêtes, problèmes liés à une base spécifique…) devant être indépendantes, Spring propose deux choses :
  - Une hiérarchie de classes d'exception d'accès aux données
  - L'encapsulation automatique de toute exception technique dans une des classes de cette hiérarchie

Spring JDBC

Les exceptions

```
String sql = "Select p.nom from SpringJdbcPersonnes p2";
List<String> list = null;
try {
    list = jt.queryForList(sql, String.class);
} catch (BadSqlGrammarException e) {
    System.err.println("Message initial : " + e.getMessage());
    System.err.println("Root Cause : " + e.getRootCause().getClass());
    System.err.println("ErrorCode = " + e.getSQLException().getErrorCode());
    System.err.println("Message = " + e.getSQLException().getMessage());
    System.err.println("SQLState = " + e.getSQLException().getSQLState());
}
return list;
```

```
Message initial : StatementCallback; bad SQL grammar [Select p.nom from SpringJdbcPersonnes p2]; nested
exception is com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: Unknown column 'p.nom' in 'field list'

Root Cause : class com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException
ErrorCode = 1054
Message = Unknown column 'p.nom' in 'field list'
SQLState = 42S22
```

Spring JDBC

Utilisation de JdbcTemplate

Démonstration

L'annotation Spring @Repository

- Equivaut à l'annotation @Component
- Spécialisée pour la couche persistance
- Permet l'injection d'un DAO

Spring JDBC

L'annotation Spring @Repository

- Fichier de configuration Spring

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-4.0.xsd">

    <context:annotation-config />
    <context:component-scan base-package="com.formation.dao" />

    <bean id="datasource" class="org.apache.commons.dbcp2.BasicDataSource"
            destroy-method="close">

        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="url" value="jdbc:mysql://localhost:3306/javaavance"
        <property name="username" value="java" />
        <property name="password" value="avance" />
    </bean>

    <bean class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource"  ref="datasource"/>
    </bean>
</beans>
```

VOIR DEMO

Spring JDBC

L'annotation Spring @Repository

- Création de l'interface et de la classe DAO

```java
public interface PersonneDAO {

    public void addPersonne(Personne p);
    public void deletePersonne(Personne p);
    public void updatePersonne(Personne p);
    public Personne getPersonne(int id);
    public List<Personne> getPersonnes();
}
```

```java
@Repository(value="personneDAO")
public class PersonneDAOImpl implements PersonneDAO {

    @Autowired
    JdbcTemplate jt;

    @Override
    public void addPersonne(Personne p) {
        String req = "insert into SpringJdbcPersonnes (nom, prenom, age) values (?,?,?)";
        jt.update(req, p.getNom(), p.getPrenom(), p.getAge());
    }

    // [...]
```

VOIR DEMO

Spring JDBC

L'annotation Spring @Repository

- Test

```
ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("ApplicationContext.xml");

PersonneDAO dao = context.getBean("personneDAO", PersonneDAO.class);

System.out.println("Liste des personnes en base :");
List<Personne> listeP = dao.getPersonnes();

for (Personne personne : listeP) {
    System.out.println(personne);
}

context.close();
```

VOIR DEMO

Spring JDBC

L'annotation Spring @Repository

Démonstration

Conclusion

- Simplification du développement d'applications utilisant directement du SQL
- Gestion des DataSources par le conteneur
- Adapté aux petites applications

TP-05-GestionTache