Spring ORM

Spring ORM

Intérêt

- Deux principaux problèmes liés à l'utilisation de JPA
  - Création et configuration de l’EntityManagerFactory
  - Gestion du cycle de vie des EntityManager
- Spring permet d'intégrer JPA
  - Gestion des composants d'accès aux données par Spring
  - Uniformisation des exceptions

Spring ORM

Configuration

- Externalisation des paramètres

```
<bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">

    <property name="location" value="classpath:datasource.properties" />

</bean>
```

```
# fichier datasource.properties

driver=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost:3306/javaavance
user=java
pass=avance
```

Spring ORM

Configuration

- Déclaration de la DataSource

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

    <property name="driverClassName" value="${driver}" />
    <property name="url" value="${url}" />
    <property name="username" value="${user}" />
    <property name="password" value="${pass}" />
</bean>
```

Spring ORM

Configuration

- Déclaration de l'EntityManagerFactory

```
<bean id="emf" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">

    <property name="packagesToScan">
        <list>
            <value>fr.eni.spring.orm.bean</value>
        </list>
    </property>

    <property name="dataSource" ref="datasource"/>

    <!--  [...] -->
```

```
    <!--  [...] -->

    <property name="jpaVendorAdapter">
        <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
            <property name="showSql" value="false"/>
            <property name="database" value="MYSQL"/>
        </bean>
    </property>

    <!--
    <property name="jpaProperties">
        <props>
            <prop key="hibernate.hbm2ddl.auto">create</prop>
            create | validate | update | create-drop
        </props>
    </property>
    -->

</bean>
```

Spring ORM

Configuration

- Déclaration du transactionManager

```
<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory" ref="emf"/>
</bean>
```

- Permet la déclaration par annotation

```
<tx:annotation-driven transaction-manager="transactionManager"/>
```

La classe DAO

- Injection directe d'une instance d'un EntityManager

  (via le bean *org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean)*

```
@Repository
public class PersonneDAOImpl implements PersonneDAO {

    @PersistenceContext
    EntityManager em;
```

Spring ORM

La classe DAO

- Gestion des transactions
  - L'annotation @Transactional est utilisée pour indiquer au conteneur les méthodes qui doivent s'exécuter dans un contexte transactionnel
  - L'annotation @Transactional s'utilise sur une classe ou une méthode
  - L'annotation @Transactional sur une classe s'applique sur toutes les méthodes publiques de la classe

Spring ORM

La classe DAO

- Ajout, modification, suppression d'un Entity

```
@Transactional
public void add(Personne p) {
    em.persist(p);
}
```

```
@Transactional
public void update(Personne p) {
    em.merge(p);
}
```

```
@Transactional
public void delete(int id) {
    Personne p = findById(id);
    if (p != null)
        em.remove(p);
}
```

```
@Transactional
public void delete(Personne p) {
    delete(p.getId());
}
```

La classe DAO

- Utilisation de la méthode executeUpdate()

```
@Transactional
public void delete(String nom) {
    em
        .createQuery("delete from Personne p where p.nom = :param")
        .setParameter("param", nom)
        .executeUpdate();
}
```

Spring ORM

La classe DAO

- Les méthodes select ne nécessitent pas @Transactional

```
public List<Personne> findAll() {
    return em
            .createQuery("select p from Personne p", Personne.class)
            .getResultList();
}
```

```
public Personne findById(int id) {
    return em.find(Personne.class, id);
}
```

```
private int getLastId(){
    return em
            .createQuery("select max(p.id) from Personne p", Integer.class)
            .getSingleResult();
}
```

Démonstration

Spring ORM

Cas de deux sources de données

```
<bean id="datasource1" class="org.apache.commons.dbcp2.BasicDataSource" destroy-method="close">..

<bean id="emf1" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
    <property name="packagesToScan">..
    <property name="dataSource" ref="datasource1"/>
    <property name="jpaVendorAdapter">..

    <property name="persistenceUnitName" value="PU1" />
</bean>
```

```
<bean id="datasource2" class="org.apache.commons.dbcp2.BasicDataSource" destroy-method="close">..

<bean id="emf2" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
    <property name="packagesToScan">..
    <property name="dataSource" ref="datasource2"/>
    <property name="jpaVendorAdapter">..

    <property name="persistenceUnitName" value="PU2" />
</bean>
```

Spring ORM

Cas de deux sources de données

- Deux transactionManager

```
<bean id="transactionManager2" class="org.springframework.orm.jpa.JpaTransactionManager">

    <property name="entityManagerFactory" ref="emf2"/>
     <qualifier value="tm2"/>
</bean>

<bean id="transactionManager1" class="org.springframework.orm.jpa.JpaTransactionManager">

    <property name="entityManagerFactory" ref="emf1"/>
    <qualifier value="tm1"/>

</bean>

<tx:annotation-driven/>
```

Spring ORM

Cas de deux sources de données

- Définir l'unité de persistance

```
@Repository
public class PersonneDAOImpl implements PersonneDAO {

    @PersistenceContext(unitName="PU1")
    EntityManager em;
```

- Définir le gestionnaire de transactions

```
@Transactional("tm1")
public void add(Personne p) {
    em.persist(p);
}
```

Démonstration

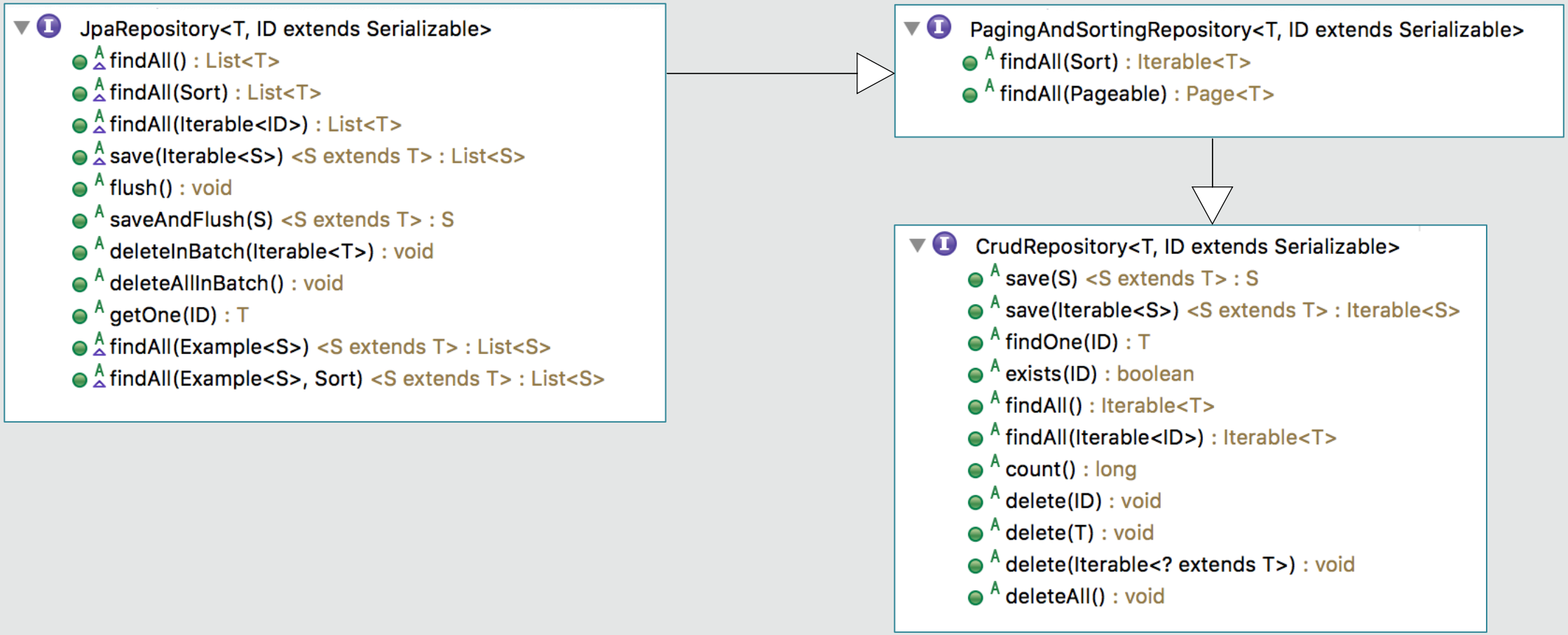
Spring ORM

Spring Data JPA

- Couche d’abstraction supplémentaire par rapport à JPA
- S'occupe de l'implémentation des fonctionnalités les plus courantes des DAO
- Spring fournit des interfaces fournissant des méthodes d'accès aux données

Spring ORM

Spring Data JPA



Spring ORM

Spring Data JPA

- Création d'une interface héritant de JpaRepository

```
import org.springframework.data.jpa.repository.JpaRepository;

public interface PersonneDAO extends JpaRepository<Personne, Integer>{

}
```

```
@Service(value="gestionPersonne")
public class GestionPersonne {

    @Autowired
    PersonneDAO pdao;

    public List<Personne> listePersonnes(){
        return pdao.findAll();
    }

    public Personne rechercherPersonne(int id){
        return pdao.findOne(id);
    }
    public void ajouterPersonne(Personne p){
        pdao.save(p);
    }
    public void modifierPersonne(Personne p){
        pdao.save(p);
    }
    public void supprimerPersonne(Personne p){
        pdao.delete(p);
    }
    public void supprimerPersonne(int id){
        pdao.delete(id);
    }

    public long nombreDElements(){
        return pdao.count();
    }
```

Spring ORM

Spring Data JPA

- Configuration

```
<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"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:jpa="http://www.springframework.org/schema/data/jpa"

    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
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
    http://www.springframework.org/schema/data/jpa
    http://www.springframework.org/schema/data/jpa/spring-jpa-1.8.xsd"
    >

    <jpa:repositories base-package="fr.eni.spring.orm.dao"
                      entity-manager-factory-ref="emf"/>


    <!-- [...] -->
```

Spring Data JPA

- Possibilité de définir dans l'interface des méthodes de requête en utilisant des mots-clés

  (https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods)

Mot-clé	Exemple	JPQL
And	findByLastnameAndFirstname	… where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	… where x.lastname = ?1 or x.firstname = ?2
Is, Equals	findByFirstname findByFirstnameIs findByFirstnameEquals	… where x.firstname = ?1

Spring ORM

Spring Data JPA

Mot-clé	Exemple	JPQL
Between	findByStartDateBetween	… where x.startDate between ?1 and ?2
LessThan	findByAgeLessThan	… where x.age < ?1
GreaterThanEqual	findByAgeGreaterThanEqual	… where x.age >= ?1
After, Before	findByStartDateAfter	… where x.startDate > ?1
IsNull, IsNotNull, NotNull	findByAge(Is)NotNull	… where x.age not null
EndingWith	findByFirstnameEndingWith	… where x.firstname like ?1 (->%)
OrderBy	findByAgeOrderByLastnameDesc	… where x.age = ?1 order by x.lastname desc
True, False	findByActiveTrue()	… where x.active = true

Démonstration