



Spring Data

Présentation

Spring Data

2

- ✓ Présentation
- ✓ Spring Data et Maven
- ✓ Spring Data Jdbc
- ✓ Spring Data JPA
- ✓ Spring Data REST
- ✓ Spring Data MongoDB

Présentation

- Spring Data a pour objectif de vous simplifier l'accès à vos données.
- Elles peuvent provenir de :
 - JDBC
 - JPA
 - LDAP
 - MongoDB, Cassandra, Solr
 - Services REST

Présentation

4

- Spring Data va simplifier le codage des DAOs
- Soit par :
 - Héritage d'une classe qui fait déjà du CRUD
 - Utilisation d'annotations qui vont simplifier le code
 - Utilisation d'interface qui seront introspectées par le Spring afin d'en générer le code des DAOs dynamiquement
- Objectif : zéro ligne de code dans ses DAOs
 - Possible uniquement dans les cas simples
- Pleins d'exemples disponibles :
 - <https://github.com/spring-projects/spring-data-examples>

Spring Data et Maven

➤ Dépendances Maven simplifiées :

```
<dependencyManagement>
  <dependencies>
    <!-- Afin d'éviter d'indiquer tous les éléments du Spring -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-framework-bom</artifactId>
      <version>${version.spring}</version>
      <scope>import</scope>
      <type>pom</type>
    </dependency>
    <!-- Afin d'éviter les problématiques de dépendances Spring Data -->
    <!-- Ne marche pas avec Spring JDBC -->
    <dependency>
      <groupId>org.springframework.data</groupId>
      <artifactId>spring-data-releasetrain</artifactId>
      <version>${version.spring-data-releasetrain}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Le <dependencyManagement> ne vous épargnera pas de placer une dépendance dans <dependencies>

Spring Data et Maven

- En fonction de ce que vous voulez faire, vous ajouterez dans `<dependencies>` vos `groupId` et `artifactId` sans préciser les versions
 - Ce n'est pas valable dans tous les cas (entre autre JDBC)

➤ Exemple : Spring Data JPA

```
<dependencies>
  <dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-jpa</artifactId>
    <!-- Version gérée par le dependencyManagement -->
  </dependency>
```

Spring Data : JDBC

- Repose sur un module/framework indépendant du Spring
 - org.springframework.data / spring-data-jdbc-core
 - com.nurkiewicz.jdbcrepository / jdbcrepository
- Avantages :
 - Vous avez un DAO CRUD déjà codé via la classe `com.nurkiewicz.jdbcrepository.JdbcRepository<E, K>`
 - ❑ E représente la classe/interface des entités
 - ❑ K représente la classe/interface de la clef primaire associée à l'entité
 - Vos interfaces de DAOs doivent implémenter `org.springframework.data.repository.PagingAndSortingRepository<E, K>` ou `org.springframework.data.repository.CrudRepository<E, K>`
 - Vous avez un complément à coder à votre `org.springframework.jdbc.core.RowMapper` le `com.nurkiewicz.jdbcrepository.RowUnmapper`

Spring Data : JPA

➤ Dépendance sur

- `org.springframework.data / spring-data-jpa`

➤ Avantages :

- Vous pouvez vous passer des classes de DAOs (et de l'*entityManager*)
- Vos interfaces de DAOs doivent implémenter `org.springframework.data.repository.PagingAndSortingRepository<E, K>` ou `org.springframework.data.repository.CrudRepository<E, K>`
 - ❑ E représente la classe/interface des entités
 - ❑ K représente la classe/interface de la clef primaire associée à l'entité
- Vos requêtes seront auto générées
 - ❑ Via le JPQL de l'annotation `@org.springframework.data.jpa.repository.Query`
 - ❑ Via le nom de votre méthode qui devra respecter les contraintes Spring Data
- Vous pouvez appeler des procédures stockées via l'annotation `@org.springframework.data.jpa.repository.query.Procedure`

Spring Data : JPA

➤ Exemple de DAO

➡ Il n'y a aucune classe qui implémente cette Interface

```
1 package com.banque.dao;
2
3 import java.util.List;
4
5 import org.springframework.data.jpa.repository.Query;
6 import org.springframework.data.repository.PagingAndSortingRepository;
7 import org.springframework.data.repository.query.Param;
8 import org.springframework.stereotype.Repository;
9
10 import com.banque.entity.impl.CompteEntity;
11
12 * Compte DAO en utilisant Spring Data avec JPA. <br/>
13 @Repository
14 public interface ICompteDAO extends PagingAndSortingRepository<CompteEntity, Integer> {
15     * Selectionne tous les comptes qui appartiennent a un utilisateur.
16     @Query("FROM CompteEntity cpt where cpt.utilisateur.id = :aUserId order by cpt.libelle")
17     public List<CompteEntity> findAllBelongToUserId(@Param("aUserId") int aUserId);
18 }
```

Spring Data : JPA

➤ Exemple de DAO

- ➡ Il n'y a aucune classe qui implémente cette Interface
- ➡ Ici, pas de requête non plus

```
1 package com.banque.dao;
2
3 import java.util.List;
4
5 import org.springframework.data.repository.CrudRepository;
6 import org.springframework.stereotype.Repository;
7
8 import com.banque.entity.impl.CompteEntity;
9
10 @Repository
11 public interface ICompteDAO extends CrudRepository<CompteEntity, Integer> {
12
13     /**
14      * Le Spring Data va fabriquer pour vous la requête en regardant le nom de la
15      * méthode :<br/>
16      * find : select<br/>
17      * By : from<br/>
18      * Customer : nom de l'objet visé<br/>
19      */
20     public List<CompteEntity> findByCompteEntity(CompteEntity ce);
21 }
```

Spring Data : JPA

11

➤ Mots clefs supportés :

And	<code>findByLastnameAndFirstname</code>	<code>where x.lastname = ?1 and x.firstname = ?2</code>
Or	<code>findByLastnameOrFirstname</code>	<code>where x.lastname = ?1 or x.firstname = ?2</code>
Is,Equals	<code>findByFirstname,findByFirstnames,findByFirstnameEquals</code>	<code>where x.firstname = ?1</code>
Between	<code>findByStartDateBetween</code>	<code>where x.startDate between ?1 and ?2</code>
LessThan	<code>findByAgeLessThan</code>	<code>where x.age < ?1</code>
LessThanEqual	<code>findByAgeLessThanEqual</code>	<code>where x.age <= ?1</code>
GreaterThan	<code>findByAgeGreaterThan</code>	<code>where x.age > ?1</code>
GreaterThanEqual	<code>findByAgeGreaterThanEqual</code>	<code>where x.age >= ?1</code>
After	<code>findByStartDateAfter</code>	<code>where x.startDate > ?1</code>
Before	<code>findByStartDateBefore</code>	<code>where x.startDate < ?1</code>
IsNull	<code>findByAgeIsNull</code>	<code>where x.age is null</code>
IsNotNull,NotNull	<code>findByAge(Is)NotNull</code>	<code>where x.age not null</code>
Like	<code>findByFirstnameLike</code>	<code>where x.firstname like ?1</code>
NotLike	<code>findByFirstnameNotLike</code>	<code>where x.firstname not like ?1</code>
StartingWith	<code>findByFirstnameStartingWith</code>	<code>where x.firstname like ?1 (parameter bound with appended %)</code>
EndingWith	<code>findByFirstnameEndingWith</code>	<code>where x.firstname like ?1 (parameter bound with prepended %)</code>
Containing	<code>findByFirstnameContaining</code>	<code>where x.firstname like ?1 (parameter bound wrapped in %)</code>
OrderBy	<code>findByAgeOrderByLastnameDesc</code>	<code>where x.age = ?1 order by x.lastname desc</code>
Not	<code>findByLastnameNot</code>	<code>where x.lastname <> ?1</code>
In	<code>findByAgeIn(Collection<Age> ages)</code>	<code>where x.age in ?1</code>
NotIn	<code>findByAgeNotIn(Collection<Age> age)</code>	<code>where x.age not in ?1</code>
True	<code>findByActiveTrue()</code>	<code>where x.active = true</code>
False	<code>findByActiveFalse()</code>	<code>where x.active = false</code>
IgnoreCase	<code>findByFirstnameIgnoreCase</code>	<code>where UPPER(x.firstname) = UPPER(?1)</code>

Spring Data : JPA

➤ Inconvénients :

- Vos paramètres de requêtes ne peuvent pas être nulles
 - ❑ Peut entraîner un code lourd côté service métiers

➤ Une grosse documentation

- <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/>

Travaux pratiques

- Reprenez le TP sur JPA
 - ➡ Faites le passer en Spring Data avec JPA



Spring Data : REST

- Dépendance sur
 - ➡ org.springframework.data / spring-data-jpa
 - ➡ org.springframework.data / spring-data-rest-webmvc
 - ➡ Un framework JSON (Jackson)
- Dans cette approche, le Spring va vous faire un DAO qui sera en plus un Web Service REST
- Le DAO étant en Spring Data, il supportera les opérations en CRUD + vos requêtes
- Les web services seront automatiquement en **HATEOAS**

Spring Data : REST

- Hypermedia as the Engine of Application State
- De cette manière votre client REST n'a pas besoin de connaître les web services
 - Il peut les découvrir dynamiquement
 - Le serveur envoie toujours les informations dans un format JSON qui contient lui-même des URLs d'appels
- Le serveur peut changer ses web services sans impacter le client
 - A partir du moment où le client fait l'effort d'utiliser les liens (links/href)

Spring Data : REST

- Sur un projet Spring Data JPA + Spring MVC classique (avec son DispatcherServlet) il suffit simplement d'ajouter le bean `org.springframework.data.rest.webmvc.config.RepositoryRestMvcConfiguration`

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context"
4     xsi:schemaLocation="http://www.springframework.org/schema/beans
5         http://www.springframework.org/schema/beans/spring-beans.xsd
6         http://www.springframework.org/schema/context
7         http://www.springframework.org/schema/context/spring-context.xsd">
8
9
10     <!-- On declare simplement le bean -->
11     <bean class="org.springframework.data.rest.webmvc.config.RepositoryRestMvcConfiguration" />
12
13 </beans>
```


Spring Data : REST

- Sur ses DAO, l'annotation `@Repository` se transforme en `@org.springframework.data.rest.core.annotation.RepositoryRestResource`
- Le reste du DAO en Spring Data ne change pas :

```
1 package com.banque.dao;
2
3 import java.util.List;
4
5 import org.springframework.data.jpa.repository.Query;
6 import org.springframework.data.repository.PagingAndSortingRepository;
7 import org.springframework.data.repository.query.Param;
8 import org.springframework.data.rest.core.annotation.RepositoryRestResource;
9
10 import com.banque.entity.impl.CompteEntity;
11
12 * Compte DAO en utilisant Spring Data avec JPA. <br/>
13 @RepositoryRestResource(collectionResourceRel = "comptes", path = "comptes")
24 public interface ICompteDAO extends PagingAndSortingRepository<CompteEntity, Integer> {
25
26 * Selectionne tous les comptes qui appartiennent a un utilisateur.
27 @Query("FROM CompteEntity cpt where cpt.utilisateur.id = :aUserId order by cpt.libelle")
34 public List<CompteEntity> findAllBelongToUserId(@Param("aUserId") int aUserId);
35 }
```

Spring Data : REST

- Lors du déploiement, tous les DAO seront disponibles en tant que contrôleur REST
- Lancer un SOAP UI pour voir la liste des services disponibles ainsi que les éléments attendus
 - Si vous êtes en Linux ou powershell vous pouvez aussi utiliser *curl* ou *Invoke-RequestMethod*
- Remarque : par défaut, les ID de vos entités ne seront pas exposés dans le flux Json
 - Ce comportement est paramétrable via la méthode `exposeIdsFor` de `RepositoryRestConfiguration`

Spring Data : REST

The screenshot displays the SoapUI 5.3.0 interface. The main window shows a REST client configuration for a GET request to the endpoint `http://localhost:8080/exo18.spring.data.rest/`. The response is a JSON object representing a collection of resources, including links, comptes, utilisateur, and operations.

Request Configuration:

- Method: GET
- Endpoint: `http://localhost:8080`
- Resource: `/exo18.spring.data.rest/`

Response (JSON):

```
1 { "_links": {
2   "comptes": {
3     "href": "http://localhost:8080/exo18.spring.data.rest/comptes?page, size, sort",
4     "templated": true
5   },
6   "utilisateur": {
7     "href": "http://localhost:8080/exo18.spring.data.rest/utilisateur?page, size, sort",
8     "templated": true
9   },
10  "operations": {
11    "href": "http://localhost:8080/exo18.spring.data.rest/operations?page, size, sort",
12    "templated": true
13  },
14  "profile": { "href": "http://localhost:8080/exo18.spring.data.rest/profile" }
15 }
```

Request Properties Table:

Property	Value
Name	Request 1

Log Area:

response time: 197ms (562 bytes)

SoapUI log http log jetty log error log wsrm log memory log

Spring Data : REST

- Il n'y a plus vraiment de couche service ni contrôleur
 - Tout est fusionné
- Dans cet approche chaque méthode de vos DAO est **transactionnelle**
 - Géré pour vous en automatique (pas besoin de @Transactional)
 - Mais vous pouvez garder/déclarer le transactionManager
 - ❑ Il vous servira sûrement pour les tests unitaires afin de forcer le rollback
- Si vous avez besoin de faire des opérations multi-table, la solution la plus simple sera la procédure stockée
 - Votre transaction sera gérée par la base de données

Travaux pratiques



- Reprenez le TP sur Spring MVC
 - Remplacez vos DAO par ceux de l'exercice Spring Data JPA
 - Supprimez vos webs services Rest
 - Faites le passer en Spring Data Rest vos DAOs, testez avec Soap UI
- Remarque : pour la transformation des dates, vous pouvez faire usage de l'annotation `@org.springframework.format.annotation.DateTimeFormat` sur vos paramètres de DAO
 - Vous pouvez aussi faire usage de la mécanique de votre API Rest
 - ❑ Par exemple `@JsonSerialize` si vous êtes en Jackson

Spring Data : MongoDB

- Spring Data supporte les bases NoSQL comme MongoDB
- Dépendance sur
 - `org.springframework.data / spring-data-mongodb`
- Avantages :
 - Vous pouvez vous passer des classes de DAOs (et du *MongoTemplate*)
 - Vos interfaces de DAOs doivent implémenter `org.springframework.data.mongodb.repository.MongoRepository<E, K>`
 - ❑ E représente la classe/interface des entités
 - ❑ K représente la classe/interface de la clef primaire associée à l'entité
 - Vos requêtes seront auto générées
 - ❑ Via le nom de votre méthode qui devra respecter les contraintes Spring Data de MongoDB

Spring Data : MongoDB

- Dans le cas de MongoDB, vos interfaces DAOs peuvent retourner des
 - Des entités 'classiques'
 - Des `java.util.stream.Stream`
 - Des objets Asynchrones
 - ❑ `java.util.concurrent.CompletableFuture;`
 - ❑ `java.util.concurrent.Future`
 - ❑ `org.springframework.util.concurrent.ListenableFuture`

Spring Data : MongoDB

➤ Exemple d'un DAO =>

➤ Et de l'entité associée

```

1 package com.banque.entity.impl;
2
3 import org.springframework.data.annotation.Id;
4 import org.springframework.data.mongodb.core.mapping.Document;
5
6 * Le bean qui represente un utilisateur. <br>
7 @Document
8 public class UtilisateurEntity {
9
10     @Id
11     private Integer id;
12
13     private String login;
14     private String password;
15     private String nom;
16     private String prenom;
17     private Boolean sex;
18     private Long derniereConnection;
19     private Long dateDeNaissance;
20     private String adresse;
21     private String telephone;
22     private Integer codePostal;
23
24 }
```

```

1 package com.banque.dao;
2
3 import java.util.concurrent.CompletableFuture;
4 import java.util.concurrent.Future;
5 import java.util.stream.Stream;
6
7 import org.springframework.data.domain.Page;
8 import org.springframework.data.domain.Pageable;
9 import org.springframework.data.domain.Slice;
10 import org.springframework.data.mongodb.repository.MongoRepository;
11 import org.springframework.scheduling.annotation.Async;
12 import org.springframework.stereotype.Repository;
13 import org.springframework.util.concurrent.ListenableFuture;
14
15 import com.banque.entity.impl.UtilisateurEntity;
16
17 * Interface representant l'utilisateur en utilisant Spring Data avec JPA. <br>
18 @Repository
19 public interface IUtilisateurDAO extends MongoRepository<UtilisateurEntity, Integer> {
20
21     /**
22      * Selectionne le premier utilisateur ayant le login indique.
23      *
24      * @param pLogin
25      *      un login.
26      * @return l'utilisateur
27      */
28     public abstract UtilisateurEntity findByLogin(String login);
29
30     public abstract Page<UtilisateurEntity> queryFirst10ByNom(String nom, Pageable pageable);
31
32     public abstract Slice<UtilisateurEntity> findTop3ByNom(String nom, Pageable pageable);
33
34     public abstract Stream<UtilisateurEntity> readAllByPrenomNotNull();
35
36     // En Asynchrone
37     @Async
38     public abstract Future<UtilisateurEntity> findByPrenom(String prenom);
39
40     @Async
41     public abstract CompletableFuture<UtilisateurEntity> findOneByPrenom(String prenom);
42
43     @Async
44     public abstract ListenableFuture<UtilisateurEntity> findOneByNom(String nom);
45
46 }
```


Spring Data : MongoDB

➤ En MongoDB

- Il n'y a pas de clef auto incrémentée, il faut faire une séquence et la gérer dans son code Java
 - ❑ Problème potentiel dans nos DAO 'sans code'
- Il n'y a pas non plus de transactionnel au sens classique du terme
 - ❑ Pas la peine de placer un @Transactional dans votre code de service