

Spring Data REST

Achref El Mouelhi

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

`elmouelhi.achref@gmail.com`



spring

Plan

- 1 Introduction
- 2 @RepositoryRestResource
- 3 Configuration avec `application.properties`
- 4 @Param
- 5 @RestResource
- 6 Cas d'une association
- 7 RepositoryRestConfiguration
- 8 @Projection
- 9 @Value
- 10 Extrait

Spring Data REST

Rappel

- Dans une application **Spring MVC**, le contrôleur, annoté par `@RestController`, utilise le repository pour assurer le service **REST**.
- Avec **Spring Data REST**, il est possible d'omettre le contrôleur et d'annoter directement le repository pour exposer les données.

Spring Data REST

Spring Data REST supporte uniquement les projets suivants

- **Spring Data JPA**
- **Spring Data MongoDB**
- **Spring Data Neo4j**
- **Spring Data GemFire**
- **Spring Data Cassandra**

Spring Data REST

Création de projet **Spring Boot**

- Aller dans `File > New > Other`
- Chercher `Spring`, dans `Spring Boot` sélectionner `Spring Starter Project` et cliquer sur `Next >`
- Saisir
 - `cours-spring-data-rest` dans `Name`,
 - `com.example` dans `Group`,
 - `cours-spring-data-rest` dans `Artifact`,
 - `com.example.demo` dans `Package`
- Cliquer sur `Next >`
- Chercher et cocher les cases correspondantes aux `Spring Data JPA`, `MySQL Driver`, `Spring Boot DevTools`, `Lombok` et `Rest Repositories` puis cliquer sur `Next >`
- Valider en cliquant sur `Finish`

Spring Data REST

Explication

- Le package contenant le point d'entrée de notre application (la classe contenant le `public static void main`) **est** `com.example.demo`
- Tous les autres packages `dao, model...` doivent être dans le package `demo`.

© Achref EL MOU

Spring Data REST

Explication

- Le package contenant le point d'entrée de notre application (la classe contenant le `public static void main`) **est** `com.example.demo`
- Tous les autres packages `dao, model...` doivent être dans le package `demo`.

Pour la suite, nous considérons

- une entité `Personne` à définir dans `com.example.demo.model`
- une interface DAO `PersonneRepository` à définir dans `com.example.demo.dao`

Spring Boot & REST

Créons une entité `Personne` dans `com.example.demo.model`

```
package com.example.demo.model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.NonNull;
import lombok.RequiredArgsConstructor;

@Entity
@NoArgsConstructor
@AllArgsConstructor
@Data
@Entity
@RequiredArgsConstructor
public class Personne {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long num;
    @NonNull
    private String nom;
    @NonNull
    private String prenom;
}
```

Spring Boot & REST

Préparons notre interface DAO `PersonneRepository`

```
package com.example.demo.dao;

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

import com.example.demo.model.Personne;

public interface PersonneRepository extends JpaRepository<Personne,
    Long> {

}
```

Spring Data REST

Dans `application.properties`, ajoutons les données permettant la connexion à la base de données et la configuration de `Hibernate`

```
spring.datasource.url = jdbc:mysql://localhost:3306/cours_data_rest?createDatabaseIfNotExist=true
spring.datasource.username = root
spring.datasource.password = root
spring.jpa.hibernate.ddl-auto = update
spring.jpa.show-sql = true
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQLDialect
spring.jpa.hibernate.naming.physical-strategy = org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
```

L'ajout de la propriété `spring.jpa.hibernate.naming.physical-strategy` permet de forcer **Hibernate** à utiliser les mêmes noms pour les tables et les colonnes que les entités et les attributs.

Spring Data REST

Pour déclarer un repository comme étant une API REST, il faut ajouter l'annotation

@RepositoryRestResource

```
package com.example.demo.dao;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.rest.core.annotation.
    RepositoryRestResource;

import com.example.demo.model.Personne;

@RepositoryRestResource
public interface PersonneRepository extends JpaRepository<Personne,
    Long> {

}
```

Spring Data REST

Pour alimenter la base de données avec quelques données au démarrage de l'application

```
@SpringBootApplication
public class CoursSpringDataRestApplication implements ApplicationRunner {

    @Autowired
    private PersonneRepository personneRepository;

    public static void main(String[] args) {
        SpringApplication.run(CoursSpringDataRestApplication.class, args);
    }

    @Override
    public void run(ApplicationArguments args) throws Exception {
        Personne personnel1 = new Personne("wick", "john");
        Personne personne2 = new Personne("dalton", "jack");
        Personne personne3 = new Personne("maggio", "carol");
        Personne personne4 = new Personne("cohen", "sophie");
        personneRepository.saveAll(Arrays.asList(personnel1, personne2, personne3, personne4));
    }
}
```

Spring Data REST

Pour tester, il faut aller sur

- `http://localhost:8080/personnes` pour récupérer la liste des personnes
- ou sur `http://localhost:8080/personnes/1` pour consulter la personne ayant l'identifiant 1
- ou sur `http://localhost:8080/personnes?size=2` pour consulter les deux premiers tuples (la première page)
- ou sur `http://localhost:8080/personnes?size=2&page=1` pour consulter la page 2 de taille 2
- ou sur `http://localhost:8080/personnes?sort=nom` pour trier le résultat selon le nom
- ou sur `http://localhost:8080/personnes?sort=nom, desc` pour trier le résultat selon le nom dans l'ordre descendant

Spring Data REST

Pour ajouter une personne, utilisons **Postman**

- Saisissez l'URL `http://localhost:8080/personnes`
- Choisissez le verbe `POST`
- Dans `Headers`, précisez la clé `Content-Type` et la valeur `application/json`
- Dans `Body`, cochez `raw` et sélectionnez `JSON (application/json)`

© Achille

Spring Data REST

Pour ajouter une personne, utilisons **Postman**

- Saisissez l'URL `http://localhost:8080/personnes`
- Choisissez le verbe `POST`
- Dans `Headers`, précisez la clé `Content-Type` et la valeur `application/json`
- Dans `Body`, cochez `raw` et sélectionnez `JSON (application/json)`

Exemple de valeurs à persister

```
{  
  "nom": "el mouelhi",  
  "prenom": "achref"  
}
```

Spring Data REST

Définissons une base pour tous les repository exposés comme des ressources REST

```
spring.data.rest.basePath=/ws
```

© Achref EL MOUELHI ©

Spring Data REST

Définissons une base pour tous les repository exposés comme des ressources REST

```
spring.data.rest.basePath=/ws
```

Nouvelles URLs pour les tests

- `http://localhost:8080/ws/personnes` pour récupérer la liste des personnes
- ou sur `http://localhost:8080/ws/personnes/1` pour consulter la personne ayant l'identifiant 1
- ou sur `http://localhost:8080/ws/personnes?size=2` pour consulter les deux premiers tuples (la première page)
- ou sur `http://localhost:8080/ws/personnes?size=2&page=1` pour consulter la page 2 de taille 2
- ou sur `http://localhost:8080/ws/personnes?sort=nom` pour trier le résultat selon le nom
- ou sur `http://localhost:8080/ws/personnes?sort=nom, desc` pour trier le résultat selon le nom dans l'ordre descendant

Spring Data REST

Autres propriétés possibles pour **Spring Data REST**

- `returnBodyOnCreate` : spécifie si un corps doit être renvoyé lors de la création d'une entité
- `returnBodyOnUpdate` : spécifie si un corps doit être renvoyé lors de la mise à jour d'une entité
- `maxPageSize` : indique le nombre maximum d'éléments dans une seule page
- `pageParamName` : modifie le nom du paramètre de requête pour sélectionner les pages (par défaut `page`)
- `sortParamName` : modifie le nom du paramètre de requête pour trier les données (par défaut `sort`)
- ...

Spring Data REST

Pour définir une méthode personnalisée

```
package com.example.demo.dao;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.repository.query.Param;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

import com.example.demo.model.Personne;

@RepositoryRestResource
public interface PersonneRepository extends JpaRepository<Personne, Long> {
    List<Personne> findByNom(@Param("nom") String nom);
}
```

Spring Data REST

Pour définir une méthode personnalisée

```
package com.example.demo.dao;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.repository.query.Param;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

import com.example.demo.model.Personne;

@RepositoryRestResource
public interface PersonneRepository extends JpaRepository<Personne, Long> {
    List<Personne> findByNom(@Param("nom") String nom);
}
```

URL pour tester

```
http://localhost:8080/ws/personnes/search/findByNom?nom=dalton
```

Spring Data REST

On peut aussi chercher les personnes dont le nom contient une certaine sous-chaine

```
package com.example.demo.dao;

import java.util.List;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.repository.query.Param;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
import com.example.demo.model.Personne;

@RepositoryRestResource
public interface PersonneRepository extends JpaRepository<Personne, Long> {
    List<Personne> findByNomContains(@Param("nom") String nom);
}
```

© Achref

Spring Data REST

On peut aussi chercher les personnes dont le nom contient une certaine sous-chaine

```
package com.example.demo.dao;

import java.util.List;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.repository.query.Param;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
import com.example.demo.model.Personne;

@RepositoryRestResource
public interface PersonneRepository extends JpaRepository<Personne, Long> {
    List<Personne> findByNomContains(@Param("nom") String nom);
}
```

URL pour tester

```
http://localhost:8080/ws/personnes/search/findByNomContains?
nom=i
```

Spring Data REST

Pour renommer le chemin vers la méthode précédente, on utilise @RestResource

```
package com.example.demo.dao;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.repository.query.Param;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
import org.springframework.data.rest.core.annotation.RestResource;

import com.example.demo.model.Personne;

@RepositoryRestResource
public interface PersonneRepository extends JpaRepository<Personne, Long> {
    @RestResource(path="nom")
    List<Personne> findByNomContains(@Param("nom") String nom);
}
```

Spring Data REST

Pour renommer le chemin vers la méthode précédente, on utilise `@RestResource`

```
package com.example.demo.dao;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.repository.query.Param;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
import org.springframework.data.rest.core.annotation.RestResource;

import com.example.demo.model.Personne;

@RepositoryRestResource
public interface PersonneRepository extends JpaRepository<Personne, Long> {
    @RestResource(path="nom")
    List<Personne> findByNomContains(@Param("nom") String nom);
}
```

URL pour tester

<http://localhost:8080/ws/personnes/search/nom?nom=i>

Spring Data REST

Pour effectuer une recherche selon le nom et paginer le résultat

```
package com.example.demo.dao;

import java.util.List;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.repository.query.Param;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
import org.springframework.data.rest.core.annotation.RestResource;
import com.example.demo.model.Personne;

@RepositoryRestResource
public interface PersonneRepository extends JpaRepository<Personne, Long> {
    @RestResource(path="nom")
    List<Personne> findByNomContains(@Param("nom") String nom, Pageable pageable);
}
```

Spring Data REST

Pour effectuer une recherche selon le nom et paginer le résultat

```
package com.example.demo.dao;

import java.util.List;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.repository.query.Param;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
import org.springframework.data.rest.core.annotation.RestResource;
import com.example.demo.model.Personne;

@RepositoryRestResource
public interface PersonneRepository extends JpaRepository<Personne, Long> {
    @RestResource(path="nom")
    List<Personne> findByNomContains(@Param("nom") String nom, Pageable pageable);
}
```

URL pour tester

```
http://localhost:8080/ws/personnes/search/nom?nom=i&page=0&size=2
```

Spring Data REST

Exercice 1

Testez, avec **Postman**, les deux méthodes **HTTP** `PUT` et `DELETE` qui permettront de modifier ou supprimer une personne.

© Achref EL MOUEZ

Spring Data REST

Exercice 1

Testez, avec **Postman**, les deux méthodes **HTTP** `PUT` et `DELETE` qui permettront de modifier ou supprimer une personne.

Exercice 2

Créer une application **Angular** qui permet à un utilisateur, via des interfaces graphiques) la gestion de personnes (ajout, modification, suppression, consultation et recherche) en utilisant les Web Services définis par **Spring Data REST**.

Spring Data REST

Créons une entité Adresse dans `com.example.demo.model`

```
@NoArgsConstructor
@Entity
@Data
@RequiredArgsConstructor
public class Adresse {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @NonNull
    private String rue;
    @NonNull
    private String codePostal;
    @NonNull
    private String ville;
}
```

Spring Data REST

Préparons notre interface **DAO** AdresseRepository

```
package com.example.demo.dao;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.rest.core.annotation.
    RepositoryRestResource;

import com.example.demo.model.Adresse;

@RepositoryRestResource
public interface AdresseRepository extends JpaRepository <Adresse, Long
    > {

}
```

Spring Data REST

Dans `Personne`, ajoutons l'association avec `Adresse`

```
@NoArgsConstructor
@AllArgsConstructor
@Data
@Entity
@RequiredArgsConstructor
public class Personne {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long num;
    @NonNull
    private String nom;
    @NonNull
    private String prenom;
    @NonNull
    @ManyToMany(cascade = CascadeType.ALL)
    private List <Adresse>adresses;
}
```

Pour alimenter la base de données avec quelques données au démarrage de l'application, modifions la classe de démarrage

```
@Override
public void run(ApplicationArguments args) throws Exception {
    Personne personnel = new Personne("wick", "john", Arrays.asList(
        new Adresse("paradis", "13015", "Marseille"),
        new Adresse("lacanau", "13700", "Marignane")
    ));
    Personne personne2 = new Personne("dalton", "jack", Arrays.asList(
        new Adresse("défense", "75000", "Paris"),
        new Adresse("five", "59000", "Lille"),
        new Adresse("corum", "34000", "Montpellier")
    ));
    Personne personne3 = new Personne("maggio", "carol", Arrays.asList(
        new Adresse("gabriel péri", "69008", "Lyon"),
        new Adresse("gerland", "69007", "Lyon")
    ));
    Personne personne4 = new Personne("cohen", "sophie", Arrays.asList(
        new Adresse("prado", "13008", "Marseille")
    ));
    personneRepository.saveAll(Arrays.asList(personnel, personne2,
        personne3, personne4));
}
```

Spring Data REST

Pour tester, il faut aller sur

- `http://localhost:8080/ws/personnes` pour récupérer la liste des personnes
- ou sur `http://localhost:8080/ws/personnes/1/adresses` pour consulter les adresses de la personne ayant l'identifiant 1
- ou sur `http://localhost:8080/ws/adresses` pour consulter la liste des adresses

© Achref EL

Spring Data REST

Pour tester, il faut aller sur

- `http://localhost:8080/ws/personnes` pour récupérer la liste des personnes
- ou sur `http://localhost:8080/ws/personnes/1/adresses` pour consulter les adresses de la personne ayant l'identifiant 1
- ou sur `http://localhost:8080/ws/adresses` pour consulter la liste des adresses

Remarques

- Pas d'accès aux personnes depuis adresses
- La relation n'est pas bidirectionnelle

Spring Data REST

Ajoutons `Personne` dans `Adresse` pour avoir une association bidirectionnelle

```
@NoArgsConstructor
@AllArgsConstructor
@Data
@Entity
@RequiredArgsConstructor
public class Adresse {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @NotNull
    private String rue;
    @NotNull
    private String codePostal;
    @NotNull
    private String ville;
    @ManyToMany(mappedBy = "adresses")
    private List<Personne> personnes;
}
```

Spring Data REST

Pour tester, il faut aller sur

- `http://localhost:8080/ws/adresses` pour récupérer la liste des adresses
- ou sur
`http://localhost:8080/ws/adresses/1/personnes` pour consulter les personnes associées à l'adresse ayant l'identifiant 1

Spring Data REST

Utilisons `@RestResource` pour modifier le chemin d'accès aux personnes depuis une adresse

```
@NoArgsConstructor
@AllArgsConstructor
@Data
@Entity
@RequiredArgsConstructor
public class Adresse {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @NonNull
    private String rue;
    @NonNull
    private String codePostal;
    @NonNull
    private String ville;
    @RestResource(path = "per")
    @ManyToMany(mappedBy = "adresses")
    private List<Personne> personnes;
}
```

Spring Data REST

Pour tester, il faut aller sur

- `http://localhost:8080/ws/adresses` pour récupérer la liste des adresses
- ou sur `http://localhost:8080/ws/adresses/1/per` pour consulter les personnes associées à l'adresse ayant l'identifiant 1

Spring Data REST

Pour charger les données relatives aux personnes dans chaque adresse

```
@NoArgsConstructor
@AllArgsConstructor
@Data
@Entity
@RequiredArgsConstructor
public class Adresse {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @NonNull
    private String rue;
    @NonNull
    private String codePostal;
    @NonNull
    private String ville;
    @RestResource(exported = false)
    @ManyToMany(mappedBy = "adresses")
    private List<Personne> personnes;
}
```

Spring Data REST

Attention, ceci déclenche une boucle infinie

```
@NoArgsConstructor
@AllArgsConstructor
@Data
@Entity
@RequiredArgsConstructor
public class Personne {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long num;
    @NonNull
    private String nom;
    @NonNull
    private String prenom;
    @NonNull
    @ManyToMany(cascade = CascadeType.ALL)
    @RestResource(exported = false)
    private List <Adresse>adresses;
}
```

Spring Data REST

Par défaut

- **Spring Data REST** n'expose pas l'identifiant
- Pour le faire, on peut injecter et utiliser `RepositoryRestConfiguration` dans la classe de démarrage

Spring Data REST

Pour exposer l'identifiant d'une entité, on utilise `RepositoryRestConfiguration`

```
@SpringBootApplication
public class CoursSpringDataRestApplication implements ApplicationRunner {

    @Autowired
    private PersonneRepository personneRepository;

    @Autowired
    private RepositoryRestConfiguration repositoryRestConfiguration;

    public static void main(String[] args) {
        SpringApplication.run(CoursSpringDataRestApplication.class, args);
    }

    @Override
    public void run(ApplicationArguments args) throws Exception {
        repositoryRestConfiguration.exposeIdsFor(Personne.class, Adresse.class);
        // + le code précédent
    }
}
```

Spring Data REST

Projection

- Si dans certains cas, je veux seulement exposer quelques attributs de ma classe (par exemple `ville` et `codePostal` pour `Adresse`).
- On peut indiquer les méthodes (getters) à exposer dans une interface annotée par `Projection`.
- L'interface de projection doit être dans le même package que le modèle (entité)

Spring Data REST

Définissons l'interface de projection AdresseProjection

```
package com.example.demo.model;

import org.springframework.data.rest.core.config.Projection;

@Projection(types = { Adresse.class }, name = "a1")
public interface AdresseProjection {
    public String getCodePostal();
    public String getVille();
}
```

Spring Data REST

Pour tester

- Allez sur
`http://localhost:8080/ws/adresses?projection=a1`
- Vérifiez que seuls le code postal et la ville sont exposés.

© Achref EL

Spring Data REST

Pour tester

- Allez sur
`http://localhost:8080/ws/adresses?projection=a1`
- Vérifiez que seuls le code postal et la ville sont exposés.

Remarque

Une projection permet d'exposer même les attributs annotés par `@JsonIgnore`.

Spring Data REST

On peut aussi utiliser une projection et l'annotation `@Value` pour fusionner les attributs

```
package com.example.demo.model;

import org.springframework.beans.factory.annotation.
    Value;
import org.springframework.data.rest.core.config.
    Projection;

@Projection(types = { Adresse.class }, name = "a1")
public interface AdresseProjection {
    @Value("#{ target.ville + ' ' + target.
        codePostal }")
    String getVilleDetails();
}
```

Spring Data REST

Extrait (Excerpts)

- une projection automatiquement appliquée à un repository **REST**
- On le précise dans l'annotation `@RepositoryRestResource` avec l'attribut `excerptProjection`

Spring Data REST

Exemple

```
package com.example.demo.dao;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

import com.example.demo.model.Adresse;
import com.example.demo.model.AdresseProjection;

@RepositoryRestResource(excerptProjection = AdresseProjection.class)
public interface AdresseRepository extends JpaRepository <Adresse, Long> {

}
```

Spring Data REST

Remarque

- En allant sur `http://localhost:8080/ws/adresses?projection=a1`, on obtient une seule clé `villeDetails`
- En allant sur `http://localhost:8080/ws/adresses`, on obtient une seule clé `villeDetails`