Spring MVC : services 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

Plan

Spring MVC

Service web (WS pour Web Service) ?

- Un programme (ensemble de fonctionnalités exposées en temps réel et sans intervention humaine)
- Accessible via internet, ou intranet
- Indépendant de tout système d'exploitation
- Indépendant de tout langage de programmation
- Utilisant un système standard d'échange (**XML** ou **JSON**), ces messages sont généralement transportés par des protocoles internet connus **HTTP** (ou autres comme **FTP**, **SMTP**...)
- Pouvant communiquer avec d'autres WS

Spring MVC

Les WS peuvent utiliser les technologies web suivantes :

- **HTTP** (Hypertext Transfer Protocol) : le protocole, connu, utilisé par le World Wide Web et inventé par Roy Fiedling.
- **REST** (Representational State Transfer) : une architecture de services Web, créée aussi par Roy Fielding en 2000 dans sa thèse de doctorat.
- **SOAP** (Simple object Access Protocol) : un protocole, défini par Microsoft et IBM ensuite standarisé par **W3C**, permettant la transmission de messages entre objets distants (physiquement distribués).
- **WSDL** (Web Services Description Language) : est un langage de description de service web utilisant le format **XML** (standardisé par le **W3C** depuis 2007).
- **UDDI** (Universal Description, Discovery and Integration) : un annuaire de WS.

HTTP, **REST** (Representational State Transfer) et **RESTful** ?

- Les API REST sont basées sur le protocole **HTTP** (architecture client/serveur) et utilisent le concept de ressource.
- Une ressource est identifiée par une URI unique.
- L'API REST utilise donc des méthodes suivantes pour l'échange de données entre client et serveur
  - `GET` pour la récupération,
  - `POST` pour l'ajout,
  - `DELETE` pour la suppression,
  - `PUT` pour la modification,
  - ...
- Plusieurs formats possibles pour les données échangées : texte, **XML**, **JSON**...
- **RESTful** est l'adjectif qui désigne une API REST.

Spring MVC

Rôle du contrôleur dans une application **MVC**

- Le contrôleur reçoit une requête **HTTP** et communique avec modèle, service, constructeur de formulaires pour retourner une réponse **HTTP** contenant une page **HTML**.
- Le contrôleur peut aussi retourner une réponse **HTTP** ne contenant pas de vue.
- Il retourne des données sous format **JSON**, **XML**...

Spring MVC

Rôle du contrôleur dans une application **MVC**

- Le contrôleur reçoit une requête **HTTP** et communique avec modèle, service, constructeur de formulaires pour retourner une réponse **HTTP** contenant une page **HTML**.
- Le contrôleur peut aussi retourner une réponse **HTTP** ne contenant pas de vue.
- Il retourne des données sous format **JSON**, **XML**...

Ceci est l'objet de ce chapitre.

Considérons le contrôleur `PersonneRestController`

```
@Controller
public class PersonneRestController {
  @Autowired
  private PersonneRepository personneRepository;

  @GetMapping("/personnes")
  public String getPersonnes(Model model) {
    List<Personne> personnes = personneRepository.findAll();
    model.addAttribute("personnes", personnes);
    return "showPersonnes";
  }
  @GetMapping("/personnes/{id}")
  public String getPersonne(@PathVariable("id") long id, Model model) {
    Personne personne = personneRepository.findById(id).orElse(null);
    model.addAttribute("personne", personne);
    return "showPersonne";
  }
  @PostMapping("/addPersonne")
  public String addPersonne(Personne personne, Model model) {
    Personne personne2 = personneRepository.save(personne);
    model.addAttribute("personne", personne2);
    return "showPersonne";
  }
}
```

Spring MVC

Explication

- Le contrôleur `PersonneRestController` contient trois méthodes permettant soit de récupérer une ou plusieurs personnes de la base de données et d'afficher le résultat dans la vue, soit d'ajouter une nouvelle personne dans la base de données et l'afficher dans la vue.
- Le contrôleur retourne deux vues (soit `showPersonnes`, soit `showPersonne`) qui servent principalement à afficher des données récupérées de la base de données

Spring MVC

Remarques

- Comment fait-on si le contrôleur doit récupérer et retourner les données sans les afficher dans une vue (Notre projet **Spring** devient donc un service web) ?
- L'affichage sera accordé à un framework Front-end tel que **Angular** ou autre

Nouveau contenu du contrôleur `PersonneRestController`

```
@Controller
public class PersonneRestController {

  @Autowired
  private PersonneRepository personneRepository;

  @GetMapping("/personnes")
  public String getPersonnes() {
    return personneRepository.findAll().toString();
  }

  @GetMapping("/personnes/{id}")
  public String getPersonne(@PathVariable("id") long id) {
    return personneRepository.findById(id).orElse(null).toString();
  }

  @PostMapping("/personnes")
  public String addPersonne(Personne personne) {
    System.out.println(personne);
    return personneRepository.save(personne).toString();
  }
}
```

Spring MVC

Problématique

- Si on saisit l'URL `localhost:8080/firstspringmvc/personnes` dans la barre d'adresse, la méthode `getPersonnes` sera exécutée
- La liste de personnes sera récupérée de la base de données
- Comme le type de la valeur de retour de cette méthode est `String`, le contrôleur va chercher à afficher la vue dont le nom est précisé après `return`
- Mais cette valeur ne correspond pas à une vue, elle correspond plutôt à des valeurs récupérées de la base de données.

Pour corriger ça, on peut utiliser l'annotation `@ResponseBody`

```
@Controller
public class PersonneRestController {

  @Autowired
  private PersonneRepository personneRepository;

  @GetMapping("/personnes")
  @ResponseBody
  public String getPersonnes() {
    return personneRepository.findAll().toString();
  }
  @GetMapping("/personnes/{id}")
  @ResponseBody
  public String getPersonne(@PathVariable("id") long id) {
    return personneRepository.findById(id).orElse(null).toString();
  }

  @PostMapping("/personnes")
  @ResponseBody
  public String addPersonne(Personne personne) {
    System.out.println(personne);
    return personneRepository.save(personne).toString();
  }
}
```

Spring MVC

Pour tester

allez à l'URL

- `localhost:8080/firstspringmvc/personnes`
- Ou `localhost:8080/firstspringmvc/personnes/1`

Spring MVC

Pour tester

allez à l'URL

- `localhost:8080/firstspringmvc/personnes`
- Ou `localhost:8080/firstspringmvc/personnes/1`

Constat

- Le résultat obtenu est une chaîne de caractère ou un tableau de chaîne de caractère
- Sachant que **Spring** nous offre la possibilité de récupérer le résultat sous format **JSON**

Pour corriger ça, on peut utiliser l'annotation `@ResponseBody`

```
@Controller
public class PersonneRestController {

  @Autowired
  private PersonneRepository personneRepository;

  @GetMapping("/personnes")
  @ResponseBody
  public List<Personne> getPersonnes() {
    return personneRepository.findAll();
  }
  @GetMapping("/personnes/{id}")
  @ResponseBody
  public Personne getPersonne(@PathVariable("id") long id) {
    return personneRepository.findById(id).orElse(null);
  }

  @PostMapping("/personnes")
  @ResponseBody
  public Personne addPersonne(Personne personne) {
    System.out.println(personne);
    return personneRepository.save(personne);
  }
}
```

Spring MVC

Avant de tester, il faut ajouter une dépendance `jackson` pour assurer la conversion d'un objet `Java` en objet `JSON` ou `XML`

```
<!-- https://mvnrepository.com/artifact/com.fasterxml.
   jackson.dataformat/jackson-dataformat-xml -->
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
    <version>2.10.1</version>
</dependency>
```

Spring MVC

Comment préciser le format (**JSON** ou **XML**) souhaité ?

- Utilisez **Postman** en choisissant la méthode `GET`
- Dans `Headers`, ajoutez les deux clés `Accept` et `Content-Type` avec la valeur `application/json` (ou `application/xml`)
- Envoyez

On peut aussi indiquer avec l'attribut `produces` les formats acceptés

```
@Controller
public class PersonneRestController {
  @Autowired
  private PersonneRepository personneRepository;

  @GetMapping(value = "/personnes", produces = { MediaType.
     APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE })
  @ResponseBody
  public List<Personne> getPersonnes() {
    return personneRepository.findAll();
  }
  @GetMapping("/personnes/{id}")
  @ResponseBody
  public Personne getPersonne(@PathVariable("id") long id) {
    return personneRepository.findById(id).orElse(null);
  }

  @PostMapping("/personnes")
  @ResponseBody
  public Personne addPersonne(Personne personne) {
    System.out.println(personne);
    return personneRepository.save(personne);
  }
}
```

Spring MVC

Pour ajouter une personne

- Utilisez **Postman** en précisant la méthode `POST`
- Dans `Headers`, ajoutez les clés `Accept` et `Content-Type` avec la valeur `application/json`
- Dans `Body`, cochez `raw` et sélectionnez `JSON`
- Ajoutez l'objet **JSON** suivant

```
{
    "nom": "maggio",
    "prenom": "carol"
}
```

- Envoyez

Spring MVC

Résultat

- Erreur et la personne n'a pas été ajoutée dans la base de données
- Message affiché dans la console : `{"num":null, "nom":null, "prenom":null}`

Spring MVC

Pour récupérer l'objet envoyé défini dans le corps de la requête `HTTP` et assurer le binding avec l'objet défini comme paramètre de la méthode `addPersonne()`, on doit utiliser l'annotation `@RequestBody`

```
@PostMapping("/personnes")
@ResponseBody
public Personne addPersonne(@RequestBody Personne personne) {

  System.out.println(personne);
  return personneRepository.save(personne);

}
```

Spring MVC

On peut aussi indiquer avec l'attribut `consumes` les formats acceptés

```
@PostMapping(value = "/personnes", consumes = { MediaType.
   APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE })
@ResponseBody
public Personne addPersonne(Personne personne) {
    System.out.println(personne);
    return personneRepository.save(personne);
}
```

Spring MVC

Constats

- Toutes les méthodes du contrôleur ne retournent pas de vue
- Toutes ces méthodes sont annotées par `@ResponseBody`

Spring MVC

Constats

- Toutes les méthodes du contrôleur ne retournent pas de vue
- Toutes ces méthodes sont annotées par `@ResponseBody`

On peut optimiser

On peut utiliser l'annotation `@RestController`

Spring MVC

Remplaçons `@ResponseBody` **par** `@RestController`

```
@RestController
public class PersonneRestController {

  @Autowired
  private PersonneRepository personneRepository;

  @GetMapping("/personnes")
  public List<Personne> getPersonnes() {
    return personneRepository.findAll();
  }

  @GetMapping("/personnes/{id}")
  public Personne getPersonne(@PathVariable("id") long id) {
    return personneRepository.findById(id).orElse(null);
  }

  @PostMapping("/personnes")
  public Personne addPersonne(@RequestBody Personne personne) {
    System.out.println(personne);
    return personneRepository.save(personne);
  }
}
```

Spring MVC

Exercice 1

Écrire puis tester les deux méthodes **HTTP** `put` et `delete` qui permettront de modifier ou supprimer une personne.

Spring MVC

Pour la suite

- supposons qu'une personne peut avoir une ou plusieurs adresses
- commençons par créer une entité `Adresse` avec 4 attributs `id`, `rue`, `ville` et `codePostal`
- déclarons dans `Personne` une liste de `Adresse`
- utilisons **Postman** pour ajouter des personnes dans la base de données avec leurs adresses

Spring MVC

La classe `Adresse`

```
@Entity
public class Adresse {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;
  private String rue;
  private String codePostal;
  private String ville;

  // N'oublions pas les getters / setters / toString /
     Constructeur sans paramètre
}
```

Spring MVC

Le nouveau contenu de la classe `Personne`

```
@Entity
public class Personne {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long num;
  private String nom;
  private String prenom;

  @ManyToMany(cascade = { CascadeType.MERGE, CascadeType.REFRESH},
     fetch = FetchType.EAGER)
  private List<Adresse> adresses = new ArrayList<Adresse>();

  // N'oublions pas les getters / setters / toString
}
```

Seules les propriétés `MERGE` et `REFRESH` sont demandées.

Spring MVC

Le contenu de `AdresseRepository`

```
package org.eclipse.firstspringmvc.dao;

import org.eclipse.firstspringmvc.model.Adresse;
import org.springframework.data.jpa.repository.
   JpaRepository;

public interface AdresseRepository extends
   JpaRepository<Adresse, Long> {

}
```

Spring MVC

Exemple d'objet JSON à persister (en utilisant Postman)

```
{
  "nom": "el mouelhi",
  "prenom": "achref",
  "adresses": [
    {
      "rue": "paradis",
      "ville": "marseille",
      "codePostal": "13015"
    }
  ]
}
```

Message d'erreur parlant d'un objet faisant référence à une instance non sauvegardée

Spring MVC

Modifions `PersonneRestController` afin qu'il puisse attacher les entités inverses avant de les persister

```
@PostMapping("/personnes")
public Personne addPersonne(@RequestBody Personne personne) {

  System.out.println(personne);
  List <Adresse> adresses = personne.getAdresses();
  for (Adresse adresse : adresses) {
    Adresse adr = null;
    if (adresse.getId() != null) {
      adr = adresseRepository.findById(adresse.getId()).orElse(
        null);
      adresses.set(adresses.indexOf(adresse), adr);
    }  else {
      adr = adresseRepository.save(adresse);
    }
  }
  return personneRepository.saveAndFlush(personne);
}
```

Spring MVC

La réponse de la requête précédente (avec les clés primaires)

```
{
  "num": 1,
  "nom": "el mouelhi",
  "prenom": "achref",
  "adresses": [
    {
      "id": 1,
      "rue": "paradis",
      "ville": "marseille",
      "codePostal": "13015"
    }
  ]
}
```

Spring MVC

Et si on voulait ajouter une personne en lui affectant l'adresse précédente

```
{
    "nom": "wick",
    "prenom": "john",
    "adresses": [
        {
            "id": 1,
            "rue": "paradis",
            "codePostal": "13015",
            "ville": "marseille"
        }
    ]
}
```

Spring MVC

Et si on voulait ajouter une personne en lui affectant l'adresse précédente

```
{
    "nom": "wick",
    "prenom": "john",
    "adresses": [
        {
            "id": 1,
            "rue": "paradis",
            "codePostal": "13015",
            "ville": "marseille"
        }
    ]
}
```

Pour l'adresse, seule la clé primaire compte, les autres attributs sont facultatifs.

Spring MVC

En renvoyant l'objet précédent, la réponse est :

```
{
    "num": 2,
    "nom": "wick",
    "prenom": "john",
    "adresses": [
        {
            "id": 1,
            "rue": "paradis",
            "codePostal": "13015",
            "ville": "marseille"
        }
    ]
}
```

Spring MVC

Modifions la classe `Adresse` pour rendre son association avec la classe `Personne` bidirectionnelle

```
@Entity
public class Adresse {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;
  private String rue;
  private String codePostal;
  private String ville;
  @ManyToMany(fetch = FetchType.EAGER ,mappedBy="adresses")
  private List<Personne> personnes = new ArrayList<Personne>();

  // N'oublions pas les getter et setter
}
```

Relancer le projet, en cas d'erreur, supprimer et recréer la base de données.

Spring MVC

Remarque

En essayant de consulter la liste de personnes (avec leurs adresses respectives), on a une boucle infinie (circular reference) car l'association est désormais bidirectionnelle.

Spring MVC

Remarque

En essayant de consulter la liste de personnes (avec leurs adresses respectives), on a une boucle infinie (circular reference) car l'association est désormais bidirectionnelle.

Solution

Pou arrêter la boucle infinie, on peut ajouter l'annotation `@JsonIgnoreProperties` dans la classe `Adresse`

Spring MVC

Nouveau contenu de la classe `Adresse`

```
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;

//autres imports

@Entity
public class Adresse {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;
  private String rue;
  private String codePostal;
  private String ville;

  @JsonIgnoreProperties("adresses")
  @ManyToMany(fetch = FetchType.EAGER,mappedBy="adresses")
  private List<Personne> personnes = new ArrayList<Personne>();

  // + le contenu précédent
}
```

Spring MVC

Nouveau contenu de la classe `Personne`

```
@Entity
public class Personne {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long num;
    @Size(min = 2)
    @NotEmpty(message = "le champ nom est obligatoire")
    private String nom;
    @NotEmpty(message = "le champ prénom est obligatoire")
    @Size(min = 2)
    private String prenom;

    @JsonIgnoreProperties("personnes")
    @ManyToMany(cascade = { CascadeType.MERGE, CascadeType.REFRESH },
       fetch =    FetchType.EAGER)
    private List<Adresse> adresses = new ArrayList<Adresse>();

    // + le contenu précédent
}
```

Spring MVC

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**.

Spring MVC

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**.

Pour régler le problème **CORS**

Il faut ajouter l'annotation `@CrossOrigin` au contrôleur.

Spring MVC

curl : **C**lient for **URL**s

- Commande permettant le transfert de données
- Utilisant plusieurs protocoles

Spring MVC

Pour demander la liste de personnes

```
curl --request GET http://localhost:8080/firstspringmvc/personnes
```

Spring MVC

Pour demander la liste de personnes

```
curl --request GET http://localhost:8080/firstspringmvc/personnes
```

Par défaut, la méthode `HTTP` est `GET`

```
curl http://localhost:8080/firstspringmvc/personnes
```

Spring MVC

Pour demander la liste de personnes

```
curl --request GET http://localhost:8080/firstspringmvc/personnes
```

Par défaut, la méthode `HTTP` est `GET`

```
curl http://localhost:8080/firstspringmvc/personnes
```

Pour récupérer seulement l'entête de la réponse

```
curl -I http://localhost:8080/firstspringmvc/personnes
```

Spring MVC

Pour avoir un résultat au format XML (Attention aux espaces non-nécessaires)

```
curl --header "Accept:application/xml" http://localhost:8080/
   firstspringmvc/personnes
```

Spring MVC

Pour avoir un résultat au format XML (Attention aux espaces non-nécessaires)

```
curl --header "Accept:application/xml" http://localhost:8080/
   firstspringmvc/personnes
```

Le raccourci

```
curl -H "Accept:application/xml" http://localhost:8080/firstspringmvc/
   personnes
```

Spring MVC

Pour avoir un résultat au format XML (Attention aux espaces non-nécessaires)

```
curl --header "Accept:application/xml" http://localhost:8080/
   firstspringmvc/personnes
```

Le raccourci

```
curl -H "Accept:application/xml" http://localhost:8080/firstspringmvc/
   personnes
```

Pour sauvegarder le résultat dans un fichier

```
curl -H "Accept:application/xml" -o personnes.xml http://localhost
   :8080/firstspringmvc/personnes
```

Spring MVC

Pour ajouter une personne

```
curl --request POST -H "Content-Type:application/json" http://localhost
   :8080/firstspringmvc/personnes --data "{\"nom\":\"wick\",\"prenom
   \":\"john\"}"
```

Spring MVC

Pour ajouter une personne

```
curl --request POST -H "Content-Type:application/json" http://localhost
   :8080/firstspringmvc/personnes --data "{\"nom\":\"wick\",\"prenom
   \":\"john\"}"
```

Notons que `--request POST` est facultatif si nous utilisons `-d`, car ce dernier implique une requête `POST`.

```
curl -H "Content-Type:application/json" http://localhost:8080/
   firstspringmvc/personnes -d "{\"nom\":\"wick\",\"prenom\":\"john\"}"
```

Spring MVC

Pour ajouter une personne

```
curl --request POST -H "Content-Type:application/json" http://localhost
   :8080/firstspringmvc/personnes --data "{\"nom\":\"wick\",\"prenom
   \":\"john\"}"
```

Notons que `--request POST` est facultatif si nous utilisons `-d`, car ce dernier implique une requête `POST`.

```
curl -H "Content-Type:application/json" http://localhost:8080/
   firstspringmvc/personnes -d "{\"nom\":\"wick\",\"prenom\":\"john\"}"
```

Le mode mono-ligne ne facilite pas la lecture, on peut utiliser le mode multi-lignes avec ^ sous Windows ou \ sous Linux (N'oublions pas l'espace avant)

```
curl -H "Content-Type:application/json" ^
http://localhost:8080/firstspringmvc/personnes ^
-d "{\"nom\":\"wick\",\"prenom\":\"john\"}"
```

Spring MVC

Pour avoir de l'aide

```
curl -h
```

Spring MVC

Gestion d'erreurs

- Si nous envoyons une requête **HTTP** de type **GET** à l'URL `http://localhost:8080/firstspringmvc/personnes/1`, nous obtiendrons un objet **JSON** contenant des informations sur la personne ayant le numéro `1`.
- Si nous demandons des informations sur une personne qui n'existe pas (par exemple numéro `10000`), nous n'obtiendrons pas de réponse mais un statut **200 OK**.
- Ce type de retour peut avoir des graves conséquences au récepteur.
- Nous préférons générer une exception pour un tel cas.

Spring MVC

Commençons par créer notre classe `PersonneNotFoundException`

```
package org.eclipse.firstspringmvc.exception;

public class PersonneNotFoundException extends RuntimeException
    {
    public PersonneNotFoundException() {
        super("Personne introuvable");
    }
}
```

Spring MVC

Utilisons l'exception dans notre méthode `GET`

```
@GetMapping("/personnes/{id}")
public Personne getPersonne(@PathVariable("id") long id)  {
    var personne =  personneRepository.findById(id).orElse(null
       );
    if (personne == null)
        throw new PersonneNotFoundException();
    return personne;
}
```

Spring MVC

En envoyant une requête `GET` à `http://localhost:8080/firstspringmvc/personnes/10000`, la réponse est :

```
état HTTP 500 - Erreur interne du serveur


Type: Rapport d'exception

message: Request processing failed; nested exception
    is org.eclipse.firstspringmvc.exception.
   PersonneNotFoundException: Personne introuvable

description: Le serveur a rencontré une erreur
   interne qui l'a empêché de satisfaire la requête.
```

Spring MVC

Pour modifier le code de 500 à 404, modifions notre classe d'exception

```
package org.eclipse.firstspringmvc.exception;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(HttpStatus.NOT_FOUND)
public class PersonneNotFoundException extends RuntimeException
    {
    public PersonneNotFoundException() {
        super("Personne introuvable");
    }
}
```

Spring MVC

En renvoyant la requête précédente, la réponse est :

```
état HTTP 404 – Non trouvé


Type: Rapport d'état

description: La ressource demandée n'est pas
   disponible.
```

Spring MVC

En renvoyant la requête précédente, la réponse est :

```
état HTTP 404 – Non trouvé


Type: Rapport d'état

description: La ressource demandée n'est pas
   disponible.
```

Le message a disparu.

Spring MVC

Modifions la classe d'exception pour ajouter le message d'erreur

```
package org.eclipse.firstspringmvc.exception;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.
   ResponseStatus;

@ResponseStatus(value= HttpStatus.NOT_FOUND, reason
   = "Personne introuvable")
public class PersonneNotFoundException extends
   RuntimeException {

}
```

Spring MVC

En renvoyant la requête précédente, la réponse est :

```
état HTTP 404 – Non trouvé


Type: Rapport d'état

message Personne introuvable

description: La ressource demandée n'est pas
   disponible.
```

Spring MVC

On peut aussi l'exception `ResponseStatusException`

```
@GetMapping("/personnes/{id}")
public Personne getPersonne(@PathVariable("id") long
    id)  {
    var personne =  personneRepository.findById(id).
       orElse(null);
    if (personne == null)
        throw new ResponseStatusException(
                HttpStatus.NOT_FOUND, "Personne
                   avec id = " + id + " est
                   introuvable");
    return personne;
}
```

Spring MVC

En renvoyant la requête précédente, la réponse est :

```
état HTTP 404 – Non trouvé


Type: Rapport d'état

message Personne avec id = 10000 est introuvable

description: La ressource demandée n'est pas
   disponible.
```

Spring MVC

Pour une opération d'ajout d'élément dans une collection, on retourne le code `HTTP` 201 (`CREATED`)

```
@ResponseStatus(HttpStatus.CREATED)
@PostMapping("/personnes")
public Personne addPersonne(@RequestBody Personne personne) {

    System.out.println(personne);
    List <Adresse> adresses = personne.getAdresses();
    for (Adresse adresse : adresses) {
        Adresse adr = null;
        if (adresse.getId() != null) {
            adr = adresseRepository.findById(adresse.getId()).orElse(
              null);
            adresses.set(adresses.indexOf(adresse), adr);
        } else {
            adr = adresseRepository.save(adresse);
        }
    }
    return personneRepository.saveAndFlush(personne);
  }
```

Spring MVC

Pour plus d'informations sur les codes **HTTP**

https://restfulapi.net/http-status-codes/