Spring MVC

P.Mathieu

Université de Lille

LP DA2I Lille
http://www.iut-a.univ-lille.fr
prenom.nom@univ-lille.fr

5 janvier 2020

1 Le Framework Spring
2 Spring MVC
3 La gestion des formulaires
4 Persistence des données
5 Maintenir la structure des URI
6 Restful en Spring

Le Framework Spring

Principe

Université de Lille

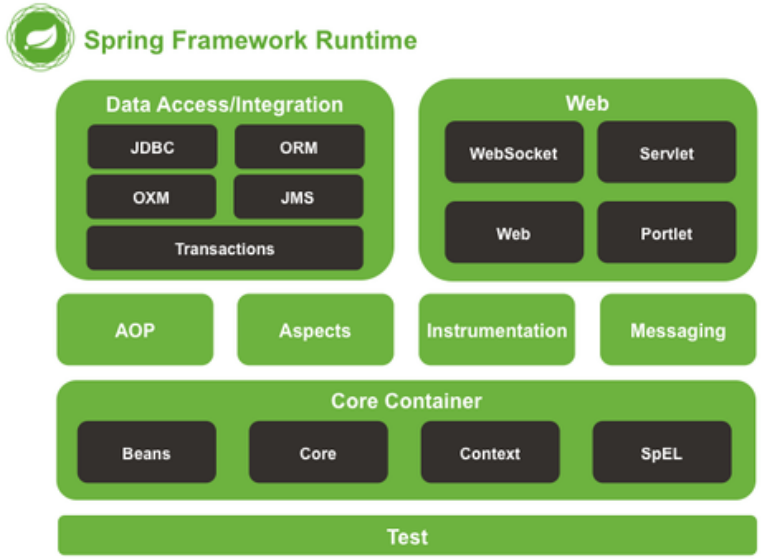
- Framework libre permettant de construire des applications Java
- Considéré comme un conteneur léger
  (les classes n'implémentent pas d'interface, c'est le Framework qui les retrouve.)
- Structure modulaire (avec de nombreux modules $>$ 20) s'appuyant sur 3 concepts clés :
  1. l'inversion de contrôle (injection de dépendances)
  2. La programmation orientée Aspects (AOP)
  3. Une couche d'abstraction

C'est un framework général.
On peut faire du Java "traditionnel" avec Spring !

Le Framework Spring

Composants principaux de Spring

Université de Lille

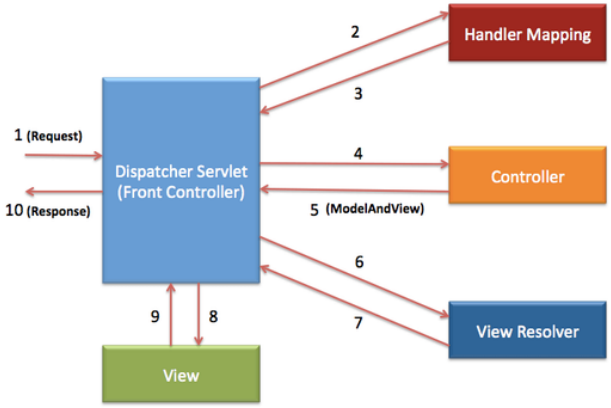


© spring.io

Le Framework Spring

Spring MVC

Composant Web de Spring basé sur une architecture MVC



© s1.quant.com

voir https://www.tutorialspoint.com/spring/
voir https://www.baeldung.com

Le Framework Spring

SpringBoot

- Micro-Framework dérivé de Spring
- Facilite grandement la configuration de Spring
- Fournit un ensemble de composants pré-configurés ainsi qu'un client `spring`
- Fournit un support d'exécution (Tomcat, Jetty, ...)
- Associé par défaut à Groovy
- Aucune configuration XML necessaire

On peut faire du Java "traditionnel" avec SpringBoot
Parmi les composants offerts , il y a Spring MVC

Le Framework Spring

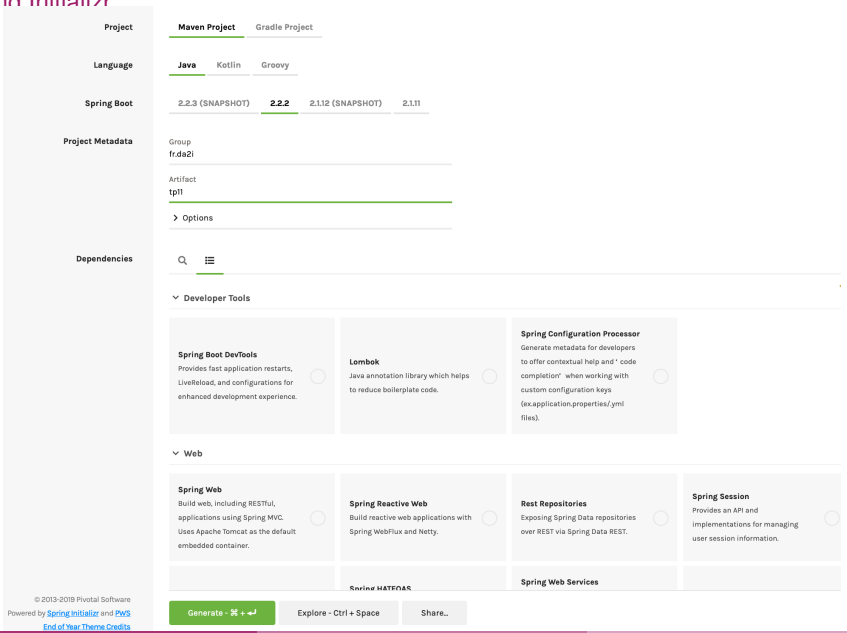
Génération du pom Maven

Spring Initializer fournit automatiquement le `.pom` adapté à SpringBoot et aux dépendances ("Spring Boot Starter Projects") souhaitées

- Aller sur https://start.spring.io/
- Ajouter les dépendances nécessaires WEB, Devtools, JPA, Postgresql, ...
- `mvn package`
  - `mvn spring-boot:run`
  - `java -jar target/monappli-0.0.1-SNAPSHOT.jar`

Le Framework Spring

Spring Initializr



Le Framework Spring

Arborescence

```
tp
|-- pom.xml
|-- src
    |-- main
    |   |-- java
    |   |   |-- fr
    |   |       |-- da2i
    |   |           |-- tp
    |   |               |-- TpApplication.java
    |   |-- resources
    |       |-- application.properties
    |       |-- static
    |       |-- templates
    |-- test
        |-- java
            |-- fr
                |-- da2i
                    |-- tp
                        |-- TpApplicationTests.java
```


1 Le Framework Spring
2 Spring MVC
3 La gestion des formulaires
4 Persistence des données
5 Maintenir la structure des URI
6 Restful en Spring

Spring MVC

SpringBoot en 111 caractères Java

```java
@Controller
class MonPremierControleur
{ @RequestMapping("/")
  @ResponseBody
  String home()
  {
    return "Hello World!";
  }
}
```

```
mvn spring-boot:run
```


Spring MVC

Les Vues

Spring Boot peut être utilisé en mode REST ou en mode génération de pages HTML dynamiques. Il est auto-configuré pour les moteurs de templates suivants :

- FreeMarker
- Groovy
- Thymeleaf (defaut)
- Mustache.js

Dans ce cas il cherche les vues dans `src/main/resources/templates`

Spring MVC

La Vue en JSP

L'usage de JSP nécessite une configuration manuelle dans `application.properties` :

```
spring.mvc.view.prefix=/WEB-INF/jsp/
spring.mvc.view.suffix=.jsp
```

(voir aussi les autres paramètres possibles)

et ajouter la dépendance `tomcat-embed-jasper` au pom

```
<dependency>
  <groupId>org.apache.tomcat.embed</groupId>
  <artifactId>tomcat-embed-jasper</artifactId>
  <scope>provided</scope>
</dependency>
```


Spring MVC

Arborescence complétée

```
tp
|-- pom.xml
|-- src
    |-- main
    |   |-- java
    |   |   |-- fr
    |   |       |-- da2i
    |   |           |-- tp
    |   |               |-- TpApplication.java
    |   |-- resources
    |   |   |-- application.properties
    |   |   |-- static
    |   |   |-- templates
    |   |-- webapp
    |       |- WEB-INF
    |          |- jsp
    |-- test
        |-- java
            |-- fr
                |-- da2i
                    |-- tp
                        |-- TpApplicationTests.java
```


Spring MVC

Les Spring Boot Starter Project

- spring-boot-starter-web-services - SOAP Web Services
- spring-boot-starter-web - Web & RESTful applications
- spring-boot-starter-test - Unit testing and Integration Testing
- spring-boot-starter-jdbc - Traditional JDBC
- spring-boot-starter-security - Authentication and Authorization using Spring Security
- spring-boot-starter-data-jpa - Spring Data JPA with Hibernate
- spring-boot-starter-actuator - To use advanced features like monitoring & tracing to your application out of the box
- spring-boot-starter-undertow, spring-boot-starter-jetty, spring-boot-starter-tomcat - To pick your specific choice of Embedded Servlet Container
- spring-boot-starter-logging - For Logging using logback
- spring-boot-starter-log4j2 - Logging using Log4j2

(Voir la liste de tous les starters)

Spring MVC

Modèle MVC

la vue `maVue.jsp`

```
  ${msg}
```

Le Contrôleur `MonControleur.jsp`

```
@Controller
public class MonControleur
{
    @RequestMapping(value="/")
    public ModelAndView hello()
    {
      ModelAndView model = new ModelAndView();
      model.setViewName("maVue");
      model.addObject("msg","Hello World !");
      return model;
    }
}
```

Spring MVC
Les contrôleurs

Deux types de contrôleurs

- `@Controller`
  Necessite des méthodes qui renvoient vers des vues
- `@RestController`
  qui est équivalent à

```
@Controller
@ResponseBody
```

On peut mettre autant de contrôleurs que l'on souhaite, chacun adapté à une sémantique particulière

Spring MVC
Le Mapping

`@RequestMapping` : Annotation de classe ou de methode de contrôleur

- `@RequestMapping(value="/")`
- `@RequestMapping(value={"/","index"})`
- `@RequestMapping(value={"/","index"} , method=RequestMethod.GET)`

- Si l'annotation est au niveau de la classe, elle est concaténée aux mappings des méthodes
- L'$*$ est supportée dans les noms utilisés
- RequestMethod = GET, POST
  ainsi que PUT, PATCH, DELETE pour archi Restful

Spring MVC
Le modèle

- `ModelAndView` (ancien style). On crée un objet `ModelAndView`. On lui affecte la vue et le modèle. On renvoie cet objet.

```
@RequestMapping(value="/")
ModelAndView hello()
{
  ModelAndView model = new ModelAndView("maVue");
  model.addObject("msg","Hello World !");
  return model;
}
```

- `Model` Le modèle est passé en paramètre de la méthode par le FrontController. La méthode le modifie. la méthode renvoie une reference à la vue sous forme de String.

```
@RequestMapping(value="/")
String hello(Model model)
{
  model.addAttribute("msg","Hello World !");
  return "maVue";
}
```


Spring MVC
Gestion des paramètres de l'URL

`@RequestParam` est utilisé dans les paramètres des méthodes pour récupérer les paramètres dans l'URL. Ils sont automatiquement "castés" et liés aux variables Java

- `String hello(@RequestParam("id") int idPersonne)`
- `String hello(@RequestParam int id)`
- `String hello(@RequestParam Map<String,Integer> reqPar)`
- `String hello(@RequestParam(value="id",required=false) int id)`
- `String hello(@RequestParam(value="id",required=false, defaultValue="5") int id)`

- Par défaut : `Value` : nom de la variable. `required` : true. defaultValue déclenchée si param null ou vide
- Il est possible de tout récupérer d'un coup en passant une Map comme argument : `@RequestParam Map<String,String> reqvars`

Spring MVC

Le Path template

Spring permet dans le RequestMapping d'indiquer des chemins paramétriques.

`@RequestMapping("/da2i/{numetu}"` toutes les requetes commencent par da2i suivi d'un truc

```
@RequestMapping("/da2i/numetu")
public String getetu(@PathVariable int numetu){
// ...
}
```

- Bien distinguer `PathVariable` et `RequestParam`. Ils ne sont pas incompatibles
- Il est possible de tout récupérer d'un coup en mettant une Map comme argument : `@PathVariable Map<String,String> pathvars`

Spring MVC

Autres paramètres des méthodes

D'autres objets J2EE peuvent être passés dans les paramètres

- `HttpServletRequest`
  et donc accès à
  - `application = request.getServletContext();`
  - `config = request.getServletConfig();`
  - `session = request.getSession();`
- `HttpServletResponse`
- `HttpSession`
- `BindingResult`

Spring MVC

L'application de démarrage

Université de Lille

```
@SpringBootApplication
public class Tp11Application {
        public static void main(String[] args) {
                SpringApplication.run(Tp11Application.class, args);

}
```

`@SpringBootApplication` inclut

- `@Configuration` : indique que cette classe est source de définitions des beans.
- `@EnableAutoConfiguration` : charge les composants nécessaires à ce type d'application (le DispatcherServlet si c'est du web par ex).
- `@ComponentScan` : charge les composants du développeur, notamment ses controleurs.

1 Le Framework Spring
2 Spring MVC
3 La gestion des formulaires
4 Persistence des données
5 Maintenir la structure des URI
6 Restful en Spring

La gestion des formulaires

Design Pattern

- Gestion dans un même controleur
- Création d'un Pojo pour récupérer tous les paramètres saisis
- Get pour afficher le formulaire
- Post pour le traiter

```
@RequestMapping(value = { "/addPersonne" }, method = RequestMethod.GET
public String showAddPersonneForm() {
    return "addPersonne";
}

@RequestMapping(value = { "/addPersonne" }, method = RequestMethod.POS
public String processAddPersonneForm(@RequestParam String nom,
                        @RequestParam String prenom  )
{
    Personne p = new Personne(prenom,nom);
    // traitement de cette personne
    return "listerPersonnes";
}
```


La gestion des formulaires

Extension du modèle

`@ModelAttribute` : annotation de méthode ou de paramètre

- AU niveau méthode : la méthode est systématiquement lancée avant l'exécution de tout traitement d'URL
- Au niveau paramètre : Recherche cet objet dans le modèle. S'il ne le trouve pas, il le crée et le remplit avec les paramètres Http de même nom (autoDataBinding) puis le range dans le modèle.

La gestion des formulaires

Formulaire avec @ModelAttribute

```
@RequestMapping(value = { "/addPersonne" }, method = RequestMethod.GET)
public String showAddPersonneForm() {
    return "addPersonne";
}

@RequestMapping(value = { "/addPersonne" }, method = RequestMethod.POST)
public String processAddPersonneForm(@ModelAttribute Personne p)
{
    // traitement de cette personne
    return "listerPersonnes";
}
```


La gestion des formulaires

La vérification des paramètres avec @InitBinder

- Chaque objet Spring peut être annoté pour vérification : `@size @Min, @Max, @Pattern @NotEmpty @email`
- Pour tester un objet passé en paramètre, il suffit de le prefixer par `@valid` dans la méthode qui l'utilise

```
@RequestMapping(method = RequestMethod.POST)
  public String traite (@Valid @ModelAttribute("user") User user,
                                  BindingResult result, Model model)
      if (result.hasErrors()) {
          return "CPasBon";
      else return "COK";
      }
```

- le paramètre `BindingResult` permet de récupérer le résultat de cette validation

La gestion des formulaires
La vérification des paramètres avec @InitBinder

- Ceci est particulièrement bien adapté aux traitements de validation de formulaires.
- L'objet concerné doit avoir été déclaré avec l'annotation `@InitBinder` (eventuellement associée à la méthode de validation, personnalisable)

```
@InitBinder("user")
public void customizeBinding (WebDataBinder binder) { ...}
```


La gestion des formulaires
Le compteur en Spring

`@SessionAttribute` fonctionne de la même manière

affCpt.jsp

```
Valeur du compteur : ${cpt.val}
```

CptControleur.java

```
@RequestMapping("/compteur")
public String gestionCpt(@SessionAttribute Compteur cpt) {
     cpt.incr();
     return "affCpt";
}
```


1 Le Framework Spring
2 Spring MVC
3 La gestion des formulaires
4 Persistence des données
5 Maintenir la structure des URI
6 Restful en Spring

Persistence des données
Spring Data JPA

- Spring Data permet de générer automatiquement une classe "repository" pour chaque Entity Bean
- Cette classe contient les méthodes CRUD, des méthodes de tri ou de pagination
  count, delete, deleteById, save, saveAll, findById, and findAll
- La classe repository est indiquée avec l'annotation `@Repository`
- Elle doit étendre l'une des interfaces `CrudRepository`, `PagingAndSortingRepository`, ou `JpaRepository`.

Persistence des données

Spring Data JPA

POJO `User.java` à créer

```
@Entity
public class User {
    @Id
    private Long id;
    private String nom;
    ....
```

Repository à définir : `UserRepository.java`

```
public interface UserRepository extends CrudRepository<User, Long> {}
// Et on a automatiquement le CRUD sur USER
// On peut bien sûr en rajouter d'autres
```

Dans le controleur

```
@Controller
public class MonControleur {
        @Autowired
        private UserRepository userRepository;

        .... userRepository.findAll())
```


Persistence des données

Spring Data JPA

Toutes les propriété doivent être définies dans `application.properties`

```
debug=false
spring.datasource.driverClassName=org.postgresql.Driver
spring.datasource.url=jdbc:postgresql://localhost/template1
spring.datasource.username=mathieu
spring.datasource.password=philippe

spring.jpa.database=POSTGRESQL
# DB2, DEFAULT, DERBY, H2, HANA, HSQL, INFORMIX, MYSQL, ORACLE, POSTGRESQL, SQL_SERVER, SYBASE

spring.jpa.show-sql=true

spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=create-drop

# spring.jpa.schema-generation.scripts.create-target=schema.sql
spring.jpa.sql-load-script-source=data.sql
logging.level.org.hibernate.SQL=DEBUG

# les annotations classiques jpa peuvent etre utilisées si elles sont préfixées par spring.jpa.pr
spring.jpa.properties.javax.persistence.schema-generation.scripts.action=drop-and-create
spring.jpa.properties.javax.persistence.schema-generation.scripts.create-target=creer.sql
spring.jpa.properties.javax.persistence.schema-generation.scripts.drop-target=detruire.sql
```


Persistence des données

La puissance de Spring data

Spring Data permet d'écrire des requêtes à partir des noms d'attributs et quelques mots-clés (And, Or, Containing, StartingWith, etc). Il se charge de traduire automatiquement ce nom de méthode en requête puis de l'exécuter !

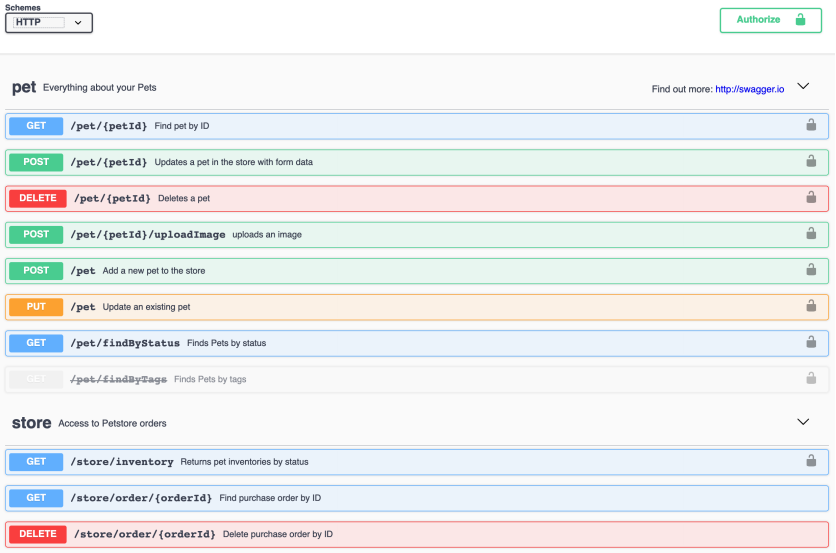
```
public interface PersonneRep extends CrudRepository {
  // recherche une personne par son attribut "nom"
  Personne findByNom(String nom);

  // ici, par son "nom" ou "prenom"
  Personne findByNomOrPrenom(String nom, String prenom);

  List<Personne> findByNomAndPrenomAllIgnoreCase(String nom, String preno

  List<Personne> findByNomOrderByPrenomAsc(String nom)
}
```

Avec Spring Data il y a en général très peu de code à écrire !

1 Le Framework Spring
2 Spring MVC
3 La gestion des formulaires
4 Persistence des données
5 Maintenir la structure des URI
6 Restful en Spring

Maintenir la structure des URI

Swagger.io



1 Le Framework Spring
2 Spring MVC
3 La gestion des formulaires
4 Persistence des données
5 Maintenir la structure des URI
6 Restful en Spring

Restful en Spring

REST : Representationnal State Transfer

- Les services Web sont des services de données via HTTP
- Plusieurs formats historiques, basés sur XML : WSDL, SOAP, UDDI,..., REST
- REST propose un formalisme plus simple
  - fournir les réponses en JSON,
  - utiliser les ordres HTTP standard
    - PUT pour remplacer une ressource (idempotente),
    - PATCH pour la mettre à jour partiellement,
    - GET pour récupérer une ressource(idempotente),
    - POST pour créer cette ressource,
    - DELETE pour la supprimer (idempotente).
  - Pas de sessions

Restful en Spring

Tout est automatique

- Spring permet de gérer l'ensemble des requêtes HTTP
- Avec @RestController L'API Jackson convertit automatiquement les objets en JSON

```
@RestController
public class PersonneController {
  @RequestMapping("/personne", method=RequestMethod.GET)
  public Personne ()
  { personneRepository.findById(5);
    return p;
  }
}
```

(Il est préférable de renvoyer le type abstrait `ResponseEntityf<T>`)

Une appli RESTFUL en Spring est quasi immédiate !