

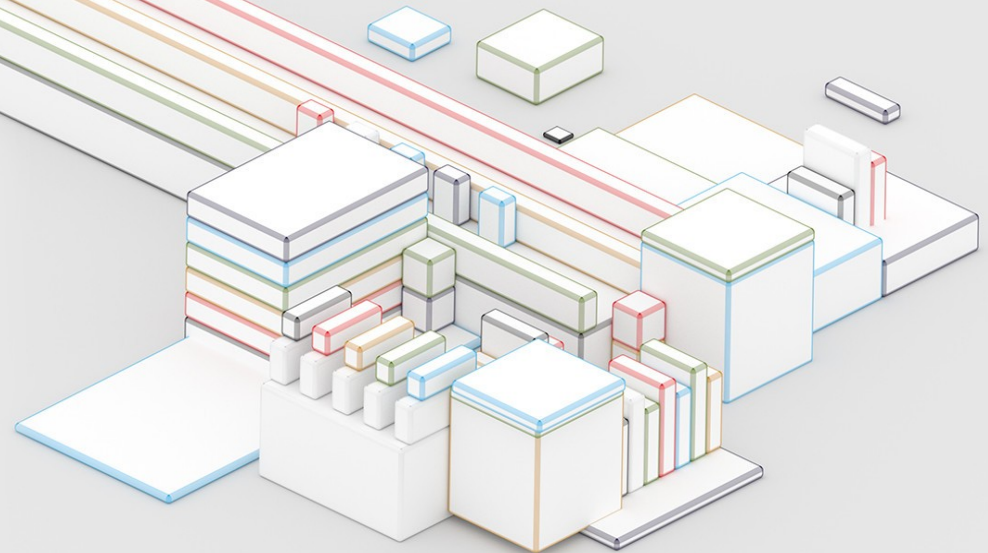
Spring MVC

Christophe Fontaine

cfontaine@dawan.fr

02/01/2023

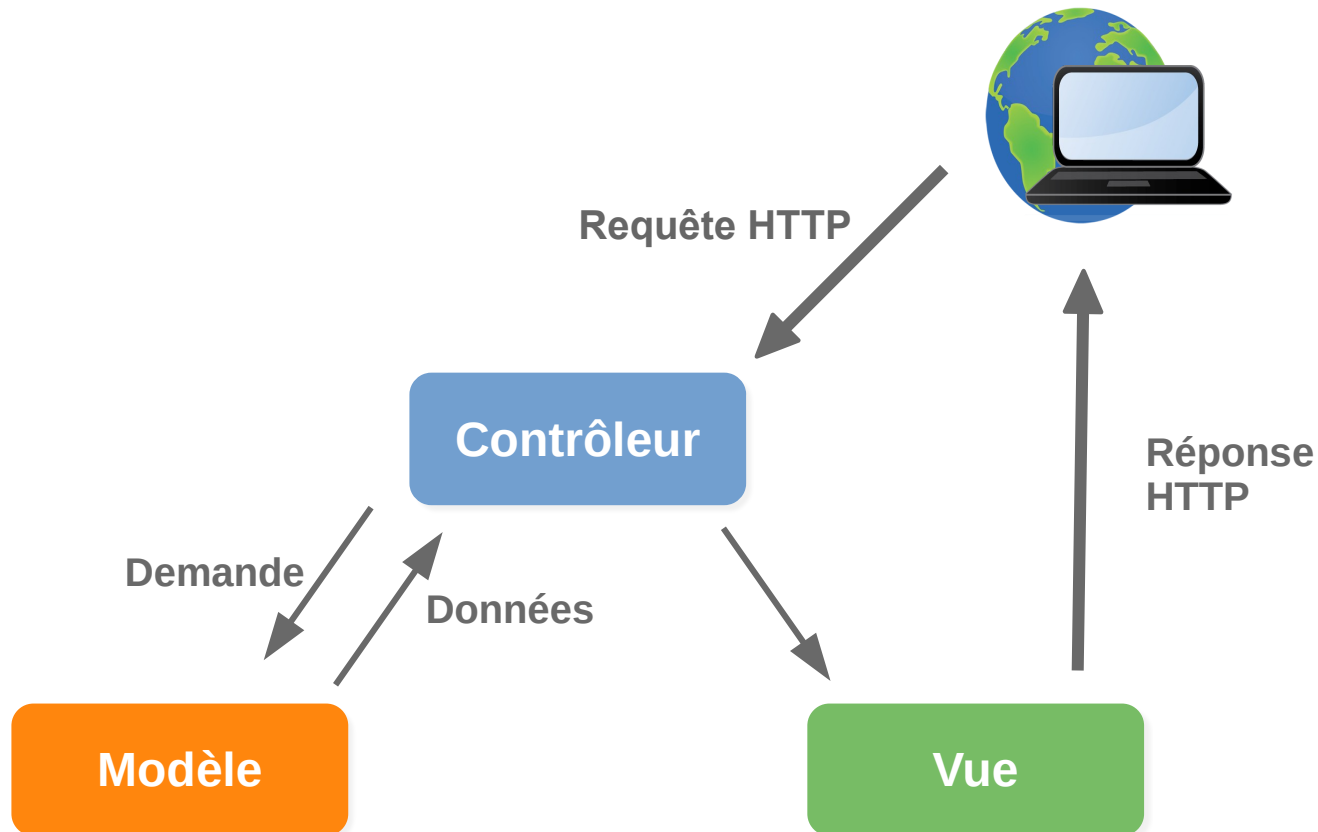
Spring Web MVC



L'architecture MVC

Model-View-Controller

- désigne la séparation des données, la manière de les restituer et du traitement sur ces données



L'architecture MVC2



MVC2

- introduction d'un front controller qui traite toutes les demandes et les renvoie au bon traitement
- n'est pas le successeur de MVC
- est plus complexe que MVC
- sépare la logique de la présentation contrairement à MVC
- est plus flexible que MVC
- est plus adapté pour de grosse application
MVC correspond bien à de petites applications

Frameworks MVC2

- Orienté requête

///Stripes

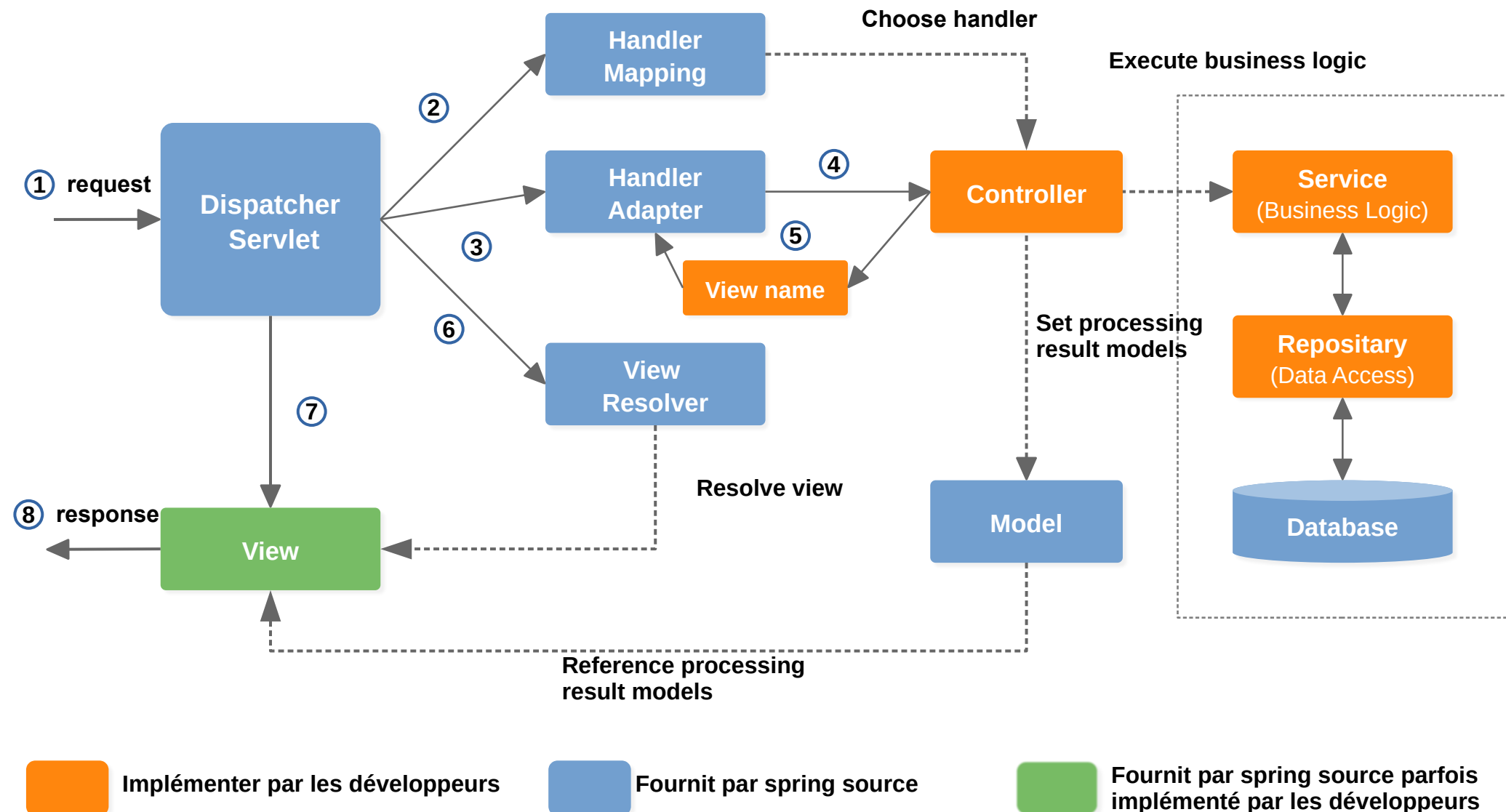


Struts²

- Orienté composant



L'architecture MVC2 de Spring



Fonctionnement



- S'exécute dans un container léger : **Tomcat**
- Front Controller : servlet **DispatcherServlet**
- Contrôleurs : des POJO/JavaBean annotés **@Controller**
- Vues : choix possible de la technologie **jsp** ou **Freemaker, Thymeleaf, Groovy Markup, Tiles ...**
- Un mapping request dans le contrôleur : **@RequestMapping**
- Des objets métiers : Objets Spring ou objets JEE

Configuration du projet Spring MVC



- Créer un projet Maven simple
 - cocher Create a simple project (skip archetype selection)
 - packaging → **war**
- Ajouter dans l'élément **<project>** de pom.xml, pour compiler en java 8 et utiliser UTF-8 avec maven

```
<properties>
  <maven.compiler.source>1.8</maven.compiler.source>
  <maven.compiler.target>1.8</maven.compiler.target>
  <project.build.sourceEncoding>
    UTF-8
  </project.build.sourceEncoding>
</properties>
```

- Ajouter les dépendances à
spring web mvc, servlet 4, jsp 2.33, jstl 1.2 ...

Configuration du projet Spring MVC

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.3.24</version>
  </dependency>
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>4.0.1</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>javax.servlet.jsp-api</artifactId>
    <version>2.3.3</version>
    <scope>provided</scope>
  </dependency>
  ...
</dependencies>
```

Configuration du projet Spring MVC



- Ajouter le plugin maven war

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-war-plugin</artifactId>
      <version>3.3.2</version>
    </plugin>
  </plugins>
</build>
```

- Créer un dossier **WEB-INF** dans **src\main\webapp**
- Ajouter le descripteur de déploiement **web.xml** dans le dossier **WEB-INF**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
  id="WebApp_ID" version="4.0">

</web-app>
```

Configuration du projet Spring MVC



- Maven Update
- Ajouter le serveur tomcat

Projet → propriétés

Filtre sur **target runtime**

↳ cocher le serveur apache tomcat

- Ajouter la vue server
 - windows→show view→server : cliquer sur le lien
No servers are available create new server ...
 - ajouter le serveur tomcat
 - Clique droit sur le serveur→ add and remove
et ajouter le projet

Configuration de l'application via XML



- Configurer le contrôleur principal dans le **web.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ... >
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring/root-context.xml</param-value>
  </context-param>
  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>
  <servlet>
    <servlet-name>springDispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/spring/servlet-context.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>springDispatcherServlet</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

Configuration de l'application via XML



- Configurer Spring dans le fichier de contexte Web : **servlet-context.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd
                           http://www.springframework.org/schema/mvc/spring-mvc.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-context.xsd">

  <mvc:annotation-driven />
  <context:component-scan base-package="fr.dawan.springmvc" />
  <bean id="viewResolver" class="org.springframework.web.servlet.
                                view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/jsp/" />
    <property name="suffix" value=".jsp" />
  </bean>
</beans>
```

Configuration de l'application via XML



```
<mvc:annotation-driven />
```

↳ Indique que les annotations seront utilisées dans le code Java

```
<context:component-scan base-package="fr.dawan.formation" />
```

↳ Précise le package dans lequel seront contenus les classes devant être découvertes par Spring

```
<bean id="viewResolver" class="org.springframework.web.  
view.InternalResourceViewResolver">
```

```
  <property name="prefix" value="/WEB-INF/jsp/" />
```

```
  <property name="suffix" value=".jsp" />
```

```
</bean>
```

↳ Définit la méthode de détermination des vues à utiliser

Configuration de l'application via annotations



- Configurer l'application : équivalent du contenu du **web.xml**
- La définition et le paramétrage de la **DispatcherServlet** est pris en charge par la super classe

```
public class ApplicationInitializer extends
    AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return null;
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class<?>[] { ApplicationConfig.class };
    }

    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }
}
```

Configuration de l'application via annotations

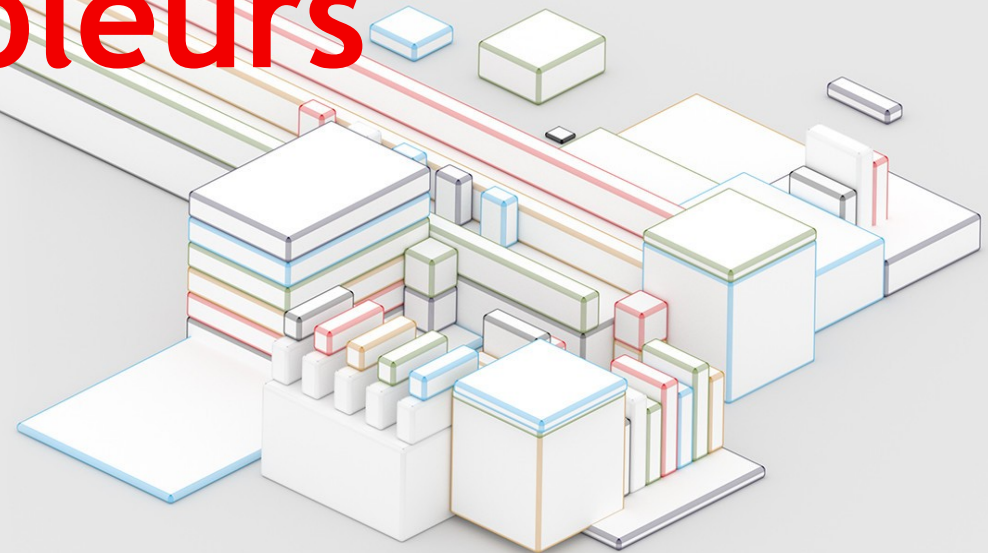


- Configurer Spring : équivalent du fichier **servlet-context.xml**

```
@Configuration
@EnableWebMvc
@ComponentScan(basePackages= {"fr.dawan"})
public class ApplicationConfig implements WebMvcConfigurer{
    public void configureViewResolvers(ViewResolverRegistry registry)
    {
        registry.jsp("/WEB-INF/jsp/", ".jsp");
    }
}
```

- **@Configuration** : indique qu'il s'agit d'une classe de configuration
- **@ComponentScan** : répertoire contenant les composants de l'application
- **configureViewResolvers** : ajout du mapping pour les JSP

Implémenter des Contrôleurs



Contrôleur Spring



Classe Java annotée **@Controller**

- Contient des méthodes liées à des mapping de requêtes
 - **@RequestMapping**
- Mappings spécialisés pour les méthodes HTTP :
 - **@GetMapping**
 - **@PostMapping**
 - **@PutMapping**
 - **@DeleteMapping**
 - **@PatchMapping**

Contrôleur Spring

```
@Controller
public class WelcomeController {

    @RequestMapping("/home")
    public String home() {
        return "home";
    }

    @GetMapping("/greeting")
    public String greetingByGet() {
        return "greetingGet";
    }

    @PostMapping("/greeting")
    public String greetingByPost() {
        return "greeting";
    }
}
```

Model, ModelMap, ModelAndView



- **Model**

Model permet de fournir des attributs utilisés dans la vues

```
@GetMapping("/testmodel")
public String testModel(Model model) {
    model.addAttribute("message", "helloworld");
    return "viewPage";
}
```

- **ModelMap**

ModelMap permet de transmettre une collection de valeurs et de les traiter comme si elles se trouvaient dans une Map

```
@GetMapping("/testmodelmap")
public String testModelMap(ModelMap map) {
    map.addAttribute("welcomeMessage", "welcome");
    map.addAttribute("message", "helloworld");
    return "viewPage";
}
```

Model, ModelMap, ModelAndView



- **ModelAndView**

ModelAndView permet de transmettre la vue et les attributs avec un seul return

```
@GetMapping("/testmodelandview")
public ModelAndView testWithModelAndView() {
    ModelAndView modelAndView = new ModelAndView("viewPage");
    modelAndView.addObject("message", "helloworld");
    return modelAndView;
}
```

RequestMapping



Peut être appliqué sur une méthode ou un contrôleur

- URL :
 - `@RequestMapping("/users")`
 - `@RequestMapping("/users", "/clients")`
- URI templates :
 - `@RequestMapping("/users/{userId}")`
 - `@RequestMapping("/users/{userId:[0-9]++}")`
- Méthodes HTTP :
 - `@RequestMapping(method={RequestMethod.GET})`
 - `@RequestMapping(method={RequestMethod.GET, ...})`

RequestMapping



- Paramètres :
 - `@RequestMapping(params="id=8")`
 - `@RequestMapping(params={"id=8", "name=DOE"})`
 - `@RequestMapping(params="id")`
- Entêtes :
 - `@RequestMapping(headers="host=127.0.0.1")`
- Consommation/production :
 - `@RequestMapping(consumes=MediaType.APPLICATION_JSON_VALUE)`
 - `@RequestMapping(produces=MediaType.APPLICATION_JSON_VALUE)`

PathVariable

Identifier un élément de l'URI

- Variable nommée

```
@RequestMapping("/users/{id}")  
public String handleRequest (@PathVariable("id") String userId, Model map)  
{  
    return "my-page";  
}
```

- Variable avec nommage implicite

```
@RequestMapping("/users/{id}")  
public String handleRequest (@PathVariable String id, Model map) {  
    return "my-page";  
}
```

- Variables multiples

```
@RequestMapping("/users/{id}/adress/{adrId}")  
public String handleRequest (@PathVariable("id") String userId,  
                             @PathVariable("adrId") String adrId, Model map) {  
    return "my-page";  
}
```


PathVariable

- Variables multiples dans une map

```
@RequestMapping("/users/{id}/adress/{adrId}")  
public String handleRequest (@PathVariable Map<String, String> vMap, Model m){  
    return "my-page";  
}
```

- Variables ambiguës

```
@RequestMapping("/users/{id}")  
public String handleRequest(@PathVariable("id") String userId, Model m){  
    return "my-page";  
}  
  
@RequestMapping("/users/{name}")  
public String handleRequest2 (@PathVariable("name") String userName, Model m) {  
    return "my-page";  
}
```

↳ **Solution** : introduire des expressions régulières

```
@RequestMapping("/users/{id:[0-9]+}")  
public String handleRequest(@PathVariable("id") String userId, Model m{...}  
@RequestMapping("/users/{name:[A-Za-z]+}")  
public String handleRequest2 (@PathVariable("name") String uName, Model m){ }
```

Identifier un paramètre de l'URL

- Paramètre nommé

```
@RequestMapping
public String employeeByDept(@RequestParam("dept") String deptName, Model m){
    return "my-page";
}
```

- Paramètre avec nommage implicite

```
@RequestMapping
public String employeeByDept (@RequestParam String dept, Model m) {
    return "my-page";
}
```

- Paramètres multiples

```
@RequestMapping
public String EmployeeByDept (@RequestParam("dept") String deptName,
                             @RequestParam("state") String stateCode, Model m) {
    return "my-page";
}
```

RequestParam



- Paramètres multiples dans une map

```
@RequestMapping()  
public String handleRequest (@RequestParam Map<String, String> paramsMap,  
                             Model m) {  
    return "my-page";  
}
```

- Paramètres ambigus : params dans le RequestMapping

```
@RequestMapping(params = "dept")  
public String handleEmployeeRequestByDept(@RequestParam("dept")  
                                           String deptName, Model map) {  
    return "my-page";  
}  
  
@RequestMapping(params = "state")  
public String handleEmployeesRequestByArea (@RequestParam("state")  
                                             String state, Model map) {  
    return "my-page";  
}
```

RequestParam



- Paramètre requis, valeur par défaut

```
@RequestMapping("/users")
public String handleRequest(@RequestParam(value = "project", defaultValue="kara")
                           String projectName, Model model){
    return "my-page" ;
}
```

- Conversion implicite

```
@RequestMapping()
public String handleRequest (@RequestParam("nbitems") int nbItems, Model model) {
    return "my-page";
}
```

- Conversion de format de dates

```
@RequestMapping()
public String handleRequest (@PathVariable("id") String employeeId,
                             @DateTimeFormat(iso = DateTimeFormat.ISO.DATE)
                             @RequestParam("startDate") LocalDate startDate,
                             Model model) {
    return "my-page";
}
```

RequestHeader



- Propose les mêmes fonctionnalités que RequestParam
 - Entête nommé
 - Entête avec nommage implicite
 - Entêtes multiples
 - Entêtes stockés dans une map
 - Entête requis, valeur par défaut
 - Conversion de type
- + Récupération de tous les entêtes :

```
@RequestMapping
public String allHeaders(@RequestHeader HttpHeaders httpHeaders, Model m) {
    return "my-page";
}
```

Lier des beans

Objectif : pouvoir mettre des types complexes dans les signatures des méthodes du contrôleur

- Création d'un JavaBean

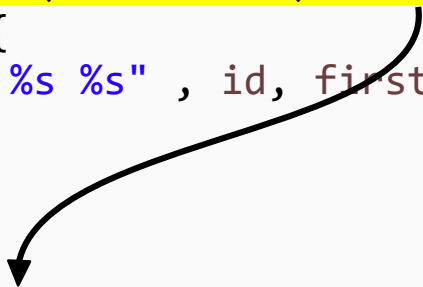
```
public class User implements Serializable{  
    private int id;  
    private String firstName;  
    private String lastName;  
    public int getId() {  
        return id;  
    }  
    public void setId(int id) {  
        this.id = id;  
    }  
    ...  
}
```

Lier des beans

- Utiliser les paramètres de la requête

```
@Controller
@RequestMapping("users")
public class UserController {
    @RequestMapping
    public String handleTradeRequest(
        @RequestParam("id") String id,
        @RequestParam("firstName") String firstName,
        @RequestParam("lastName") String lastName,
        Model map) {
        String msg = String.format("User: %s %s %s" , id, firstName, lastName);
        return "my-page";
    }

    @RequestMapping
    public String handleTradeRequest (User user, Model map) {
        String msg = String.format("User: %s %s %s", user.getId(),
            user.getFirstName(), user.getLastName());
        return "my-page";
    }
}
```

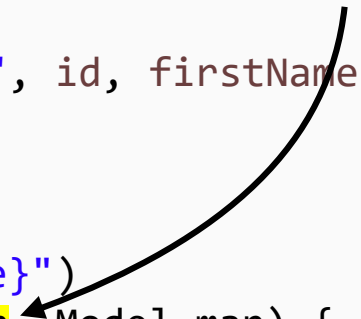


Lier des beans

- Utiliser les composants de l'URL

```
@Controller
@RequestMapping("users")
public class UserController {
    @RequestMapping("{id}/{lastName}/{firstName}")
    public String handleTradeRequest(@PathVariable("id") String id,
                                     @PathVariable("firstName") String firstName,
                                     @PathVariable("lastName") String lastName,
                                     Model map) {
        String msg = String.format("User: %s %s %s", id, firstName, lastName);
        return "my-page";
    }

    @RequestMapping("{id}/{lastName}/{firstName}")
    public String handleTradeRequest (User user, Model map) {
        String msg = String.format("User: %s %s %s", user.getId(),
                                    user.getFirstName(), user.getLastName());
        return "my-page";
    }
}
```



Vue: Redirection

- **Prefixe: redirect**

- Répond avec un code de retour 302
- Le navigateur fera une nouvelle requête vers la nouvelle URL

```
public String redirectPrefix() {  
    // ...  
    return "redirect:/redirectedUrl";  
}
```

- **Prefixe : forward**

- se passe entièrement côté serveur
- l'URL ne changera pas dans le navigateur

```
public String fowardPrefix() {  
    // ...  
    return "forward:/redirectedUrl";  
}
```

ModelAttribute



- Permet d'annoter une méthode qui retourne la valeur d'une entrée du modèle

```
@Controller
public class WelcomeController {
    @GetMapping("/greeting")
    public String greetingByPost(Model model) {
        model.addAttribute("titre", "Test de vue");
        model.addAttribute("description", "tester le passage d'infos du
                                controleur");

        return "default";
    }
    @ModelAttribute("current_time")
    public LocalDateTime getCurrentTime() {
        return LocalDateTime.now();
    }
    @ModelAttribute("user")
    public void getRequestingUser(@RequestParam(value="name", required=false)
                                String userName, Model model) {
        model.addAttribute("user", userName==null?"John DOE":userName);
    }
}
```

ModelAttribute

- Utiliser en paramètre d'une méthode du contrôleur

```
@Controller
public class WelcomeController {
    @GetMapping("/greeting")
    public String greetingByPost(@ModelAttribute("user") User user, Model m) {
        model.addAttribute("titre", "Test de vue");
        model.addAttribute("description", "pour tester le passage d'infos du
                                controleur");

        if (user.getLastName().equals("DOE")) {
            return "default";
        } else {
            return "greeting";
        }
    }

    @ModelAttribute("user")
    public User getRequestingUser(@RequestParam(value="name", required=false)
                                   String userName, Model model) {
        User u = new User();
        u.setLastName(userName==null?"DOE":userName);
        return u;
    }
}
```

Flash Attribute

- Les flash attributes permettent de passer des attributs via une redirection

```
@PostMapping
public String sourceFAttr(..., RedirectAttributes rAttr){
    // ...
    int value = 42;
    rAttr.addFlashAttribute("nomAttribut", value);
    return "redirect:url";
}

// ...

@GetMapping()
public void cibleFAttr(@ModelAttribute("nomAttribut"), int value){
    // ...
}
```

Intercepteur



- **3 méthodes :**

preHandle avant la méthode du contrôleur

postHandle entre la méthode du contrôleur et la vue

afterCompletion une fois que la vue est rendue

```
public class MyCounterInterceptor extends HandlerInterceptorAdapter {  
    private AtomicInteger counter = new AtomicInteger(0);  
  
    @Override  
    public boolean preHandle(HttpServletRequest request, HttpServletResponse  
                             response, Object handler) throws Exception {  
        HttpSession session = request.getSession(true);  
        request.setAttribute("visitorCounter", counter.incrementAndGet());  
        return true;  
    }  
}
```

- Accès à la session et à la requête

Intercepteur



- **Contrôleur : en paramètre de méthode**

- Accès à la session

- ```
@SessionAttribute(name = "sessionStartTime")
```

- Accès à la requête

- ```
@RequestAttribute(name = "sessionStartTime")
```

Gestion des exceptions



Annotation d'une méthode **@ExceptionHandler** dans le contrôleur :

- Cette méthode sera invoquée lors du lancement d'une exception

```
@ExceptionHandler  
public String handleError() {...}
```

- Possibilité de spécifier l'exception à gérer

```
@ExceptionHandler({SQLException.class,DataAccessException.class})  
public String databaseError() {...}
```

- Cascade de gestionnaires d'exception
- Généraliser la gestion des exceptions dans une classe annotée **@ControllerAdvice**



Plus d'informations sur <http://www.dawan.fr>

**Contactez notre service commercial au
09.72.37.73.73 (prix d'un appel local)**