

Java Enterprise Edition (JEE)

Chapitre 04 : Spring & Thymeleaf

Sébastien Chèvre

Plan du module – chapitres

Java Entreprise Edition - JEE

- Chapitre 01 : Introduction et motivations
- Chapitre 02 : Machine Virtuelle Java
- Chapitre 03 : Java EE
- **Chapitre 04 : *Spring & Thymeleaf***
- Chapitre 05 : *ORM & DAO avec Spring Data & JPA*
- Chapitre 06 : Sécurité avec *Spring Security*
- Chapitre 07 : Tests avec *Spring Testing*
- Chapitre 08 : SOA & Micro-services avec *Spring*
- Chapitre 09 : *JMS (Java Messaging Service)*

- Projet

Objectifs du chapitre

Chapitre 04 : Spring & Thymeleaf

A la fin de ce chapitre, les étudiants peuvent:

- Comprendre les principaux «building blocks» d'une application **Spring**
- Comprendre les éléments de bases de l'écosystème Spring:
 - Spring MVC
 - Spring Boot
- Implémenter une application simple **Spring**
- Comprendre les différences entre la plateforme **JEE** et **Spring**

Plan du chapitre

Chapitre 04 : Spring & Thymeleaf

- **Ecosystème Spring**
- **Spring framework**
 - IOC
 - AOP
 - Beans et composants
 - Configuration
- **Spring MVC**
 - Architecture et principe
 - Contexte et Dispatch Servlet
 - Contrôleur et View Resolver
 - Configuration XML et JSP
- **Spring Boot**
 - Spring Boot Starter
- **Maven**
 - Concepts de base
- **Spring Boot MVC**
 - Thymeleaf
- **Mise en pratique**

JEE

Sémantique des diapositives



À savoir théoriquement (TE)



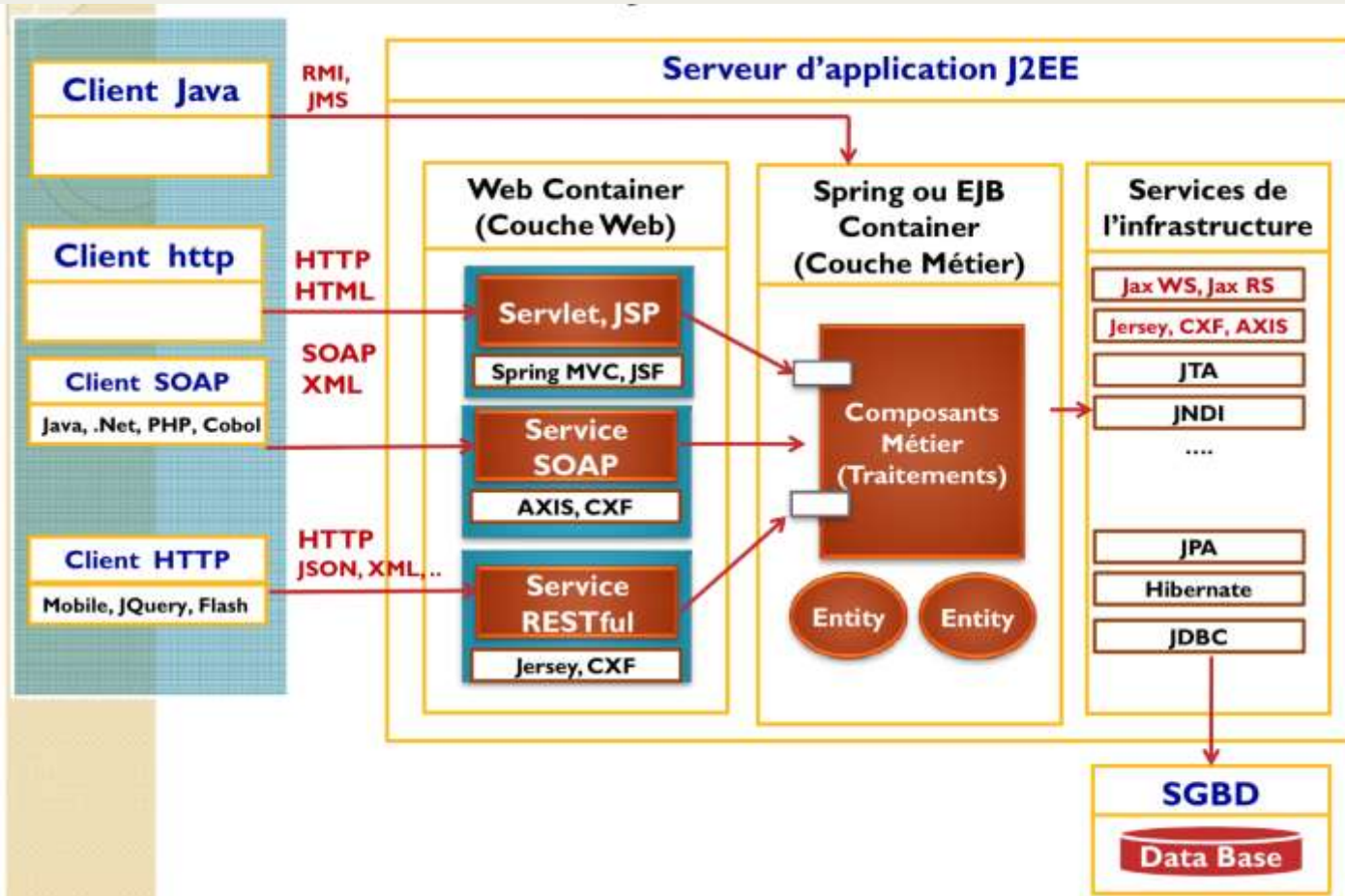
Sensibilisation, illustration de concepts, exemples



Aspects pratiques

Java Entreprise Edition

Architecture de référence

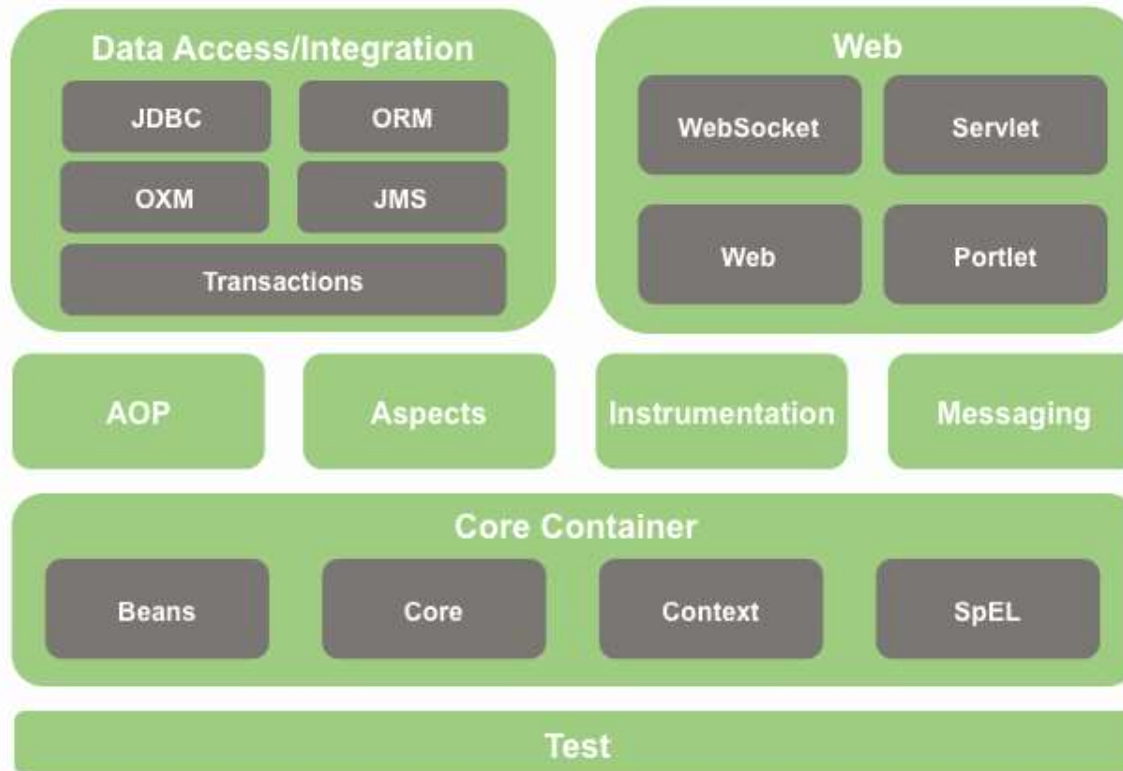


Spring Framework

Vue d'ensemble des modules

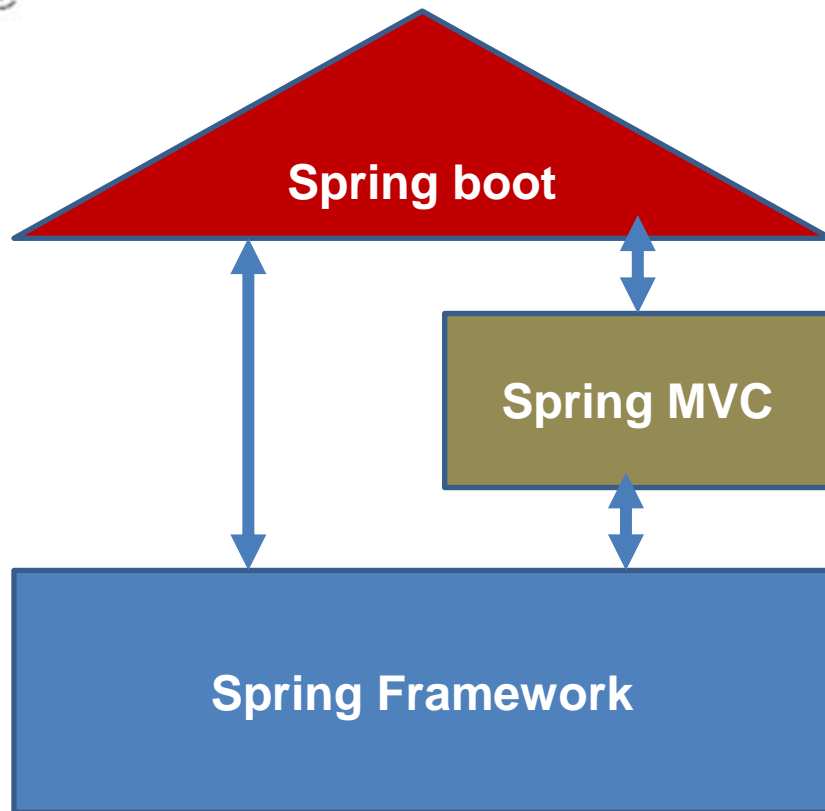


Spring Framework Runtime



Spring - Spring MVC - Spring Boot

Vue d'ensemble



- **Auto-configuration:** le framework est capable de s'autoconfigurer en fonction des dépendances présentes (jar)
- **Spring Boot Starter Projects :** Ensemble de bibliothèques basées sur des patterns connus, permettant de simplifier la gestion des dépendances de l'application

Le framework **Spring MVC** fournit une manière de créer des applications web. Il définit des concepts simples tels que **DispatchServlet**, **ModelAndView** et **ViewResolver**, qui permettent simplement et rapidement de développer des applications web d'entreprises.

Brique de base de Spring. Fournit la gestion de l'injection de dépendance et les contextes de bases

Spring Framework



<https://spring.io/>

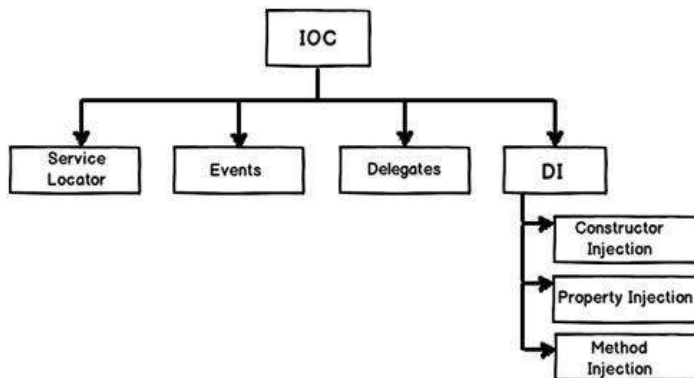
Spring

Concepts de bases



- **IoC – Inversion de contrôle**

- Pattern d'architecture générique définissant le fait que le flux d'exécution de l'application *n'est plus sous le contrôle directe de l'application mais géré par le framework*
- Avec Spring, le pattern IoC est implémenté via *l'injection de dépendance (DI)*



Spring

Concepts de bases



- **AOP – Aspect Oriented Programming**

- Permet une séparation des concepts métiers des concepts cross-plateforme, comme:
 - *Logging*
 - *Transaction*
 - *Gestion de cache*
 - *Sécurité*



Spring

Concepts de bases



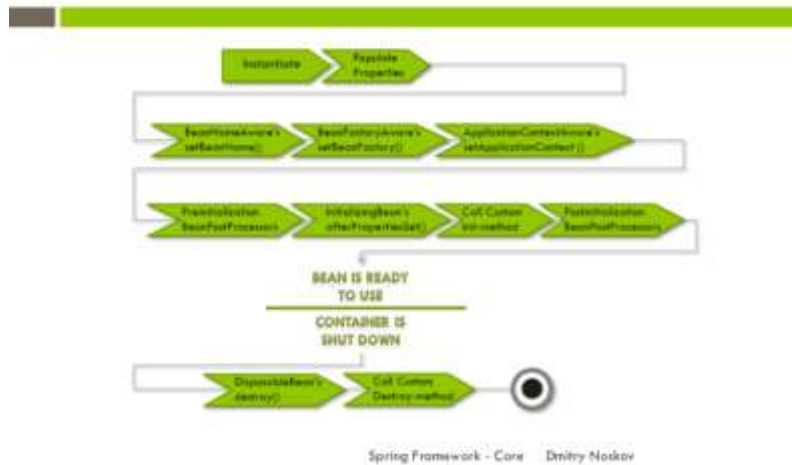
- **Alternative «légère» à JEE**
 - Solution «légère» pour la construction d'application d'entreprise
 - Basée sur les POJO (Plain Old Java Object) à la place des EJB
 - Il ne nécessite pas de serveur d'application
 - Peut être déployée en tant qu'archive *war*
 - Peut embarquée le container applicatif (*Tomcat, Jetty, Undertow*)

Spring Bean



- Un bean Spring est un objet géré par Spring
 - Initialisation, cycle de vie géré par Spring
 - Configuré par méta donnée (xml ou annotation)
 - Possède un constructeur par défaut

Bean lifecycle

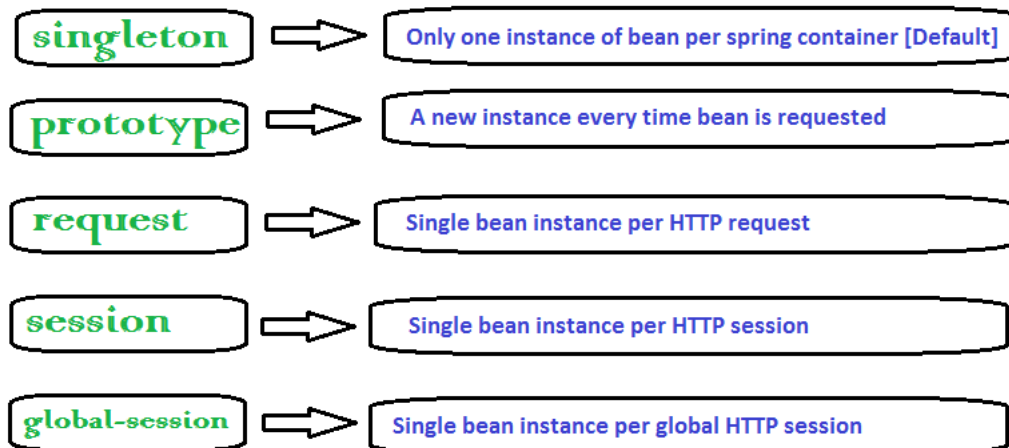


Spring Bean



- Un bean Spring possède une visibilité définie
 - Par défaut: singleton
 - Peut être configurée

Spring Bean Scopes



Spring

Component vs Bean



- Définition d'un bean (via xml)

```
<!-- un bean avec visibilité par défaut (singleton) -->
<bean id="firstBean" class="ch.hearc.jee.bean.FirstBean"></bean>

<!-- un bean avec visibilité par prototype -->
<bean id="firstBean" class="ch.hearc.jee.bean.FirstBean" scope="prototype"></bean>
```

- Définition d'un bean (via annotation)

```
@Bean
public FirstBean firstBean() {
    return new FirstBean();
}

@Bean
@Scope("prototype")
public FirstBean firstBean() {
    return new FirstBean();
}
```

Spring

Component vs Bean



- Définition d'un bean (via **@Component**)

```
@Component  
public class FirstBean {  
    ...  
}
```

- L'annotation **@Component** s'utilise sur la classe (contrairement à **@Bean** qui s'utilise sur une méthode)
- L'annotation **@Bean** doit s'utiliser sur une classe ayant été annoté **@Configuration**
- Plusieurs type de component existe:
 - **@Controller**, **@RestController**
 - **@Service**
 - **@Repository**

Spring @Configuration



- Permet de définir une classe Java comme source de configuration (à la place de xml)
- L'annotation **@Bean** doit se trouver dans une classe annoté **@Configuration**

```
@Configuration
public class AppConfig {

    @Bean
    public FirestBean() {
        return new FirstBean();
    }
}
```

Spring @Autowired



- Permet de définir l'injection de dépendance en mode Java (non xml)
- Injecte des beans standard
- Injecte une implémentation d'interface si présente

```
public class App {  
  
    @Autowired  
    FirstBean a;  
}
```

Spring Core

Synthèse annotations



Annotation	Fonction	Elément
@Component	Définit un bean	Classe
@Bean	Définit un bean	Méthode
@Controller @RestController	Définit un contrôleur optimisé. @RestController est pour simplifier la gestion des API sans vue (<i>xml, json</i>)	Classe
@Service @Repository	Composants avec sémantique service et repository	Classe
@Autowired	Injection d'objets (bean) automatique	Champs Méthode
@Configuration	Définit une classe utilisée servant à configurer des bean.	Classe
@ComponentScan	Spécifie les packages à scanner contenant les composants. Utiliser avec @Configuration	Classe

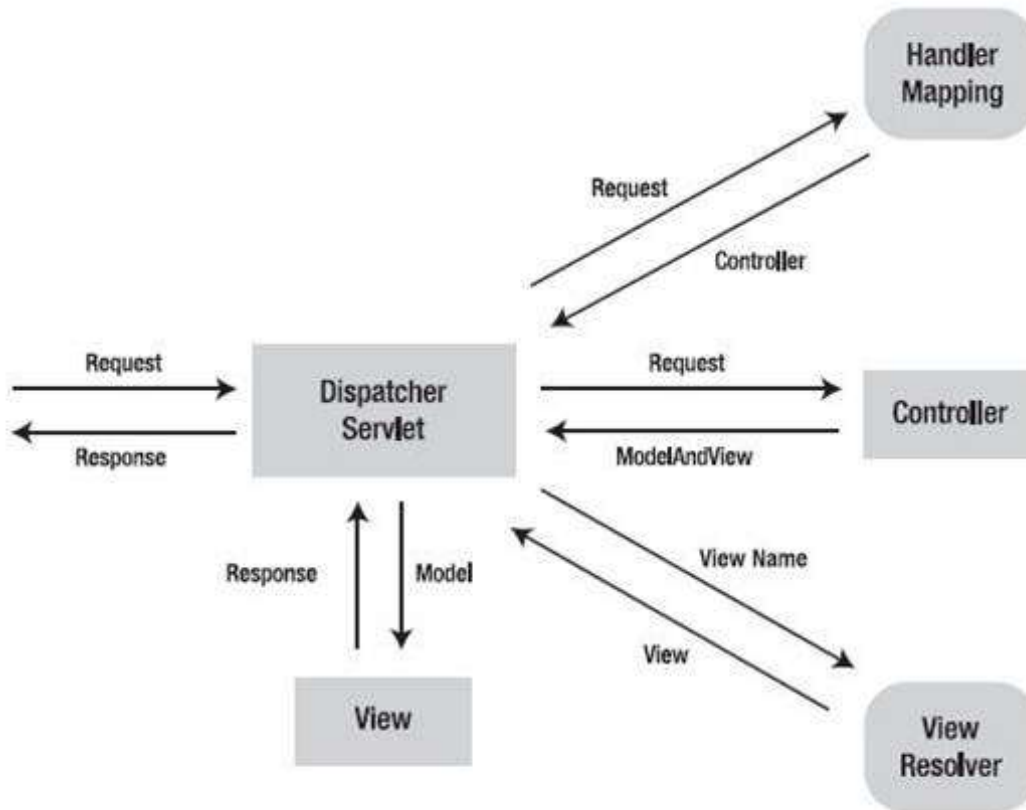
Bleu : @component et spécialisation

Spring MVC

<https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html>

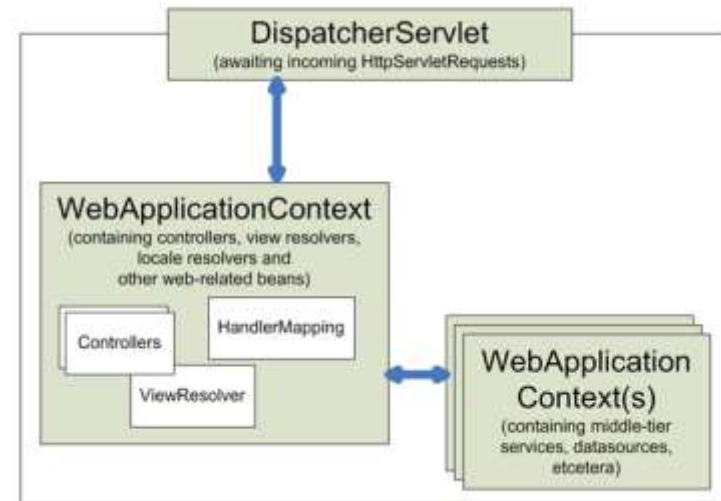
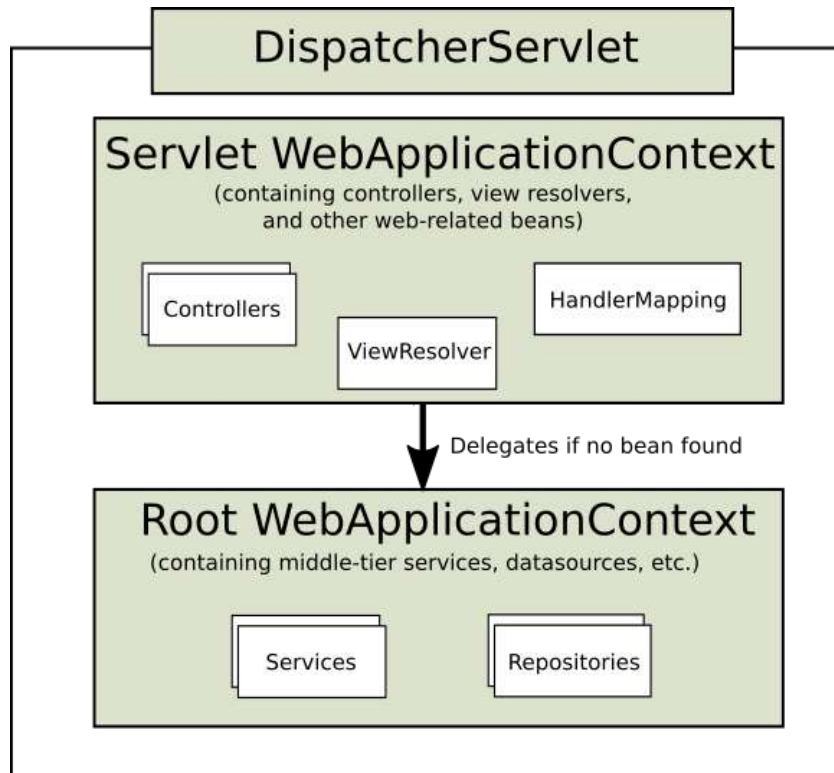


Spring MVC Architecture & Flux



Spring MVC

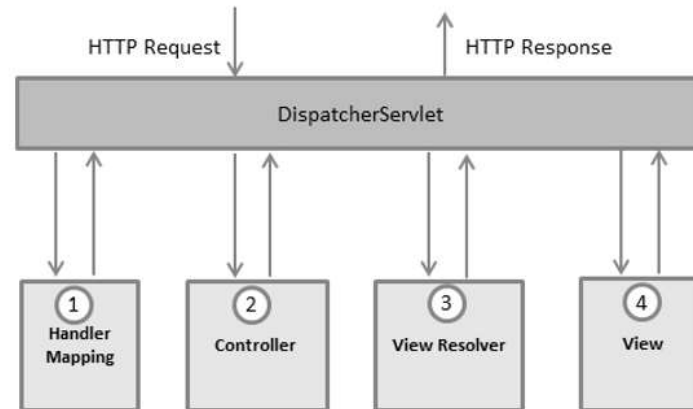
Contexte web



Spring MVC DispatcherServlet



- La «**dispatchServlet**» traite toutes les requêtes et réponses http
- Une fois la requête reçue, consultation du «**handlerMapping**» afin d'appeler le contrôleur approprié ①
- Le contrôleur traite la requête, appelle la méthode appropriée et construit le modèle. Il retourne le nom de la page au «**disptachServlet**» ②
- Le «**disptachServlet**» fait appel au «**viewResolver**» afin de construire la vue définie par la requête ③
- Une fois la vue finalisée, le «dispatchServlet» passe le modèle à la vue qui est finalement envoyée au navigateur ④



Spring MVC Controller



- Le contrôleur Spring:
 - Intercepte les requêtes entrantes
 - Convertit les données de la requête (corps, paramètres) en structure interne
 - Envoie les données au modèle pour la suite du traitement
 - Le «**dispatchServlet**» peut être considéré comme «**front controller**»

Spring MVC Controller



```
@Controller
@RequestMapping(value="/test")           //écoute sur le path /test
public class BasicController {

    @GetMapping                           //méthode http get
    public ModelAndView getData() {

        ModelAndView mw = new ModelAndView();
        mw.setViewName("test");           //recherche une vue test (test.jsp)
        mw.getModel().put("data", "Hello world"); //ajoute des données au modèle

        return mw;
    }
}
```

Spring MVC ViewResolver



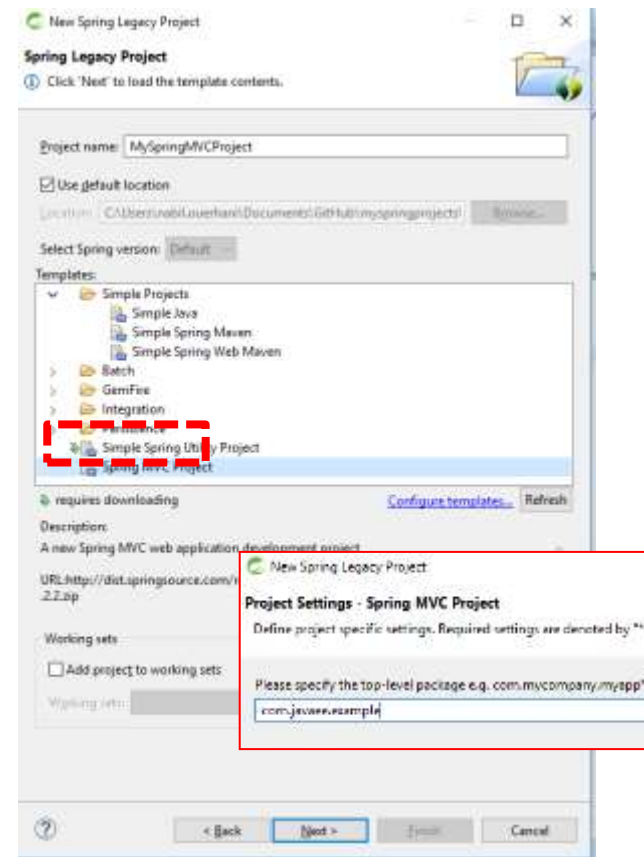
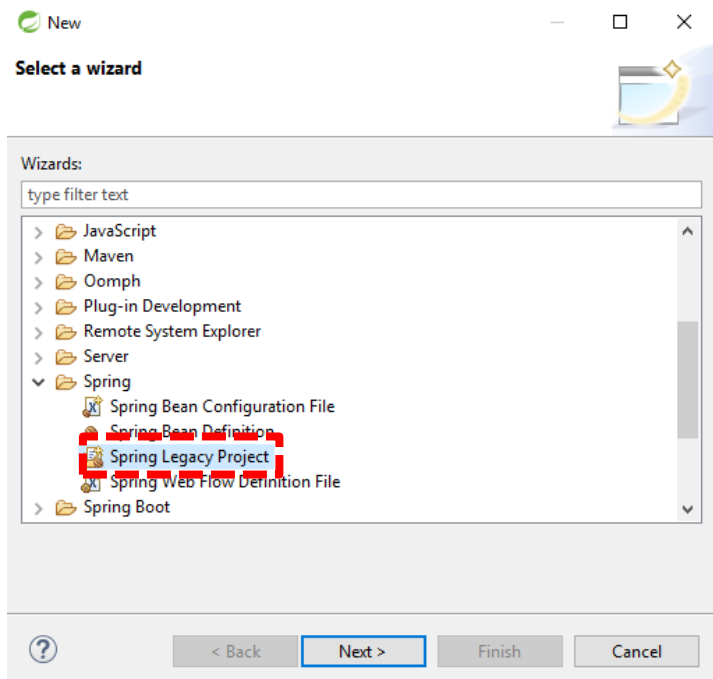
- Le ViewResolver:
 - Génère les vue finales basées sur les modèles
 - Fonctionne avec des JSP mais aussi des moteurs de templates (*Thymeleaf*, *Mustache*, etc...)

Spring Tool Suite

Premier projet Spring



- Créer un nouveau projet Spring MVC



Spring MVC HomeController



```

1 package com.javaee.examples;
2
3 import java.text.DateFormat;
13
14 /**
15  * Handles requests for the application home page.
16  */
17 @Controller
18
19 public class HomeController {
20
21     private static final Logger logger = LoggerFactory.getLogger(HomeController.class);
22
23     /**
24      * Simply selects the home view to render by returning its name.
25      */
26     @RequestMapping(value = "/", method = RequestMethod.GET)
27
28     public String home(Locale locale, Model model) {
29         logger.info("Welcome home! The client locale is {}.", locale);
30
31         Date date = new Date();
32         DateFormat dateFormat = DateFormat.getDateInstance(DateFormat.LONG, DateFormat.LONG,
33
34         String formattedDate = dateFormat.format(date);
35
36         model.addAttribute("serverTime", formattedDate );
37
38         return "home";
39     }
40

```

Spring MVC web.xml



```
<!-- Processes application requests -->
<servlet>
  <servlet-name>appServlet</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>appServlet</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

Spring MVC

ServletContext.xml



```
<!-- Resolves views selected for rendering by @Controllers to .jsp resources in the /WEB-INF/views directory -->
<beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <beans:property name="prefix" value="/WEB-INF/views/" />
    <beans:property name="suffix" value=".jsp" />
</beans:bean>
```

```
<context:component-scan base-package="com.javaee.examples" />
```

Spring MVC

Home.jsp



```
1 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
2 <%@ page session="false" %>
3 <html>
4 <head>
5     <title>Home</title>
6 </head>
7 <body>
8 <h1>
9     Hello world!
10 </h1>
11
12 <P> The time on the server is ${serverTime} </P>
13 </body>
14 </html>
15
```

Spring Boot



<https://spring.io/projects/spring-boot>

Spring Boot

A quoi ça sert



- **Starter project**

- Simplification de la gestion des dépendances pour des patterns connus (web, integration, messaging, etc...)

- **Auto-configuration**

- Spring Boot permet de configurer automatiquement votre projet / application en fonction des dépendances (jar) que vous avez spécifier
- Exemple: si vous spécifiez (dans le pom.xml) que vous voulez une application qui utilise *mySQL*, *Spring Boot* configure automatiquement votre projet afin d'inclure les jars nécessaire
- L'utilisation de l'annotation **@SpringBootApplication** permet d'activer l'auto-configuration

- **Artefact**

- un jar «self-contained» qui contient tout (application, Web container, etc).

Spring Boot

Spring Starter project (STS)



New Spring Starter Project

Service URL:

Name:

☒ Use default location

Location:

Type: Packaging:

Java Version: Language:

Group:

Artifact:

Version:

Description:

Package:

Working sets

☐ Add project to working sets

Working sets:

New Spring Starter Project Dependencies

Spring Boot Version:

Available:

Selected:

- Cloud AWS
- Cloud Circuit Breaker
- Cloud Config
- Cloud Contract
- Cloud Core
- Cloud Discovery
- Cloud Messaging
- Cloud Routing
- Cloud Support
- Cloud Tracing
- Core
- I/O
- Integration
- NoSQL
- Ops
- Pivotal Cloud Foundry
- SQL
- Spring Cloud GCP
- Template Engines
- Web

Spring Initializr

<https://start.spring.io/>



https://start.spring.io

Paramètres Favorites Inter - Tessera del Tif

SPRING INITIALIZR

bootstrap your application now

Generate a Maven Project with Java and Spring Boot 2.0.5

Project Metadata

Artifact coordinates

Group

Artifact

Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

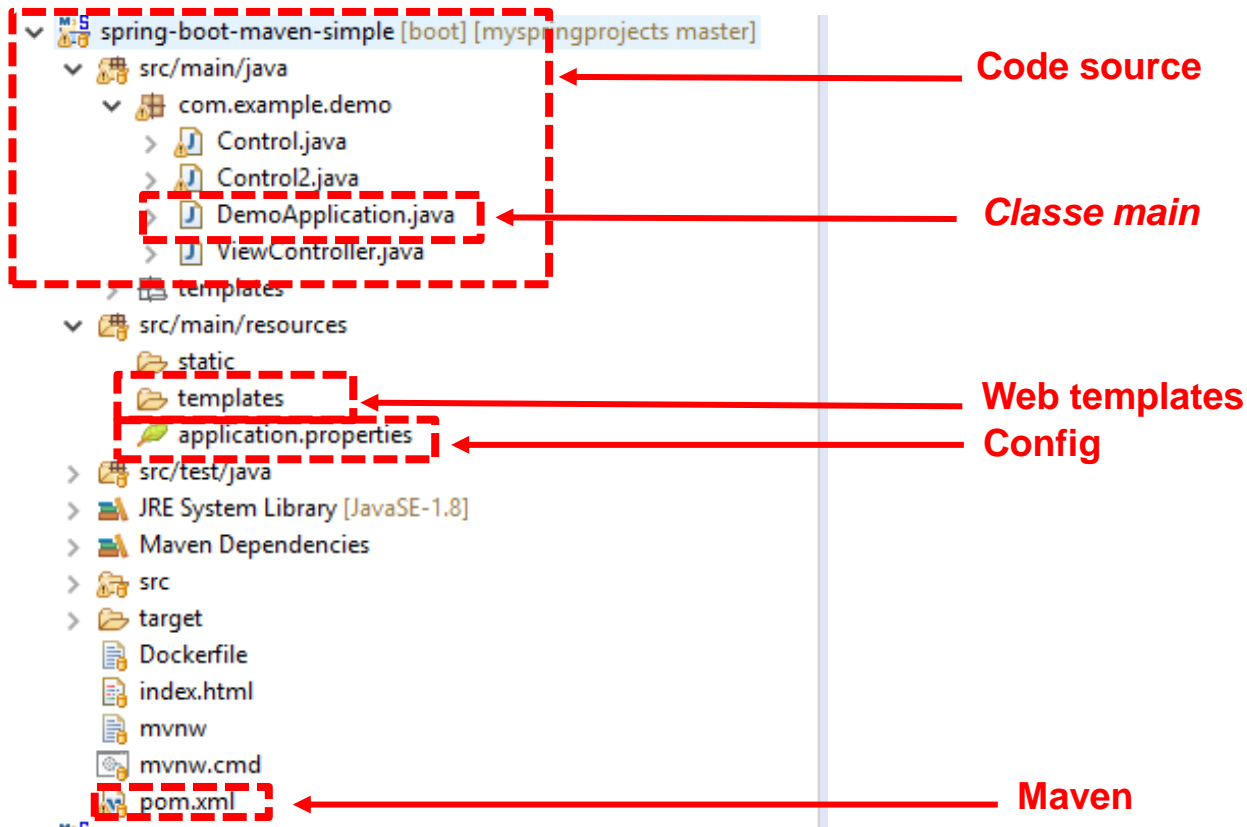
Selected Dependencies

Generate Project alt + ↵

Don't know what to look for? Want more options? [Switch to the full version.](#)

Spring Boot

Structure de projet



Spring Boot

Classe Main



```
1 package com.example.demo;  
2  
3 import org.springframework.boot.SpringApplication;  
4 import org.springframework.boot.autoconfigure.SpringBootApplication;  
5  
6 @SpringBootApplication  
7 public class DemoApplication {  
8  
9     public static void main(String[] args) {  
10         SpringApplication.run(DemoApplication.class, args);  
11     }  
12 }  
13
```

Spring Starter project pom.xml



```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apac
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>demo</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <name>demo</name>
  <description>Demo project for Spring Boot</description>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.3.RELEASE</version>
    <relativePath /> <!-- lookup parent from repository -->
  </parent>
```

La magie !

Maven

C'est quoi?



Apache Maven Project
<http://maven.apache.org/>



- Maven est un gestionnaire de projet
- Sous license Apache
- Permet de gérer:
 - Les dépendances
 - La construction
 - Les tests
 - La documentation
 - Et le reste....

Maven

Project Object Model



Apache Maven Project
<http://maven.apache.org/>

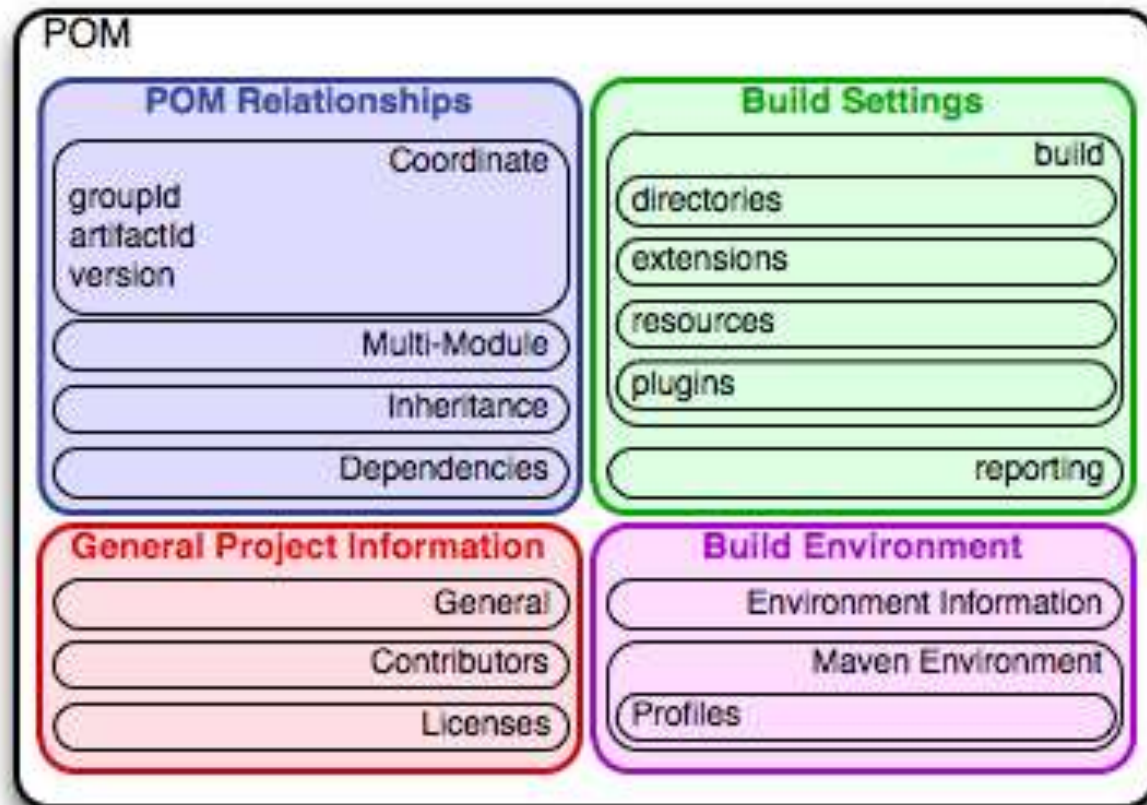


- Basé sur le concept de POM (*Project Object Model*)
 - Fichier **pom.xml**, c'est tout!
- Basé sur des cycles de vie (3 par défaut), eux même structurés en phases

Cycle de vie	Description
clean	Nettoyage des répertoires de build
default	Compilation, test, packaging, déploiement
site	Reporting, documentation

Maven

Project Object Model



Maven

clean lifecycle



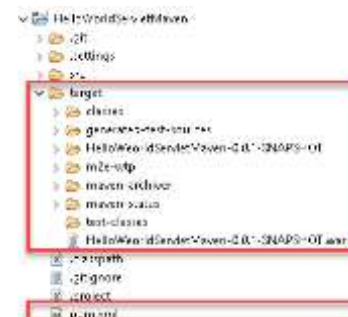
- Cycle permettant le nettoyage du répertoire de build
 - Par défaut: **target**

Phase

pre-clean

clean

post-clean



Commande

mvn clean

Exécution du cycle de vie clean (complet)

mvn clean:clean

Exécution de la phase clean du cycle de vie clean

Maven

default lifecycle

- Cycle gérant les aspects de construction jusqu'au déploiement



Phases

validate

generate-sources

process-sources

generate-resources

process-resources

compile

generate-test-resources

process-test-resources

test-compile

test

prepare-package

package

pre-integration-test

verify

install

deploy

Maven

default lifecycle



Commande	
mvn validate	Exécution de la phase validate
mvn compile	Exécution jusqu'à la phase compile
mvn test	Exécution jusqu'à la phase test
mvn install	Exécution jusqu'à la phase install

Commande à connaître:

mvn clean install

Maven

Gestion des dépendances



- Repository central maven
 - <https://mvnrepository.com/>
- Possibilité d'implémenté des repository « on site »
 - Perso
 - Entreprise
- Tout Java Open Source

Maven

Structure de projet



Maven recommande une structure de projet standardisée.
Ce n'est pas obligatoire, mais fortement recommandé!

- **src**

- **main** → *Ressources principales*
 - **java** → *Code source java*
 - **resources** → *Ressources du projet non java*
 - **webapp** → *Ressources web (optionnel)*

- **test**

- **java**
- **resources**



Maven

Répertoire *target*



Le répertoire ***target***, est le répertoire de travail de *maven*. Tous les fichiers générés par *maven* se trouvent dans ce répertoire.



Thymeleaf



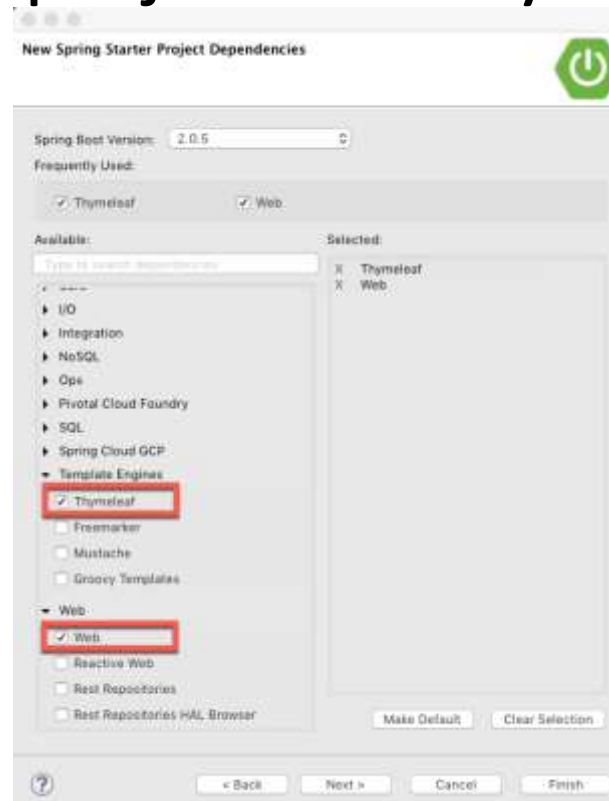
Thymeleaf

Projet Spring

Paramétrage Thymeleaf



- Création d'un projet avec Thymeleaf



Approfondissement Thymeleaf



- **Thymeleaf**
 - Moteur de template
 - Côté serveur
 - Application web et standalone
 - Séparation claire vue – code métier
 - License Apache 2.0
 - Génération xml, xhtml, html5, css, js
 - Balises et expressions

Documentation

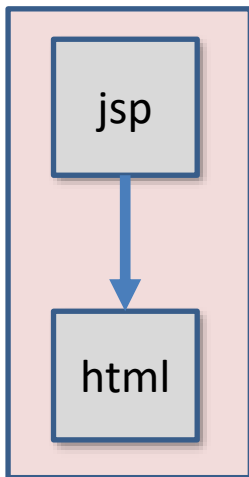
- <https://www.thymeleaf.org/documentation.html>

Génération HTML côté serveur

historique et évolution

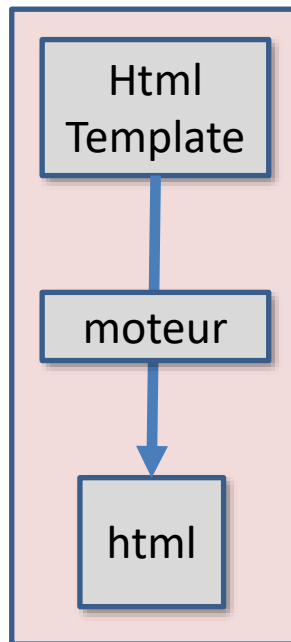


Avant



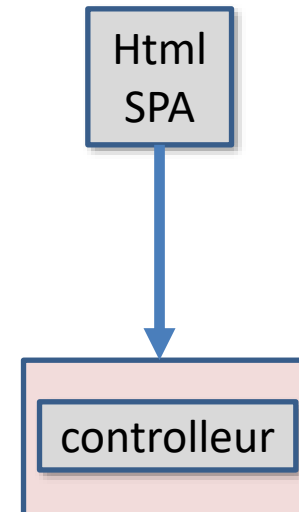
*Code Java dans la JSP.
La séparation des
responsabilités peut être
compliquée. Les fichiers
JSP ne sont pas utilisables
dans un autre contexte
que Java*

Maintenant



*Standard HTML.
Pas de code serveur
en dehors des balises
propres au moteur*

Après

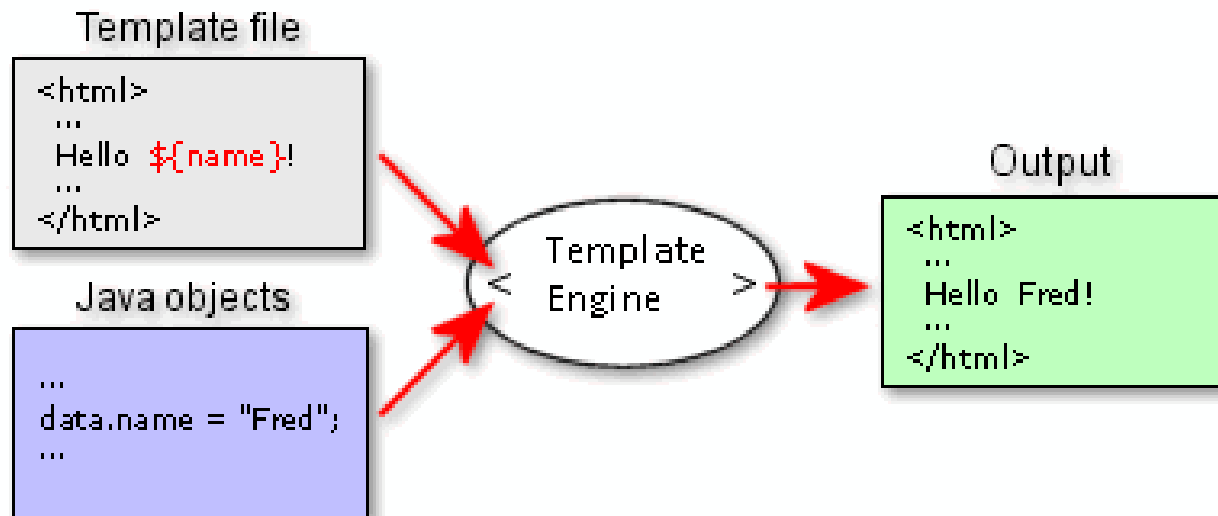


*Le serveur ne retourne plus de
vue au sens html. Une application de type
SPA (Single Page Application) va
consommer des
services fournissant des résultats sous
formes xml ou json*

Approfondissement Thymeleaf



- **Fonctionnement**



Thymeleaf namespace



```
<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
<meta charset="ISO-8859-1">  
<title>Insert title here</title>  
</head>  
<body>  
<h1>Hello from Thymeleaf template</h1>  
</body>  
</html>
```

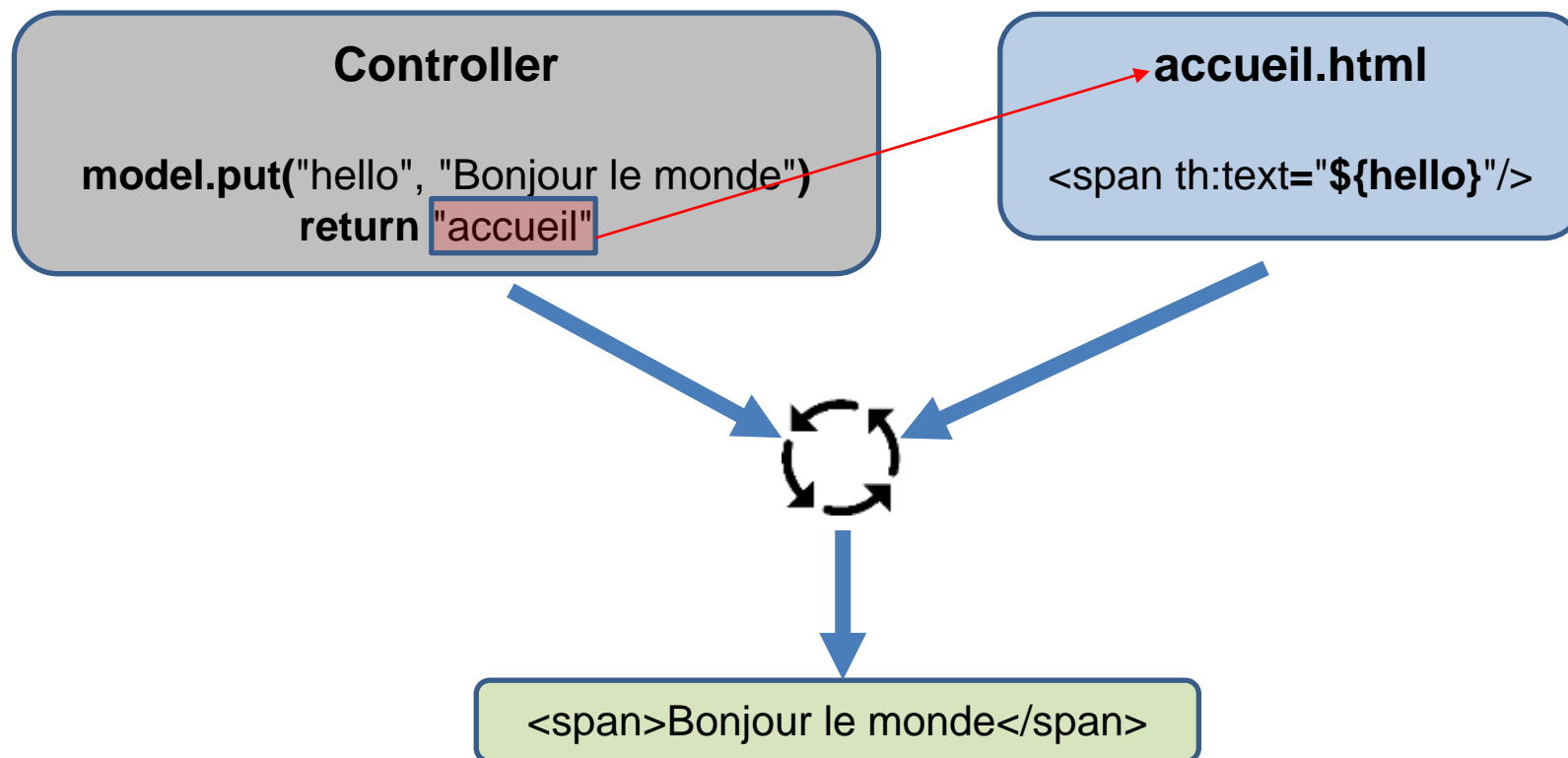
```
<html xmlns:th="http://www.thymeleaf.org">
```

Thymeleaf

Approfondissement



Fonctionnement avec Spring Boot



Approfondissement Thymeleaf syntaxe



- **Balises de bases**
 - **th:text**
Affichage de texte dans la page
 - **th:each**
Itération sur une collection
 - **th:if / th:unless**
Exécution d'instructions conditionnels
 - **th:switch**
Exécution de branchements conditionnels

Approfondissement Thymeleaf syntaxe



- **Expression de bases**

- **\$** : résolution de variable. Syntaxe Spring EL
`${etudiant.nom}` : *fournit la valeur de la propriété nom de l'instance étudiant fournit dans le modèle*
- ***** : comme \$, mais exécuter sur un objet sélectionné
`*{nom}` : *fournit la valeur de la propriété nom de l'instance étudiant fournit dans le contexte (itération, p.ex)*
- **#** : expression i18n
`#{etudiant.nom}` : *fournit la valeur de la propriété etudiant.nom fournit par la configuration de la localisation*
- **@** : expression de lien (URL). Utilisé pour générer des URLs
`@{css/accueil.css}` : *url vers une feuille de style*

Approfondissement Thymeleaf syntaxe



- Literals
 - Text literals: 'one text', 'Another one!', ...
 - Number literals: 0, 34, 3.0, 12.3, ...
 - Boolean literals: true, false
 - Null literal: null
 - Literal tokens: one, sometext, main, ...
- Text operations:
 - String concatenation: +
 - Literal substitutions: |The name is \${name}|
- Arithmetic operations:
 - Binary operators: +, -, *, /, %
 - Minus sign (unary operator): -
- Boolean operations:
 - Binary operators: and, or
 - Boolean negation (unary operator): !, not
- Comparisons and equality:
 - Comparators: >, <, >=, <= (gt, lt, ge, le)
 - Equality operators: ==, != (eq, ne)
- Conditional operators:
 - If-then: (if) ? (then)
 - If-then-else: (if) ? (then) : (else)
 - Default: (value) ?: (defaultvalue)

```
<span th:text="'Type utilisateur : ' + (${user.isAdmin()}) ? 'Administrator' :  
(${user.type} ?: 'Inconnu')"></span>
```

Thymeleaf

Approfondissement



- **th:text**

```
<p class="card-text">Variable [message]:  
  <span th:text="${message}"></span>  
</p>
```

- **th:each**

```
<tr th:each="etudiant: ${etudiants}">  
  <td th:text="${etudiant.nom}" />  
  <td th:text="${etudiant.age}" />  
</tr>
```

- **th:if / th:unless**

```
<td th:if="${etudiant.sexe} == 'M'" th:text="Homme" />  
<td th:unless="${etudiant.sexe} == 'M'" th:text="Femme" />
```

- **th:switch**

```
<td th:if="${etudiant.sexe} == 'M'" th:text="Homme" />  
<td th:unless="${etudiant.sexe} == 'M'" th:text="Femme" />
```

Thymeleaf

Approfondissement



- **Fragments**
 - **th:insert**
Insertion d'un fragment dans la balise appelante
 - **th:replace**
Remplacement de la balise appelant par un fragment

Thymeleaf

Approfondissement



- **Fragments – th:insert**

Définition d'un fragment (*fichier fragments/main.html*):

```
<div th:fragment="fragment-1">  
  <span>Fragment html...</span>  
</div>
```

Insertion:

```
<div th:insert="fragments/main.html :: fragment-1">  
</div>
```

Résultat:

```
<div>  
  <span>Fragment html...</span>  
</div>
```

Thymeleaf

Approfondissement



- **Fragments – th:replace**

Définition d'un fragment (*fichier fragments/main.html*):

```
<div th:fragment="fragment-1">  
  <span>Fragment html...</span>  
</div>
```

Remplacement:

```
<div th:replace="fragments/main.html :: fragment-1">  
</div>
```

Résultat:

```
<span>Fragment html...</span>
```

Thymeleaf

Approfondissement



- **Formulaires**
 - th:action
 - th:object
 - th:field

Thymeleaf

Formulaire exemple



```
<form action="#" th:action="@{/etudiant}" th:object="${etudiant}"
method="post">
    <input type="text" th:field="*{prenom}" id="prenom">
    <select class="form-control" id="sexe" th:field="*{sexe}">
        <option value="M">Masculin</option>
        <option value="F">Féminin</option>
    </select>
    <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

url de soumission du formulaire

Modèle envoyé au serveur

Propriété du modèle

Thymeleaf

Controlleur Spring - exemple



```
@PostMapping("/etudiant")
public String saveEtudiant(
    @Valid @ModelAttribute Etudiant etudiant,
    BindingResult errors,
    Model model) {
    if (!errors.hasErrors()) {
        dao.save(etudiant);
    }
    return ((errors.hasErrors()) ? "etudiant" : "redirect:basic");
}
```

Annotations and parameters in the code are annotated with arrows and labels:

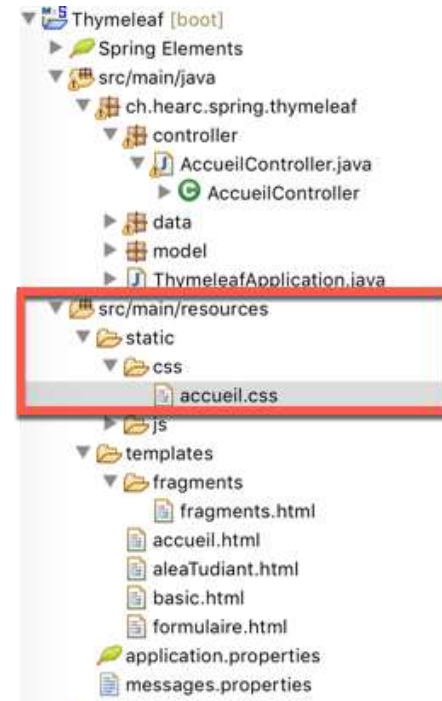
- `@Valid` → *Objet permettant de gérer les erreurs*
- `@ModelAttribute` → *Le modèle*
- `Etudiant etudiant` → *L'attribut étudiant du modèle*

Thymeleaf

Approfondissement – Styles et scripts



- **Styles et scripts**
 - CSS «standard»



```
<link rel="stylesheet" type="text/css" th:href="@{/css/accueil.css}" />
```

Thymeleaf

Approfondissement – Styles et scripts



- **Styles et scripts**

- CSS «WebJars»
 - <https://www.webjars.org/>
 - Exemple avec Bootstrap

```
<link rel="stylesheet" type="text/css"
href="webjars/bootstrap/4.1.3/css/bootstrap.min.css" />
```

Inclusion html

```
<!-- https://mvnrepository.com/artifact/org.webjars/bootstrap -->
<dependency>
  <groupId>org.webjars</groupId>
  <artifactId>bootstrap</artifactId>
  <version>4.1.3</version>
</dependency>
```

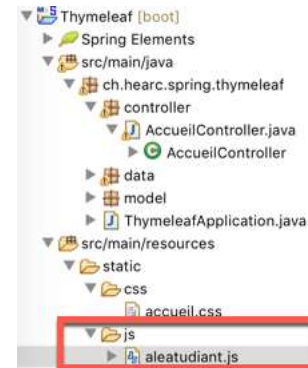
Dépendances maven → pom.xml

Thymeleaf

Approfondissement – Styles et scripts



- **Styles et scripts**
 - JS «Standards»



```
<script type="text/javascript" src="js/aleatudiant.js"></script>
```

Thymeleaf

Approfondissement – Styles et scripts



- **Styles et scripts**

- JS «WebJars»
 - <https://www.webjars.org/>
 - Exemple avec Bootstrap

```
<script type="text/javascript"  
src="webjars/bootstrap/4.1.3/js/bootstrap.min.js"></script>
```

Inclusion html

```
<!-- https://mvnrepository.com/artifact/org.webjars/bootstrap -->  
<dependency>  
  <groupId>org.webjars</groupId>  
  <artifactId>bootstrap</artifactId>  
  <version>4.1.3</version>  
</dependency>
```

Dépendances maven → pom.xml

Thymeleaf

Approfondissement – Exercices



- <http://itutorial.thymeleaf.org/>
- Exercice 1 (Beans)
- Exercice 6 (Itérations)
- Exercice 8 (Conditions)
- Exercice 12 (Formulaire)

Projet exemple GitHub

- <https://github.com/Cours-HE-ARC/thymeleaf-produit-jpa>