Université Claude Bernard Lyon 1

M1IF13 – Web avancé, Web mobile
Le framework Spring

Master 1 Informatique

Lionel Médini

Février 2020

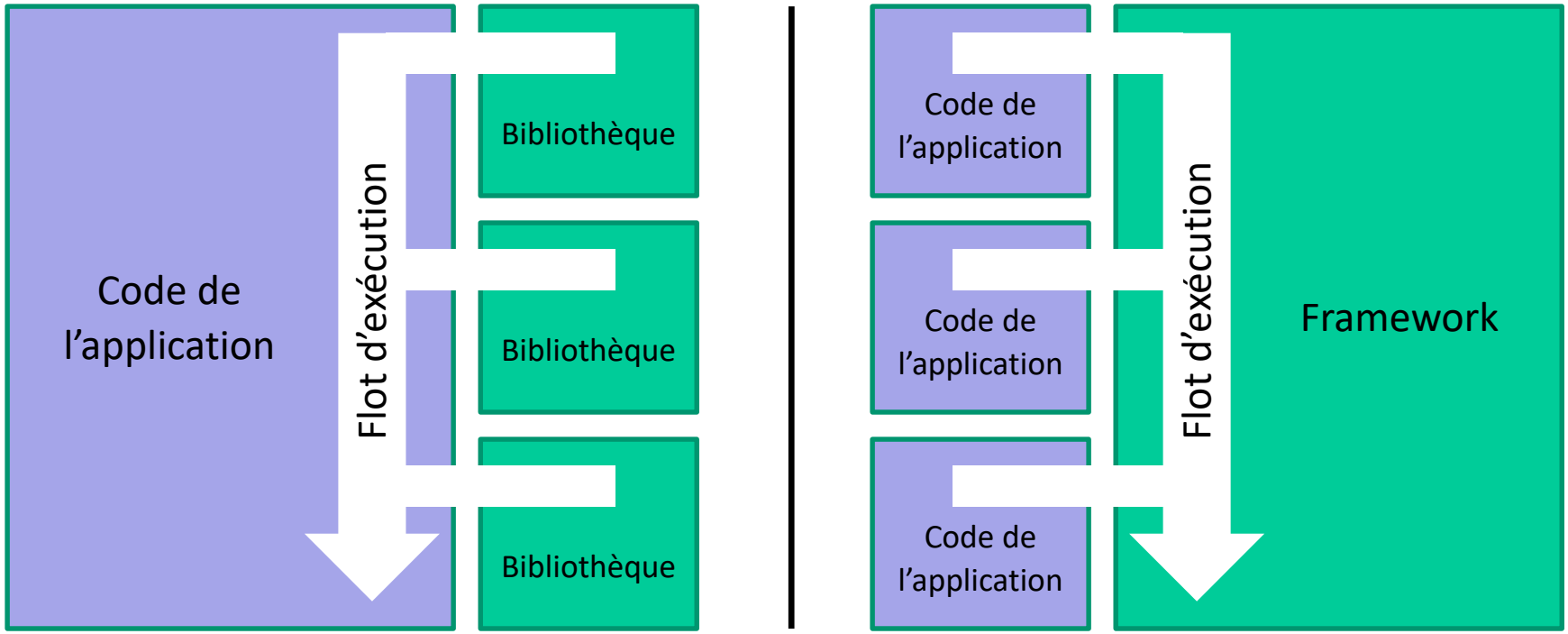
Rappel : Inversion de contrôle

- Principe général
  - Une application (Web) complexe fait nécessairement appel à du code externe pour gérer des services non métier
    - sécurité
    - persistance
    - …

  ➔ Qui contrôle le flot d'exécution d'une application ?
    - votre code
    - un des outils que vous utilisez
  - En programmation classique
    - D'où provient le main ?
  - En MVC
    - Qui dirige le contrôleur ?

Rappel : Inversion de contrôle

- Différence bibliothèque / framework



  - Remarque : dans la littérature, on trouve l'appellation « framework » pour beaucoup de choses qui n'en sont pas

Le framework Spring

- Historique
  - Juin 2003 : sortie de la première version de Spring framework
  - 2004 : création de la société SpringSource par Rod Johnson
    - publication du livre“Expert One-on-One J2EE Design and Development” qui justifie la création de Spring
  - 2006 : sortie de la V. 2 de Spring
  - 2008 : rachat de Spring par VMWare
    - Sortie de la V. 3 du framework
    - Nombreux sous-projets : Spring Security, Spring Data, Spring AMQP…
  - Actuellement (nov. 2017) : V. 5.1.2 RELEASE

Le framework Spring

- Fondements
  - Réaction à Java 2 EE
    - EJB2 : trop complexes
    - Framework intégrant de nombreuses fonctionnalités

➔Architecture autour d'un « conteneur léger »

➔Les composants sont des POJO

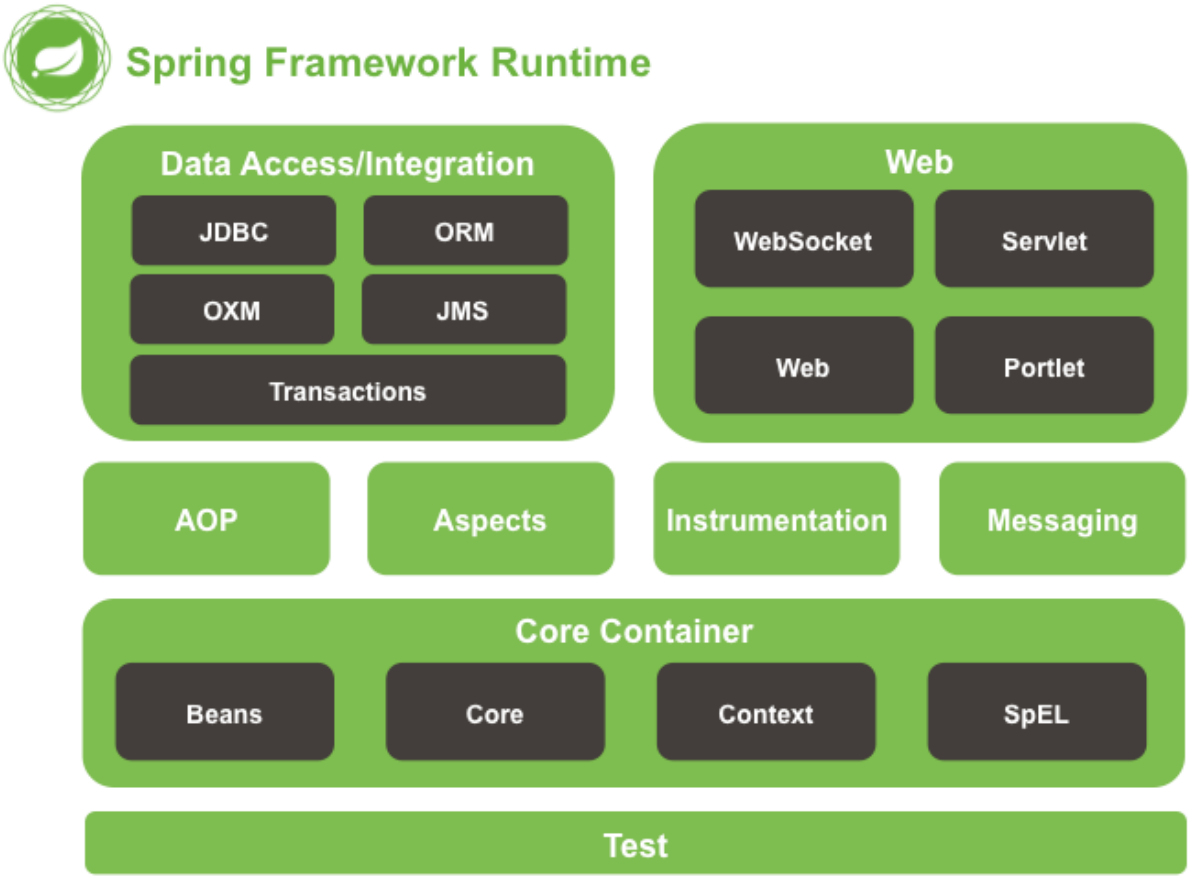
➔Intégration de fonctionnalités fournies par d'autres projets Open Source

➔Struts, Hibernate, JUnit, AspectJ, JSF…

➔La configuration tient une part centrale de la conception

➔« Opinionated »

Architecture globale



Source : http://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/overview.html

Spring Core container

- Rôle
  - Implémente le pattern IoC
    - Fournit un conteneur
    - Gère et met en œuvre les composants (beans)
  - Applique la configuration
    - Injection de dépendances par constructeur ou par setters
  - Fournit un contexte applicatif
  - Fournit des services annexes
    - AOP, communication orientée message, événements, services spécifiques à l'application (Web…)

Spring Core container

- Interfaces principales
  - org.springframework.beans.BeanFactory
    - Instancie les beans
    - Injecte les dépendances / gère la configuration
  - org.springframework.context.ApplicationContext
    - Dérive de la précédente
    - Représente (en partie) le conteneur (!)
    - Rajoute des services : AOP, messages, événements…

ApplicationContext

- Implémentations dans les applications standalone
  - ClassPathXmlApplicationContext ou FileSystemXmlApplicationContext
  - Dépend de la méthode d'accès au fichier de config Spring
  - À instancier dans la classe principale de l'application
    - ➔ **Crée un conteneur lié au contexte applicatif**
  - Exemples

```
ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");
```

ou

```
ApplicationContext context = new ClassPathXmlApplicationContext("services.xml",
"daos.xml");
```

ApplicationContext

- Implémentations dans les applications Web
  - Instanciation par le conteneur Web à partir du fichier de configuration de l'application (web.xml)
  - Crée un conteneur Spring lié au contexte
  - Utilisation d'un ContextLoader
    - **org.springframework.web.context.ContextLoaderListener** (à partir de Servlet 2.4)
    - Remarque : ContextLoaderServlet (jusqu'à Servlet 2.3) ne fait plus partie de l'API Spring 3.0
  - Remarque
    - Il faut s'assurer que le fichier web.xml est bien pris en compte par le conteneur de servlets

Composants

- Les beans Spring sont des POJOs
  - Instanciés par le conteneur
    (à partir de **BeanDefinition**s)
    - Nom (id)
    - Classe (pleinement qualifiée)
    - Dépendances
    - Scope
  - Exemple

```
<bean id="exampleBean" class="examples.ExampleBean"/>
```

Doc : https://docs.spring.io/spring/docs/5.1.2.RELEASE/spring-framework-reference/core.html#beans-definition

Composants

- Injection de dépendances
  - Par constructeur
  - Par propriétés (setters)
- Scopes
  - Singleton (défaut)
  - Prototype : une instance par dépendance d'un autre bean
  - Request, session, global session : spécifique au conteneur Web
  - User-defined

Composants

- Techniquement, il existe plusieurs sortes de beans
  - Controller
  - Service
  - Resource
  - …
- Conventions de nommage
  - Utiliser les mêmes conventions que pour les instances d'objets (camel-case)
  - Sinon : risque d'erreur si un élément fait de l'introspection

Exemple d'application


```
// create and configure beans
ApplicationContext context =
      new ClassPathXmlApplicationContext("services.xml",
"daos.xml");

// retrieve configured instance
PetStoreService service =
      context.getBean("petStore", PetStoreService.class);

// use configured instance
List<String> userList = service.getUsernameList();
```

Configuration

- Référentiel de dépendances
  - Définit un ensemble de beans
  - Précise leurs dépendances
    - Valeurs d'initialisation
    - Collaborateurs
  - 3 syntaxes
    - XML
    - Annotations
    - Programmation
  - Remarque : il peut y avoir plusieurs configurations dans un même conteneur

Configuration

- Configuration par fichier XML

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

  <bean id="Toto" class="monPackage.beans.TotoBean">
    <constructor-arg type="String" value="Bonjour"/>
    <property name="prop1" value="Il fait beau"/>
    <property name="prop2" ref="Titi"/>
  </bean>

  <bean id="Titi" class="monPackage.beans.TitiBean"/>
</beans>
```

Injection par constructeur

Injection par setter

Collaborateur

Configuration

- Configuration par annotations
  - Nécessite un fichier de configuration – presque – vide

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-
context.xsd">
   <context:annotation-config/>
   <context:component-scan base-package="mon.package.de.base"/>
</beans>
```

Configuration

- Configuration par annotations
  - Annotations de classes
    - @Component : composant générique
    - @Repository : dérive de @Component, dédié à la persistance
    - @Service : dérive de @Component, dédié aux services (objets du modèle)
    - @Controller : dérive de @Component, voir Spring Web MVC
  - Annotations internes aux classes (setters)
    - @Required : force le conteneur à injecter une valeur (définie explicitement ou par autowiring)
    - @Autowired : injection par résolution du référentiel de dépendances

Configuration

- Exemple de bean annoté

```
@Service
public class SimpleMovieLister {
 private MovieFinder movieFinder;
 private ActorFinder actorFinder;
  @Required
  public void setMovieFinder(MovieFinder movieFinder) {
      this.movieFinder = movieFinder;
  }
  @Autowired
  public void setActorFinder(MovieFinder actorFinder) {
      this.actorFinder = actorFinder;
  }
}
```

Configuration

- Configuration par programmation (1/2)
  - On crée une classe de configuration
    - Annotation : @Configuration
  - On y déclare les beans
    - Annotation : @Bean
  - On instancie le contexte en lui passant cette classe en paramètre

Configuration

- Configuration par programmation (1/2)
  - Exemple

```
@Configuration
public class AppConfig {
  @Bean
  public MyService myService() {
      return new MyServiceImpl();
  }
}
```

-------- dans le main --------

```
ApplicationContext ctx = new
  AnnotationConfigApplicationContext(AppConfig.class);
MyService myService = ctx.getBean(MyService.class);
myService.doStuff();
```

Configuration

- Configuration par programmation (2/2)
  - Autre méthode
    - Instancier un contexte vide
    - Utiliser context.register()

```
public static void main(String[] args) {
  AnnotationConfigApplicationContext ctx =
                new AnnotationConfigApplicationContext();
  ctx.register(AppConfig.class, OtherConfig.class);
  ctx.register(AdditionalConfig.class);
  ctx.refresh();
  MyService myService = ctx.getBean(MyService.class);
  myService.doStuff();
}
```

Configuration

- Résolution automatique du référentiel de dépendances (autowiring)
  - S'applique spécifiquement à chaque bean

```
<bean id="Titi" class="TitiBean" autowire="constructor"/>
```

    - Annotation @Autowired
  - Valeurs
    - no (défaut) : pas d'autowiring
    - byName : par nom de propriété
    - byType : par type de propriété
    - constructor : par type d'arguments du constructeur

Configuration

- Interfaces d'« awareness »
  - Par défaut, un bean ne connaît pas le conteneur.
  - Il est possible de rendre le conteneur visible par un bean à l'aide de l'interface ApplicationContextAware

```
public class MonBean implements ApplicationContextAware
```

  - En fait, on peut rendre un bean « aware » de différentes parties du framework
    - Contexte (conteneur)
    - Contexte Web
    - Son nom de bean dans la configuration…

Configuration

- Gestion du cycle de vie
  - Il est possible de spécifier les méthodes de cycle de vie d'un bean dans la configuration
  - On appelle ces méthodes « initialization callback » et « destruction callback »

```
<bean id="exampleBean" class="examples.ExampleBean"
                       init-method="init"
                       destroy-method="destroy"/>
```

  - Spring fournit des mécanismes plus fins à l'aide des interfaces LifeCycle et LifeCycleProcessor

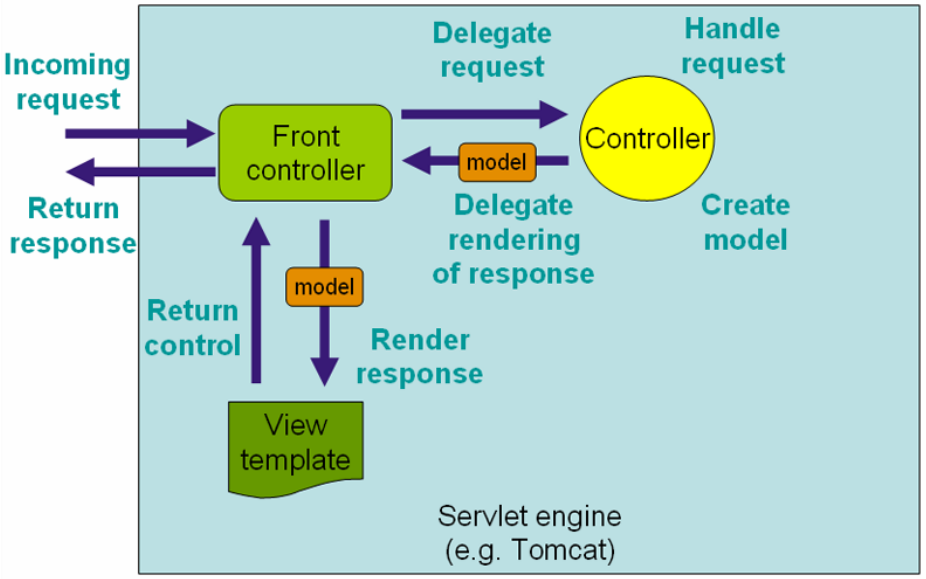
Configuration

- Remarques
  - Il existe de nombreuses autres options de configuration
    - Collections
    - Nested beans
    - …

➔ https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html

Spring Web MVC

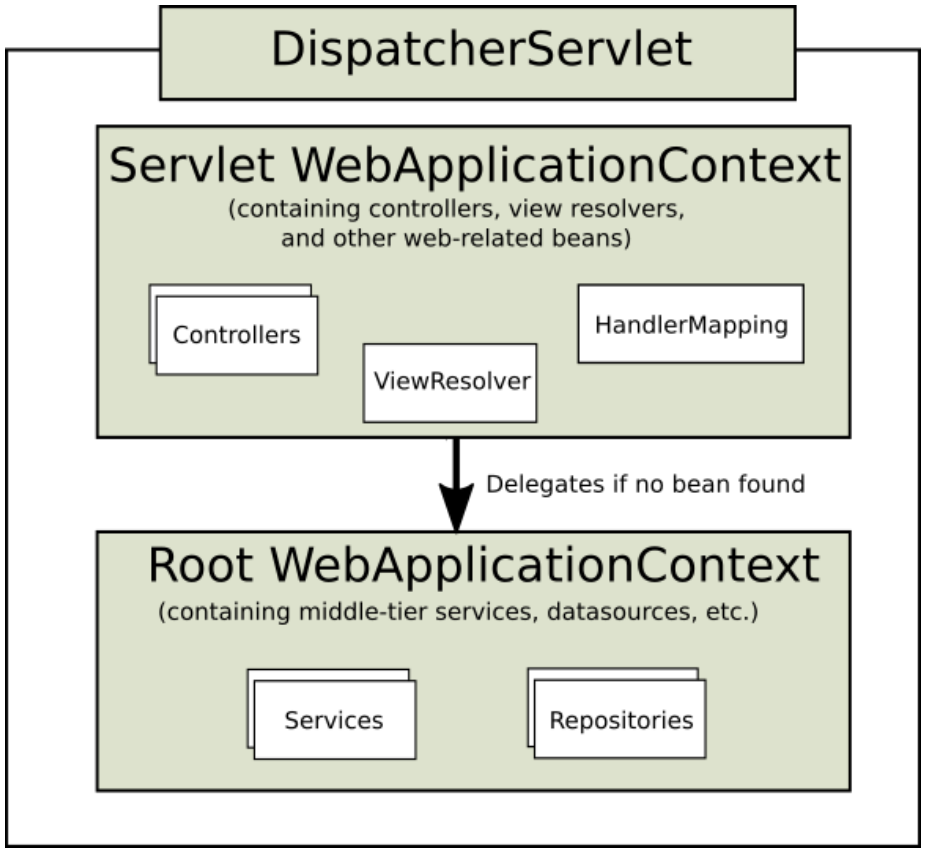
- MVC de type 2
  - Front controller : DispatcherServlet (fournie par Spring)
  - Contrôleurs délégués : composants (@Controller)



Source : http://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/mvc.html

WebApplication Context

- Hiérarchies de contextes
  - Root : 1 contexte commun à l'application
  - Servlet : des contextes spécifiques à chaque DispatcherServlet



Source : https://docs.spring.io/spring/docs/5.0.1.RELEASE/spring-framework-reference/images/mvc-context-hierarchy.png

WebApplication Context

- Configuration d'un Servlet WebApplicationContext (obligatoire) 1/2

```
<web-app>
    <servlet>
        <servlet-name>example</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>example</servlet-name>
        <url-pattern>/example/*</url-pattern>
    </servlet-mapping>
</web-app>
```

  - Remarques
    - Cette configuration nécessite un fichier de configuration de composants nommé : `/WEB-INF/example-servlet.xml`
    - Mapper les URLs sur `/*` est une mauvaise idée…

WebApplication Context

- Configuration d'un Servlet WebApplicationContext (obligatoire) 2/2

```
<web-app>
    <servlet>
        <servlet-name>example</servlet-name>
        <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/tpspring-servlet.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>example</servlet-name>
        <url-pattern>/example/*</url-pattern>
    </servlet-mapping>
</web-app>
```

  - Permet de choisir le(s) nom(s) du(des) fichier(s) XML

WebApplication Context

- Configuration du Root WebApplicationContext (facultatif)
  - Exemple de fichier web.xml

```
<webapp>
  <listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/daoContext.xml
              /WEB-INF/applicationContext.xml</param-value>
  </context-param>
</webapp>
```

Composants MVC

- Exemple de contrôleur annoté

```
@Controller
@RequestMapping("/appointments")
public class AppointmentsController {
    private final AppointmentBook appointmentBook;

    @Autowired
    public AppointmentsController(AppointmentBook apptmentBook) {
        this.appointmentBook = apptmentBook;
    }

    @RequestMapping(method = RequestMethod.GET)
    public String get() {
        return "appointments/today";
    }
}
```

Frameworks MVC : Spring

- Spring Web MVC
  - Méthodes de service (Handler methods)
    - Annotées avec `@RequestMapping` (ou `@GetMapping`, `@PostMapping`…)
    - Permettent
      - De récupérer les paramètres de la requête
      - De faire du data binding entre les paramètres et le modèle
      - D'appeler les beans concernés
      - De passer les infos (`Model`) nécessaires à la vue pour générer la réponse
    - https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc-ann-methods

Frameworks MVC : Spring

- Spring Web MVC
  - Méthodes de service (Handler methods)
    - Signature « flexible »
      - Paramètres
        - » Model, `@ModelAttribute`
        - » Paramètres de la requête : `@RequestParam`
        - » Paramètres « classiques des servlets : `ServletRequest`, `ServletResponse`, `HttpSession`
        - » …
      - Valeurs de retour
        - » String : nom de vue (cf. slide précédent)
        - » Objet `View`
        - » Objet `ModelAndView`
        - » String annotée `@ResponseBody`
        - » …
    - https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc-ann-methods

Frameworks MVC : Spring

- Spring Web MVC
  - Méthodes de service (Handler methods)
    - Exemples

```
@RequestMapping
@ModelAttribute
public void populateModel(@RequestParam String number, Model model) {
    model.addAttribute(accountRepository.findAccount(number));
    // add more ...
}
```

```
@PostMapping("/login")
public ModelAndView login(LoginData loginData) {
    if (LOGIN.equals(loginData.isValid())) {
        return new ModelAndView("success", new User("test"));
    } else {
        return new ModelAndView("failure", null);
    }
}
```

Composants MVC

- View resolving
  - Objectif : faire correspondre une vue au retour du contrôleur
  - Interface View
    - Traite la requête en fonction d'une technologie de vue (JSP, JSF…)
  - Interface ViewResolver
    - Fournit un mapping entre nom de vue et objet View

Composants MVC

- View resolving
  - Exemple de configuration

```
<bean id="viewResolver"
   class="org.springframework.web.servlet.view.UrlBasedV
iewResolver">
  <property name="viewClass"
   value="org.springframework.web.servlet.view.JstlView"
/>
  <property name="prefix" value="/WEB-INF/jsp/"/>
  <property name="suffix" value=".jsp"/>
</bean>
```

Spring Boot

- Position du problème
  - Actuellement : de nombreux projets peuvent être liés à une application Spring
  - Framework puissant mais lourd à configurer
- Objectif général
  - Simplifier la configuration des projets
- Principe général
  - Convention over configuration

Spring Boot

- Principes
  - Créer facilement une application Java standalone
    - Avec très peu de configuration
      - Les choix de configuration par défaut sont déjà faits (« opinionated »)
    - Capable de faire tourner une application Spring (Web ou non)
    - Avec en standard la gestion d'un ensemble de préoccupations non fonctionnelles
      - serveurs embarqués, sécurité, métriques, configuration externalisée…
  - Pas de génération de code ni de fichier de configuration XML
  - Une interface en ligne de commande pour démarrer plus rapidement

Spring Boot

- Utilisation
  - Créer un projet Maven qui hérite d'un parent
    - spring-boot-starter-parent
  - Insérer les dépendances et les plugins nécessaires dans le pom.xml
  - Lier le code à une classe principale
    - public static void main
  - Lancer l'application
    - Avec un goal Maven spécifique : `mvn spring-boot:run`
    - Comme une application Java standard : java -jar …
    - Générer un fichier war et le déployer sur votre serveur

Spring Boot

- Exemple

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.0.BUILD-SNAPSHOT</version>
    <relativePath/>
</parent>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
```

Spring Boot

- Fonctionnement
  - @EnableAutoConfiguration déclenche un ensemble de mécanismes qui « devinent » la configuration en fonction des dépendances dans le pom.xml
    - Exemple : spring-boot-starter-web → projet Web
    - Spring boot essaye aussi de deviner à partir des dépendances non « spring-boot-starter-… »
  - La méthode main appelle la méthode run de SpringApplication avec en argument
    - Le composant Spring principal
    - Les arguments éventuellement passés en ligne de commande

Spring Boot

- Exemple

```
import org.springframework.boot.*;
import org.springframework.boot.autoconfigure.*;
import org.springframework.stereotype.*;
import org.springframework.web.bind.annotation.*;

@Controller
@EnableAutoConfiguration
public class SampleController {
    @RequestMapping("/")
    @ResponseBody
    String home() {
        return "Hello World!"; }
    public static void main(String[] args) throws Exception {
        SpringApplication.run(SampleController.class, args);
    }
}
```

Spring Boot

- Structuration du code (annotations)
  - **@EnableAutoConfiguration** Identifie la classe principale et démarre la détection de configuration
  - **@ComponentScan** déclenche la recherche de composants dans les sous-packages
  - **@Configuration** spécifie une classe de configuration (annotation Spring standard)

Spring Boot

- Structuration du code (annotations)
  - **@SpringBootApplication**
    - Rassemble les annotations
      - **@ComponentScan**
      - **@Configuration**
      - **@EnableAutoConfiguration**
    - À utiliser
      - Pour « économiser » les annotations dans un projet standard
      - Quand la classe principale n'est pas seule dans un package

Spring Boot

- Starters
  - Dépendances standard aux autres projets Spring dans le pom.xml
  - Permettent à Spring Boot de générer une configuration
  - Forme
    - Starters « officiels » : spring-boot-starter-*
    - Possibilité de créer ses starters soi-même : moi-spring-boot-starter
  - Liste des starters officiels

    https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#using-boot-starter

Spring Boot

- Structuration du code (bonnes pratiques)
  - Pas de « default package »
  - Position de la classe principale (point d'entrée)
    - À la racine et les autres packages au-dessous (meilleure solution)
    - Pas seule dans un package (utiliser l'annotation @SpringBootApplication)
  - Préférer les classes de configuration Java plutôt que les fichiers de configuration XML

Spring Boot

- Configuration spécifique
  - Si besoin de modifier la configuration par défaut

```
@Configuration
 @EnableAutoConfiguration(
    exclude={DataSourceAutoConfiguration.class})
public class MyConfiguration {
    ...
}
```

Spring Boot

- Création d'un projet vide

  https://start.spring.io/

Spring AOP

- Deux outils de POA dans Spring
  - Spring AOP : basé sur CGLIB (dans Spring Core depuis V5)
  - AspectJ (depuis la version 2.5)
- Spring AOP
  - Utilise des classes et de la configuration
    - Un aspect est une classe « normale » (POJO)
    - Les pointcuts relient le code de l'application aux méthodes d'un aspect
  - Permet le weaving à la compilation et à l'exécution

Spring AOP

- Spring AOP
  - Exemple 1 : configuration XML

```
<aop:config>
 <aop:pointcut id="servicePointcut"
               expression="execution(* ew.service.*.*(..))"/>
 <aop:aspect id="loggingAspect" ref="monLogger">
  <aop:before method="logMethodEntry"
              pointcut-ref="servicePointcut"/>
  <aop:after-returning method="logMethodExit"
              returning="result" pointcut-
ref="servicePointcut"/>
 </aop:aspect>
</aop:config>

<bean id="monLogger" class="ew.aop.MonLogger"/>
<bean name="monService" class="ew.service.MonService" />
```

Spring AOP

- Spring AOP
  - Exemple 2 : annotations

```
@Aspect
@Component
public class ExampleAspect {
    @Before("execution(* com.xyz.myapp.dao.*.*(..))")
    public void doAccessCheck() {
        // ...
    }
    @Around("com.xyz.myapp.SystemArchitecture.businessService()")
    public Object doBasicProfiling(ProceedingJoinPoint pjp) throws
Throwable {
        // start stopwatch
        Object retVal = pjp.proceed();
        // stop stopwatch
        return retVal;
}
```

Source : https://docs.spring.io/spring/docs/5.1.2.RELEASE/spring-framework-reference/core.html#aop-advice

Conclusion

- Avantages de Spring
  - Légèreté du framework (…)
  - S'appuie sur des solutions open source éprouvées
  - Possibilité de « plugger » d'autres fonctionnalités
  - Configuration rapide des applications « simples »
  - Très utilisé
  - Documentation abondante

Conclusion

- Faiblesses de Spring
  - Complexité croissante
    - Beaucoup de sous-projets
    - 3 types de configurations possibles
    - Trop de « magie » ?
  - Choix entre Spring et Java EE moins évident
    - EJB 3.0 plus simples

Références

- http://spring.io
- http://docs.spring.io/spring/docs/current/spring-framework-reference/html/
- http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html
- http://projects.spring.io/spring-boot/
- https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/
- http://docs.spring.io/spring/docs/current/javadoc-api/
- https://github.com/spring-projects/spring-mvc-showcase