Master 2 Traitement
de l'Information

Université Claude Bernard Lyon 1

Le framework Spring

Conception d'Applications Hétérogènes Distribuées

Lionel Médini

Septembre-novembre 2015

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…

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

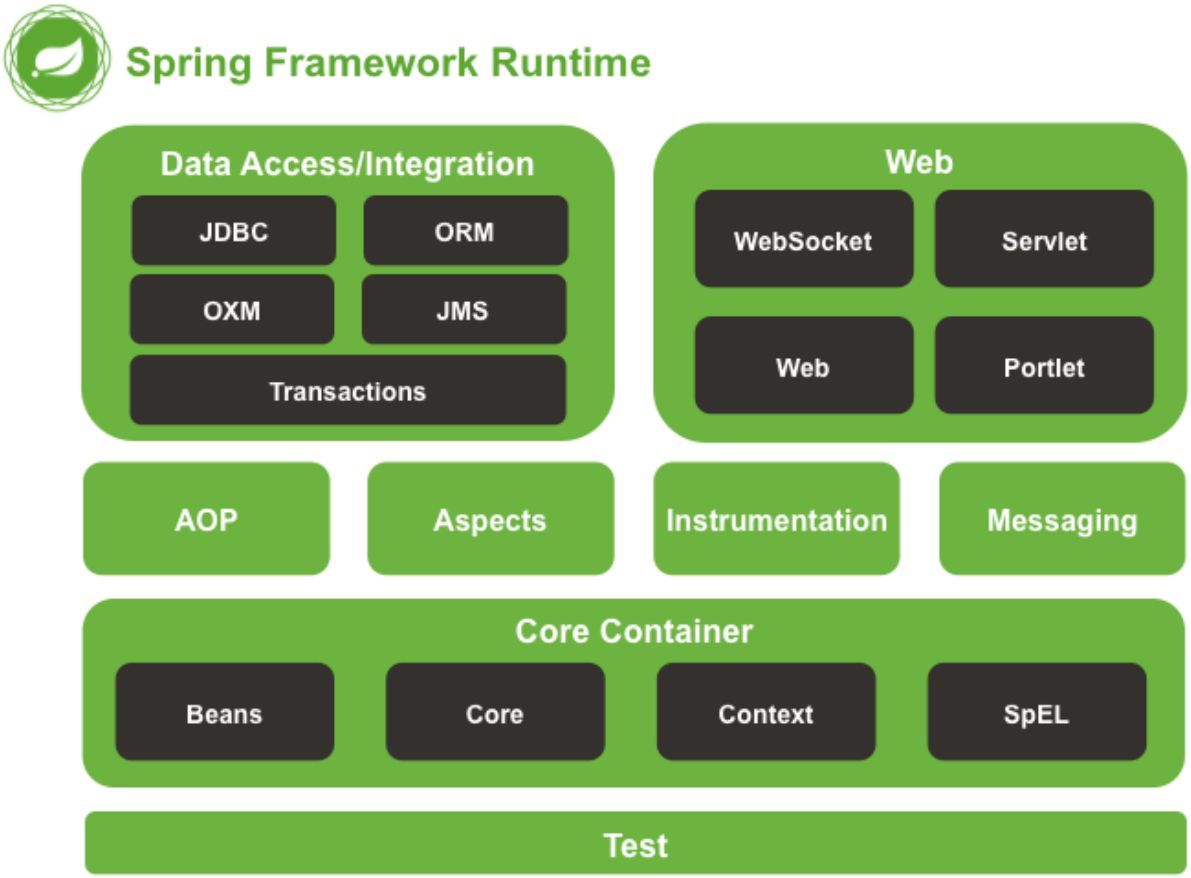
  ➔La configuration tient une part centrale de la conception

  ➔Gestion des fonctionnalités transverses à l'aide d'aspects

  ➔Intégration de fonctionnalités fournies par d'autres projets open source

  ➔Struts, Hibernate, JUnit, AspectJ…

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

Spring Core container

- Implémentations de ApplicationContext
  - 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
    - Exemples

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

ou

```
ApplicationContext context = new ClassPathXmlApplication
Context(new String[] {"services.xml", "daos.xml"});
```

Spring Core container

- Implémentations de ApplicationContext
  - Dans les applications Web
    - Instanciation par le conteneur Web à l'aide du fichier de configuration (web.xml)
    - 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

Spring Core container

- Implémentations de ApplicationContext
  - Dans les applications Web
    - Exemple de fichier web.xml

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/daoContext.xml
               /WEB-INF/applicationContext.xml</param-value>
</context-param>

<listener>
  <listener-class>org.springframework.web.context.ContextLoader
Listener</listener-class>
</listener>
```

Spring Core container

- Implémentations de ApplicationContext
  - Dans les applications Web
    - Remarques :
      - Eclipse peut générer cette configuration automatiquement
      - Il faut s'assurer que le fichier web.xml est bien pris en compte par le conteneur de servlets
      - Les fichiers XML passés en paramètres permettent de créer des contextes

Spring Core container

- Composants
  - Les beans Spring sont des POJOs
  - Instanciés par le conteneur (à partir d'un nom de classe)
  - Le nombre d'instances dépend de leur « scope »
    - Singleton (défaut)
    - Prototype : une instance par dépendance d'un autre bean
    - Request, session, global session : spécifique au conteneur Web
    - User-defined

Spring Core container

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

Spring Core container

- 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

Spring Core container

- Configuration :
  - Configuration par annotations
    - Annotations de classes
      - @Component : composant générique
      - @Repository : dérive de @Component, dédié à la persistence
      - @Service : dérive de @Component, dédié aux services (objets du modèle)
      - @Controller : dérive de @Component, dédié à la présentation (!)
    - 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

Spring Core container

- 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/>
</beans>
```

Spring Core container

- 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;
  }
}
```

Spring Core container

- 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

Spring Core container

- 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();
```

Spring Core container

- 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();
}
```

Spring Core container

- 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

Spring Core container

- 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…

Spring Core container

- 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

Spring Core container

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

    ➔http://docs.spring.io/spring/docs/3.2.5.BUILD-SNAPSHOT/spring-framework-reference/html/beans.html

Spring AOP

- Deux outils de POA dans Spring
  - Spring AOP : basé sur CGLIB
  - 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

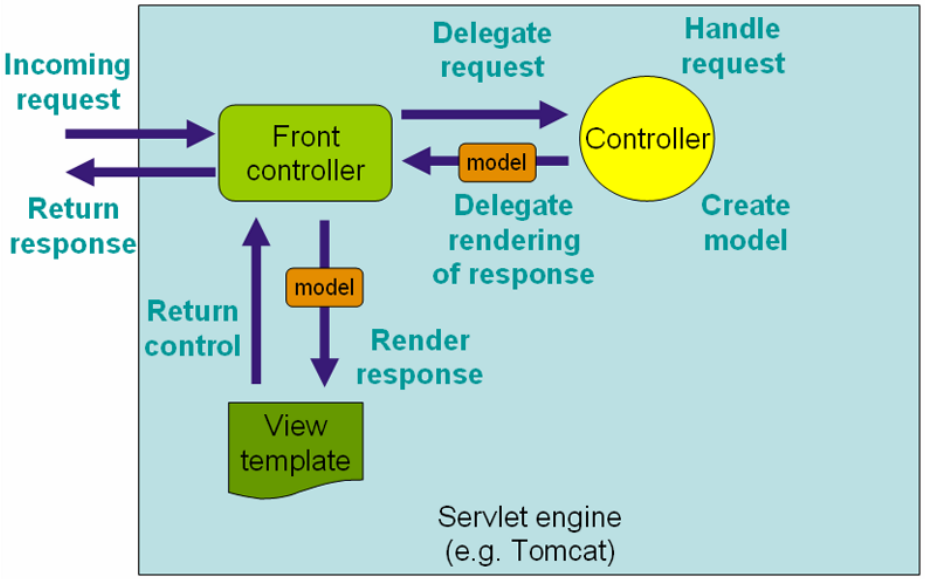
- Spring AOP
  - Exemple

```
<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 Web MVC

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



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

Spring Web MVC

- Exemple de configuration (web.xml)

```
<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>
```

  - Remarque
    - Cette configuration nécessite un fichier de configuration des contrôleurs nommé : /WEB-INF/example-servlet.xml

Spring Web MVC

- Exemple de configuration (xxx-servlet.xml)
  - Pour utiliser la configuration par annotations

```
<?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:p="http://www.springframework.org/schema/p"
   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:component-scan base-package="monAppli.web"/>
</beans>
```

Spring Web 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";
    }
}
```

Spring Web 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

Spring Web MVC

- View resolving
  - Exemple de configuration

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

Test

- Module de gestion des tests
  - Unitaires
    - Test des beans (POJOs) en dehors des conteneurs

Conclusion

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

Conclusion

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

Références

- http://spring.io
- http://docs.spring.io/spring/docs/3.2.4.RELEASE/spring-framework-reference/html/overview.html
- http://docs.spring.io/spring/docs/3.2.5.BUILD-SNAPSHOT/spring-framework-reference/html/beans.html
- http://www.jmdoudoux.fr/java/dej/chap-spring.htm