Spring : Mise en place et explication de principe de l'injection de dépendance

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en Programmation par contrainte (IA)
Ingénieur en Génie logiciel

elmouelhi.achref@gmail.com

Plan

Introduction

Spring, c'est quoi ?

- Un framework Java (JEE aussi)
- Créé par l'australien Rod JOHNSON en 2003.
- Développée par **Sun** puis **Oracle**
- Basé sur le concept de l'inversion de contrôle par la recherche et l'injection de dépendance.

Introduction

Le noyau Spring (Spring Core) comporte

- `Une fabrique` de beans (composants/classes java)
- `Un conteneur` pour stocker les beans
- `AOP` : Aspect Oriented Programming
- `JDBC` : Java Database Connectivity
- `ORM` : Object Relational Mapping
- `Servlet` pour la programmation web
- `JUnit` : pour les tests unitaires
- ...

Intégrer Spring sous Eclipse

Déroulement

- Dans le menu `Help`, choisir `Eclipse Marketplace`
- Dans la zone de saisie `Find`, saisir `Spring tools`
- Attendre la fin de chargement, ensuite sélectionner `Spring Tools 4 for Spring Boot (aka Spring Tool Suite 4)`
- Puis cliquer sur `Install`
- Enfin attendre la fin d'installation et redémarrer Eclipse

Inversion de contrôle ou injection de dépendance

Dépendance entre objets ?

Les objets de la classe C_1 dépendent des objets de la classe C_2 si :

- C_1 a un attribut objet de la classe C_2
- C_1 hérite de la classe C_2
- C_1 dépend d'un autre objet de type C_3 qui dépend d'un objet de type C_2
- Une méthode de C_1 appelle une méthode de C_2

Inversion de contrôle ou injection de dépendance

Injection de dépendance, pourquoi ?

- un mécanisme permettant de dynamiser la gestion de dépendance entre objets
- facilite l'utilisation des composants qu'on n'a pas développés
- minimise l'instanciation statique d'objets (avec l'opérateur `new`)

Inversion de contrôle ou injection de dépendance

Solutions avec Spring

- en utilisant un fichier `.xml`
- ou, en utilisant les annotations

Inversion de contrôle ou injection de dépendance

Pour cela, on peut

- soit créer un `Java Project` et ajouter les librairies Spring nécessaires.
- soit créer un `Maven Project` et ajouter les dépendances vers les librairies Spring dans le `pom.xml`

Inversion de contrôle ou injection de dépendance

Si **Eclipse** ne reconnait pas `ApplicationContext` ou `ClassPathXmlApplicationContext`

Il faut ajouter les fichiers suivants (que l'on peut trouver dans le `.m2/repository` au build path du projet)

- `spring-context`
- `commons-logging`
- `spring-beans`
- `spring-core`
- `spring-aop`
- `spring-expression`
- `spring-context-support`

Inversion de contrôle ou injection de dépendance

Pour cela

- Créer un projet maven `File > New > Maven Project`
- Cliquer sur `Next`
- Choisir `maven.archetype.quickstart`
- Valider
- Dans `pom.xml`, ajouter une dépendance vers `spring context support` et enregistrer

Inversion de contrôle ou injection de dépendance

Ajoutons la dépendance `spring-context-support` **dans** `pom.xml`

```
<!-- https://mvnrepository.com/artifact/org.
   springframework/spring-context-support -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context-support</artifactId>
    <version>5.1.1.RELEASE</version>
</dependency>
```

Inversion de contrôle ou injection de dépendance

Créons une classe `Personne` dans le package `org.eclipse.model`

```
package org.eclipse.model;

public class Personne {
  private int id;
  private String nom;

  public Personne() {}

  public Personne(int id, String nom) {
    this.id = id;
    this.nom = nom;
  }
  // + les getters setters
  public void afficher(){
    System.out.println(this.id+" "+this.nom);
  }
}
```

Inversion de contrôle ou injection de dépendance

Première solution avec un fichier .xml : créer un fichier `applicationContext` **dans** `src/main/java`

```
<?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"
  xsi:schemaLocation="http://www.springframework.org/schema/
     beans
                  http://www.springframework.org/schema/beans/
                    spring-beans-3.0.xsd">

  <bean id="per" class="org.eclipse.model.Personne">
    <constructor-arg value="1" type="int"></constructor-arg>
    <constructor-arg value="wick"></constructor-arg>
  </bean>
</beans>
```

Par défaut, le type d'un attribut est `string` (par exemple, `nom`)

Inversion de contrôle ou injection de dépendance

Et comment utiliser cet objet ?

```
package org.eclipse.classes;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.
   ClassPathXmlApplicationContext;

public class Main{

  public static void main(String[] args) {
    ApplicationContext context = new
       ClassPathXmlApplicationContext("applicationContext.xml")
       ;
    Personne p = context.getBean("per",Personne.class);
    p.afficher();
  }
}
```

Pas d'opérateur `new` ici.

Inversion de contrôle ou injection de dépendance

Et comment utiliser cet objet ?

```
package org.eclipse.classes;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.
   ClassPathXmlApplicationContext;

public class Main{

  public static void main(String[] args) {
    ApplicationContext context = new
       ClassPathXmlApplicationContext("applicationContext.xml")
       ;
    Personne p = context.getBean("per",Personne.class);
    p.afficher();
  }
}
```

Pas d'opérateur `new` ici.

Et si on a une dépendance avec une autre classe ou une collection

Inversion de contrôle ou injection de dépendance

Commençons par créer la classe `Adresse`

```
package org.eclipse.model;

public class Adresse {
  private String rue;
  private String codeP;
  private String ville;
  public Adresse() {
  }
  public Adresse(String rue, String codeP, String ville){
    this.rue = rue;
    this.codeP = codeP;
    this.ville = ville;
  }
  public String toString() {
    return "Adresse [rue=" + rue + ", codeP=" + codeP + "
       , ville=" + ville + "]";
  }
}
```

Inversion de contrôle ou injection de dépendance

Modifions la classe `Personne`

```
public class Personne {
  private int id;
  private String nom;
  private Adresse adresse;

  public Personne() {}

  public Personne(int id, String nom, Adresse adresse) {
    this.id = id;
    this.nom = nom;
    this.adresse = adresse;
  }

  public void afficher(){
    System.out.println(this.id+" "+this.nom + " " +
      adresse.toString());
  }
}
```

Inversion de contrôle ou injection de dépendance

Le fichier `applicationContext.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"
  xmlns:p="http://www.springframework.org/schema/p"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
                http://www.springframework.org/schema/beans/spring-
                   beans-3.0.xsd">

  <bean id="adresse" class="org.eclipse.model.Adresse">
    <constructor-arg value="paradis"></constructor-arg>
    <constructor-arg value="13015"></constructor-arg>
    <constructor-arg value="Marseille"></constructor-arg>
  </bean>

  <bean id="per" class="org.eclipse.model.Personne">
    <constructor-arg value="1" type="int"></constructor-arg>
    <constructor-arg value="wick"></constructor-arg>
    <constructor-arg>
          <ref bean="adresse"/>
    </constructor-arg>
  </bean>
</beans>
```

Inversion de contrôle ou injection de dépendance

On ne modifie rien dans le main

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.
   ClassPathXmlApplicationContext;

public class Main{

  public static void main(String[] args) {
    ApplicationContext context = new
       ClassPathXmlApplicationContext("applicationContext
       .xml");
    Personne p = context.getBean("per",Personne.class);
    p.afficher();
  }
}
```

Toujours sans l'opérateur `new`.

Inversion de contrôle ou injection de dépendance

Et les collections ?

```
import java.util.Iterator;
import java.util.List;

public class Personne {
  private int id;
  private String nom;
  private List<String> sports;

  public Personne() {}

  public Personne(int id, String nom, List<String> sports) {
    this.id = id;
    this.nom = nom;
    this.sports = sports;
  }

  public void afficher(){
    System.out.println(this.id+" "+this.nom );
    System.out.println("Mes sports : ");
    for(String sport : sports)
      System.out.println(sport);
  }
}
```

Inversion de contrôle ou injection de dépendance

Le fichier `applicationContext.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"
  xmlns:p="http://www.springframework.org/schema/p"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
                http://www.springframework.org/schema/beans/spring-
                   beans-3.0.xsd">

  <bean id="per" class="org.eclipse.model.Personne">
    <constructor-arg value="1" type="int"></constructor-arg>
    <constructor-arg value="wick"></constructor-arg>
    <constructor-arg>
      <list>
        <value>foot</value>
        <value>hand</value>
        <value>basket</value>
      </list>
    </constructor-arg>
  </bean>
</beans>
```

Inversion de contrôle ou injection de dépendance

On ne modifie toujours rien dans le main

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.
   ClassPathXmlApplicationContext;

public class Main{

  public static void main(String[] args) {
    ApplicationContext context = new
       ClassPathXmlApplicationContext("applicationContext
       .xml");
    Personne p = context.getBean("per",Personne.class);
    p.afficher();
  }
}
```

Toujours sans l'opérateur `new`.

Inversion de contrôle ou injection de dépendance

Remarques

- Dans les exemples précédents, on a fait une injection de dépendance en utilisant le constructeur.
- Mais, on peut aussi faire une injection de dépendance en utilisant le setter.

Inversion de contrôle ou injection de dépendance

Supprimons les différents constructeurs de l'exemple précédent

```
public class Personne {
        private int id;
        private String nom;
        public int getId() {
                return id;
        }
        public void setId(int id) {
                this.id = id;
        }
        public String getNom() {
                return nom;
        }
        public void setNom(String nom) {
                this.nom = nom;
        }
        public void afficher(){
            System.out.println(this.id+" "+this.nom );
        }
}
```

Inversion de contrôle ou injection de dépendance

Le fichier `applicationContext.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"
  xmlns:p="http://www.springframework.org/schema/p"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
                http://www.springframework.org/schema/beans/spring-
                   beans-3.0.xsd">

  <bean id="per" class="org.eclipse.model.Personne">
    <property name="id">
      <value>1</value>
    </property>
    <property name="nom">
      <value>Wick</value>
    </property>
  </bean>
</beans>
```

Inversion de contrôle ou injection de dépendance

On ne modifie toujours rien dans le main

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.
   ClassPathXmlApplicationContext;

public class Main{

  public static void main(String[] args) {
    ApplicationContext context = new
       ClassPathXmlApplicationContext("applicationContext
       .xml");
    Personne p = context.getBean("per",Personne.class);
    p.afficher();
  }
}
```

Toujours sans l'opérateur `new`.

Inversion de contrôle ou injection de dépendance

Et si on a un objet de type `Adresse` dans `Personne`

```
public class Adresse {
  private String rue;
  private String codeP;
  private String ville;
  public Adresse() {
  }
  public Adresse(String rue, String codeP, String ville){
    this.rue = rue;
    this.codeP = codeP;
    this.ville = ville;
  }
  public String toString() {
    return "Adresse [rue=" + rue + ", codeP=" + codeP + "
       , ville=" + ville + "]";
  }
}
```

Inversion de contrôle ou injection de dépendance

Modifions la classe `Personne`

```
public class Personne {
  private int id;
  private String nom;
  private Adresse adresse;

  public int getId() {
    return id;
  }

  public void setId(int id) {
    this.id = id;
  }

  public String getNom() {
    return nom;
  }

  public void setNom(String
     nom) {
    this.nom = nom;
  }

  public Adresse getAdresse()
     {
    return adresse;
  }

  public void setAdresse(
     Adresse adresse) {
    this.adresse = adresse;
  }

  public void afficher(){
    System.out.println(this.id
       +" "+this.nom + " " +
       adresse.toString());
  }
}
```

Inversion de contrôle ou injection de dépendance

Le fichier `applicationContext.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"
  xmlns:p="http://www.springframework.org/schema/p"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
                http://www.springframework.org/schema/beans/spring-
                   beans-3.0.xsd">
  <bean id="adr" class="org.eclipse.model.Adresse">
    <constructor-arg value="paradis"></constructor-arg>
    <constructor-arg value="13015"></constructor-arg>
    <constructor-arg value="Marseille"></constructor-arg>
  </bean>
  <bean id="per" class="org.eclipse.model.Personne">
    <property name="id" value="1"></property>
    <property name="nom">
      <value>Wick</value>
    </property>
    <property name="adresse" ref="adr">
    </property>
  </bean>
</beans>
```

Inversion de contrôle ou injection de dépendance

On ne modifie rien dans le main

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.
   ClassPathXmlApplicationContext;

public class Main{

  public static void main(String[] args) {
    ApplicationContext context = new
       ClassPathXmlApplicationContext("applicationContext
       .xml");
    Personne p = context.getBean("per",Personne.class);
    p.afficher();
  }
}
```

Toujours sans l'opérateur `new`.

Inversion de contrôle ou injection de dépendance

On peut aussi utiliser l'usine de bean dans le main

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.
   ClassPathXmlApplicationContext;
import org.springframework.beans.factory.BeanFactory;

public class Main {

  public static void main(String[] args) {
    ApplicationContext context = new
       ClassPathXmlApplicationContext("applicationContext
       .xml");
    BeanFactory factory=(BeanFactory) context;
    Personne p=(Personne)factory.getBean("per");
    p.afficher();
  }
}
```

Toujours sans l'opérateur `new`.

Inversion de contrôle ou injection de dépendance

Remarques

- Nous verrons aussi qu'on peut également utiliser des annotations (telles que `@Autowired`, `@Component`, `@Service`, `@Resource`, `@Repository`...) pour faire l'injection de dépendance

Inversion de contrôle ou injection de dépendance

Dans le package `org.eclipse.nation`**, créons une interface** `European` **et deux classes** `French` **et** `English` **qui l'implémentent**

```
public interface European {
        public void saluer();
}
```

```
public class French implements European{
        public void saluer() {
                System.out.println("Bonjour");
        }
}
```

```
public class English implements European{
        public void saluer() {
                System.out.println("Hello");
        }
}
```

Inversion de contrôle ou injection de dépendance

Préparons le `main` et notre fichier de configuration

```
public class Main {
  public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext("
       applicationContext.xml");
    European e = (European) context.getBean("european");
    e.saluer();
  }
}
```

```
<?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"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
     http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
     http://www.springframework.org/schema/context
     http://www.springframework.org/schema/context/spring-context-4.0.
        xsd">
  <context:component-scan base-package="org.eclipse.nation" ></
     context:component-scan>
 </beans>
```

Inversion de contrôle ou injection de dépendance

Remarques

- La balise `<context:component-scan base-package="org.eclipse.nation" > </context:component-scan>` permet d'indiquer l'emplacement de beans
- Pour déclarer un bean, il faut annoter la classe par `@Component`
- `European` est une interface, donc on ne peut instancier.

Inversion de contrôle ou injection de dépendance

Ajoutons l'annotation `@Component`

```
@Component
public class French implements European{
        public void saluer() {
                System.out.println("Bonjour");
        }
}
```

```
@Component
public class English implements European{
        public void saluer() {
                System.out.println("Hello");
        }
}
```

Modifions le `main`

```
public class Main {
  public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext("
      applicationContext.xml");
    European e = (European) context.getBean("french");
    e.saluer();
  }
}
```

Inversion de contrôle ou injection de dépendance

Remarques

- En exécutant, un `Bonjour` s'affiche. Si on remplace `french` par `english` dans `getBean()`, un `Hello` sera affiché.
- Le **CamelCase** est important pour la reconnaissance des beans.
- Toutefois, nous pouvons aussi attribuer des noms à nos composants pour pouvoir les utiliser plus tard.

Inversion de contrôle ou injection de dépendance

Exemple

```
@Component("eng")
public class English implements European{
        public void saluer() {
                System.out.println("Hello");
        }
}
```

Modifions le `main`

```
public class Main {
  public static void main(String[] args) {
    ApplicationContext context = new
       ClassPathXmlApplicationContext("applicationContext.xml")
       ;
    European e = (European) context.getBean("eng");
    e.saluer();
  }
}
```

On ne peut plus appeler le bean par le nom `english`

Inversion de contrôle ou injection de dépendance

Exemple 2 :

```
@Component
public class Car {
  public void start() {
    System.out.println("Voiture démarrée et prête à
       rouler");
  }
}
```

```
@Component
public class Person {
  Car c;
  public void drive() {
    System.out.println("Je suis prêt à conduire");
    c.start();
  }
}
```

Inversion de contrôle ou injection de dépendance

```
public class Main {
  public static void main(String[] args) {
    ApplicationContext context = new
       ClassPathXmlApplicationContext("
       applicationContext.xml");
    Person p = (Person) context.getBean("person");
    p.drive();
  }
}
```

En exécutant ce code, une erreur sera affichée car on ne peut appeler la méthode `start()` si la classe `car` n'a pas été instanciée.

Inversion de contrôle ou injection de dépendance

Solution : utiliser une annotation pour créer l'objet `c` après instanciation de la classe `Person`

```
@Component
public class Person {
   @Autowired
   Car c;
   public void drive() {
      System.out.println("Je suis prêt à conduire");
      c.start();
   }
}
```

Inversion de contrôle ou injection de dépendance

Remarques

- `@Autowired` cherche les beans selon le type.
- Pour chercher un bean selon le nom, il faut utiliser l'annotation `@Qualifier`

Inversion de contrôle ou injection de dépendance

```
@Component("c1")
public class Car {
  public void start() {
    System.out.println("Voiture démarrée et prête à
      rouler");
  }
}
```

```
@Component
public class Person {
  @Autowired
  @Qualifier("c1")
  Car c;
  public void drive() {
    System.out.println("Je suis prêt à conduire");
    c.start();
  }
}
```

Inversion de contrôle ou injection de dépendance

Remarques

- Il existe plusieurs autres notations telles que `@Repository`, `@Service`... que nous verrons dans les chapitres suivants.
- Il est possible de faire les mêmes opérations précédentes en créant un projet Spring avec Maven.