

Génie Logiciel Avancée

Architecture logicielle évoluée :

Framework Spring

03- Spring Ioc

Mohamed ZAYANI

2025/2026

Plan

- Qu'est-ce que IoC (Inversion od Control) ?
- Qu'est ce que DI (Dependency Injection)
- DI par description textuelle
- DI par annotation
- Modes d'injection dans Spring
- Mise en œuvre de IoC avec Spring et Spring Boot
- Exemples d'annotations utiles

Qu'est-ce que IoC ?

• Notion générale

- ✓ Dans la programmation **traditionnelle**, c'est le développeur qui contrôle la création et la gestion des objets.
- ✓ Avec **IoC (Inversion of Control)**, c'est le framework (**Spring**) qui prend le contrôle et se charge de :
 - créer les objets (nommés **beans**),
 - gérer leurs cycles de vie,
 - injecter les dépendances nécessaires.
- ✓ Une dépendance = un objet dont une classe a besoin pour fonctionner.
- ✓ Plutôt que de créer cette dépendance avec **new**, Spring va l'injecter dynamiquement.

→ Avantages

- Réduire le couplage entre classes.
- Rendre le code plus testable (on peut injecter de faux objets pour les tests).
- Rendre l'application plus flexible et maintenable.

Injection de dépendances (DI)

L'injection de dépendances (DI) est la façon dont IoC est appliquée dans Spring.

• Définition

- ✓ L'injection de dépendance (DI) consiste à éviter une dépendance directe entre deux classes en définissant **dynamiquement** la dépendance plutôt que statiquement.
- ✓ C'est un concept qui intervient généralement au début de l'exécution de l'application
 - Pour le cas du JEE, c'est au niveau de démarrage du serveur d'application JEE
 - Pour Spring, Il s'agit d'un conteneur **Spring IoC (Inversion of Control)** qui s'occupe de cette gestion en mémoire d'injection de **beans (objets Java)** dynamiquement

• Mise en oeuvre

Il consiste à créer dynamiquement (injecter) les dépendances entre les différents objets en s'appuyant:

- ❖ sur une description (**fichier de configuration**)
- ❖ ou de manière programmatique (**par annotation**)

Exemple sans IoC (couplage fort)

```
package dev.spring.service;  
public class MonService  
{  
    public void execute()  
    {  
        System.out.println("Service exécuté !");  
    }  
}
```

```
package dev.spring.vue;  
import dev.spring.service.MonService;  
public class Client  
{ // Création directe (statique)  
    private MonService service = new MonService();  
  
    public void doSomething()  
    {  
        service.execute();  
    }  
}
```

Constat:

- Ici, « Client » est **fortement couplé** à « MonService » car il le crée directement (new MonService()).
- Si on change l'implémentation de « MonService », il faudra modifier « Client »

Exemple avec IoC (couplage faible)

```
package dev.spring.service;
import org.springframework.stereotype.Service;

@Service
public class MonService {
    public void execute() {
        System.out.println("Service exécuté !");
    }
}
```

```
package dev.spring.vue;
import dev.spring.service.MonService;
import org.springframework.stereotype.Component;

@Component
public class Client {
    private final MonService service;
    // Injection de dépendance par constructeur
    public Client(MonService service) {
        this.service = service;
    }
    public void doSomething() {
        service.execute();
    }
}
```

Constat:

- Ici, Client ne crée plus l'objet « MonService ».
- C'est Spring qui injecte **automatiquement** une instance de « MonService » dans « Client ».

loc (avec un fichier texte)

• Mise en oeuvre

- ❖ Les noms des classes d'implémentation à injecter sont déclarés à l'extérieur du code source de la classe appelante dans un fichier texte de configuration.
- ❖ Le programme récupère, au moment de son exécution, les noms des classes d'implémentation à instancier à partir du fichier texte en question.

Exemple

Placer le nom de la classe d'implémentation (Ex: 'B') dans un fichier « config.txt » (sous un dossier « Ressources »)

1. Utiliser un objet 'Scanner' pour ouvrir le fichier en mode lecture
2. Récupérer le nom de la classe à instancier 'nomB'
3. Charger la classe en mémoire (avec le type 'Class')
4. Instancier un objet dynamiquement
5. Puis l'injecter dans l'objet 'a' avec la méthode 'setB()'

Pour utiliser une nouvelle version B2 de B, il suffit d'éditer le fichier 'config.txt' et écrire 'B2' au lieu de 'B' sans toucher le code source de A

```
import java.io.File;
import java.util.Scanner;
public class TestIoCParFichierTexte {
    public static void main(String[] args) throws Exception{
        A a = new A();

        Scanner scanner = new Scanner(new File("ressources/config.txt")); //1
        String nomB = scanner.nextLine(); // 2
        Class classeImp = Class.forName(nomB); // 3
        IB x = (IB) classeImp.newInstance(); // 4
        a.setB(x); //5
        a.afficher();
    }
}
```

loc (avec un fichier XML)

• Mise en oeuvre

- ❖ Les noms des classes d'implémentation à injecter sont déclarés dans un fichier XML dans des balises `<bean>` comme suit:

```
<?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/beans/spring-
    beans.xsd">

    <bean class="B" id = "x"></bean>
    <bean class="A" id = "a">
        <property name="b" ref="x"></property>
    </bean>
</beans>
```

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

public class TestIoCParFichierXML {

    public static void main(String[] args) {
        ApplicationContext springContext =
            new ClassPathXmlApplicationContext("config.xml");
        A a =(A) springContext.getBean("a");
        a.afficher();
    }
}
```

Liste des fichiers
JAR nécessaires



- commons-logging-1.2.jar
- spring-aop-4.3.2.RELEASE.jar
- spring-beans-4.3.2.RELEASE.jar
- spring-context-4.3.2.RELEASE.jar
- spring-core-4.3.2.RELEASE.jar
- spring-expression-4.3.2.RELEASE.jar

- ❖ Il est nécessaire d'ajouter les bibliothèques suivantes pour réaliser l'injection de dépendance

Pour utiliser une nouvelle version B2 de B, il suffit d'éditer le fichier 'config.xml' et écrire 'B2' au lieu de 'B' sans toucher le code source de A

loc (Par annotation – Spring loc)

- **Mise en oeuvre**
- ❖ Pour se faire, utiliser l'annotation « `@repository` » (pour la classe ‘B’) qui permet d’informer **Spring** que la classe d’implémentation ‘B’ est un **Spring BEAN**
- ❖ L’annotation « `@Repository` » permet de marquer la classe pour être utilisée par la suite par **Spring** en cas de besoin d’une instance d’une classe qui implémente l’interface « `IB` » (Ici c’est la classe B)
- ❖ Puis, utiliser l’annotation « `@Autowired` » dans la classe appelante (‘A’), juste avant la déclaration de l’attribut ‘`b`’ de type ‘`IB`’, pour indiquer l’emplacement de l’injection de dépendance.
- ❖ Spring cherche une classe déclarée (avec l’une des annotations `@Repository` ou `@Service` ou `@Component`) et qui implémente l’interface de déclaration de l’attribut ayant l’annotation « `@Autowired` ».

```
package cdi;
import org.springframework.stereotype.Repository;
@Repository
public class B implements IB {
    private String message = "classe B";
    public String getMessage()
    {
        return message;
    }
}
```

```
package cdi;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
@Service
public class A {
    @Autowired
    private IB b;
    public void setB(IB b) {
        this.b = b;}
    public void afficher()
    {
        System.out.println("Je suis une :" + b.getMessage());
    }
}
```

Modes d'injection dans Spring

L'injection de dépendances (ID), dans Spring, peut être effectuée de plusieurs manières:

❖ Par constructeur

```
package dev.spring.VUE;
import dev.spring.service.MonService;
import org.springframework.stereotype.Component;

@Component
public class Client {
    private final MonService service;
    // Injection de dépendance par constructeur
    @Autowired
    public Client(MonService service)
    {
        this.service = service;
    }
    public void doSomething() {
        service.execute();
    }
}
```

❖ Par modificateur

```
package dev.spring.VUE;
import dev.spring.service.MonService;
import org.springframework.stereotype.Component;

@Component
public class Client {
    private MonService service;
    // Injection de dépendance par modificateur
    @Autowired
    public void setService(MonService service)
    {
        this.service = service;
    }
    public void doSomething() {
        service.execute();
    }
}
```

❖ Par attribut

```
package dev.spring.VUE;
import
dev.spring.service.MonService;
import org.springframework.stereotype.Component;

@Component
public class Client {
    // Injection de dépendance par attribut
    @Autowired
    private MonService service;

    public void doSomething()
    {
        service.execute();
    }
}
```

L'annotation **@Autowired** permet d'indiquer l'emplacement d'injection

Conteneur Spring (Bean Container)

- Les objets gérés par Spring sont appelés **Beans**.
- Les beans sont déclarés via :
 - Annotations stéréotypes:
 - ❖ **@Component:** (composant générique),
 - ❖ **@Service:** (pour la couche service ou métier),
 - ❖ **@Repository:** (pour la couche DAO (accès aux données)),
 - ❖ **@Controller /@RestController:** (pour la couche web)
 - ou configuration dans un fichier XML (ancienne approche).
- Les beans sont injectés via :
 - Dans une classe de type « bean »: (déclarée comme un bean)
 - Par constructeur (avec l'annotation **@Autowired**) (surtout pour la version 4 de SpringBoot)
 - Par modificateur (avec l'annotation **@Autowired**)
 - Par attribut (avec l'annotation **@Autowired**)
 - Dans une classe non bean:
 - Par la méthode « **getBean** »
de la classe « **ApplicationContext** »

```
package cdi;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
public class TestIoCParAnnotation {
    public static void main(String[] args) {
        ApplicationContext springContext = new AnnotationConfigApplicationContext("cdi");
        A a =(A) springContext.getBean(A.class);
        a.afficher();}}
```

Mise en œuvre de « IoC » avec Spring et Maven

- Pour réaliser l’IoC et la DI vec **Spring**, il est nécessaire de déclarer deux dépendances:
 - **spring-context**
 - et **spring-core**
- Exemple minimal avec **Maven**:

```
<!-- Spring Core -->
<!-- http://mvnrepository.com/artifact/org.springframework/spring-core -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>4.1.4.RELEASE</version>
</dependency>
<!-- Spring Context -->
<!-- http://mvnrepository.com/artifact/org.springframework/spring-context -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>4.1.4.RELEASE</version>
</dependency>
```

- Cela inclut déjà :
 - **spring-core** (IoC, DI, AOP de base),
 - **spring-context** (conteneur de beans, ApplicationContext).

IoC avec Spring et maven

▪ Première solution:

```
package dev.spring;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import dev.spring.vue.*;

public class DemoApplication {
    public static void main(String[] args) {
        ApplicationContext context = new AnnotationConfigApplicationContext("dev.spring");
        Client client = context.getBean(Client.class);
        client.doSomething();
    }
}
```

▪ Deuxième solution:

Créer un contexte à partir
d'une configuration
contenue
dans la classe
"AppConfiguration"

```
package dev.spring;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import dev.spring.vue.*;
public class DemoApplication2 {
    public static void main(String[] args) {
        // créer un contexte à partir d'une classe de configuration
        ApplicationContext context= new AnnotationConfigApplicationContext(AppConfiguration.class);
        Client client = context.getBean(Client.class);
        client.doSomething();
    }
}
```

```
package dev.spring;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
@Configuration
@ComponentScan({"dev.spring"})
public class AppConfiguration { }
```

Mise en œuvre de « IoC » avec Spring Boot

- Avec **Spring Boot**, aucune dépendance spécifique à IoC n'est nécessaire.
→ L'IoC et DI font partie du **spring-context** et **spring-core** qui est inclu par défaut dans tout projet Spring Boot..
- Exemple minimal avec **Maven**:

```
<!-- Dépendance principale Spring Boot -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
    <version>3.5.5</version>
</dependency>
```

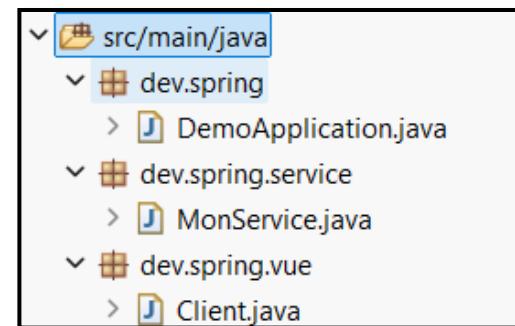
- Cela inclut déjà :
 - **spring-core** (IoC, DI, AOP de base),
 - **spring-context** (conteneur de beans, ApplicationContext).

@SpringBootApplication

- Spring Boot suit la règle : Convention over Configuration.
- Autrement dit, il y a très peu de configuration manuelle à faire.
- Dans la classe principale (MainClass), utiliser l'annotation « `@SpringBootApplication` » qui active automatiquement le conteneur IoC.

```
package dev.spring;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
@SpringBootApplication
public class DemoApplication
{
    public static void main(String[] args)
    {
        ApplicationContext context =
            SpringApplication.run(DemoApplication.class, args);
        //La classe « Client » est déjà déclarée comme un Bean
        Client client = context.getBean(Client.class);
        client.doSomething();
    }
}
```

Hiérarchie des packages



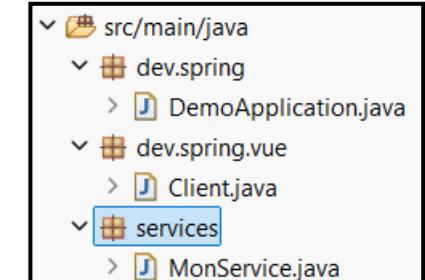
- Pour trouver les beans à injecter, Spring Boot scanne par défaut le package où se trouve la classe principale et ses sous-packages.
- Il est donc recommandé de mettre `@SpringBootApplication` à la racine du projet (dans le package parent).
- Mais si les classes sont en dehors de la hiérarchie du package principal, il est nécessaire d'ajouter une autre annotation « `@ComponentScan` » qui permet d'indiquer à Spring où chercher les beans.

@ComponentScan

- Si on change le nom du package de la classe « **MonService** » pour qu'il soit en dehors de la hiérarchie de la classe principale comme suit:

```
package services;
import org.springframework.stereotype.Service;
@Service
public class MonService
{ public void execute()
{
    System.out.println("Service exécuté !");
}
```

Nouvelle Hiérarchie des packages



- L'exécution de la classe principale déclenche l'erreur suivante:

```
*****
APPLICATION FAILED TO START
*****
Description:
Parameter 0 of constructor in dev.spring.vue.Client required a bean of type 'services.MonService' that could not be found.

Action:
Consider defining a bean of type 'services.MonService' in your configuration.[
```

- Ceci nécessite d'ajouter l'annotation:

`@ComponentScan(basePackages = {"dev.spring", "services"})`

dans la classe principale pour indiquer à Spring de scanner à la fois les deux packages « **dev.spring** » et « **services** »

```
@SpringBootApplication
@ComponentScan(basePackages = {"dev.spring", "services"})
public class DemoApplication {
    public static void main(String[] args) {
        ApplicationContext context =
            SpringApplication.run(DemoApplication.class, args);
        Client client = context.getBean(Client.class);
        client.doSomething();
    }
}
```

@Bean et @Configuration

- Une autre approche pour déclarer les beans est d'utiliser une classe de configuration Java avec des méthodes annotées par **@Bean**.
- C'est **plus déclaratif** et donne un contrôle explicite sur quels objets Spring instancie et injecte.:.

```
package dev.spring.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import dev.spring.service.MonService;
import dev.spring.vue.Client;
@Configuration
public class AppConfig {
    @Bean
    public MonService monService() {
        return new MonService(); // Spring gère cette instance
    }
    @Bean
    public Client client(MonService monService) {
        // Spring injecte automatiquement le bean MonService en paramètre
        return new Client(monService);
    }
}
```

- Enlever les annotations **@Component** et **@Service** des classes des beans,
- Elles deviennent des classes POJOs simples

```
package dev.spring.service;

public class MonService
{
    public void execute()
    {
        System.out.println("Service exécuté !");
    }
}
```

```
package dev.spring.vue;
import org.springframework.beans.factory.annotation.Autowired;
import dev.spring.service.MonService;
public class Client
{
    private final MonService monService;
    // Injection de dépendance par constructeur
    @Autowired
    public Client(MonService monService)
    {
        this.monService = monService;
    }
    public void doSomething()
    {
        monService.execute();
    }
}
```