어노테이션 [`Bean`, Component, Configuration, Autowired]

주차	23주차
스터디 일자	@2024/04/05

어노테이션이란?

사전적 의미로는 주석이라는 뜻이다.

자바에서 Annotation은 주석처럼 쓰이며 특별한 의미, 기능을 수행하도록 하는 기술이다.

즉, 프로그램에게 추가적인 정보를 제공해주는 메타데이터라고 할 수 있다. (`meta data` : 데이터를 위한 데이터)

코드에 메타데이터를 작성하여 직관적인 코딩이 가능하게 만들어주며 이에 따라 생산성이 증대된다는 장점을 가지고 있다.

Spring Bean 등록 어노테이션

- **@Configuration**
- **@Component**
- **@Bean**

@Comfiguration

스프링 IOC 컨테이너에게 해당 클래스가 Bean구성 클래스임을 알려주는 어노테이션

@Component vs @Bean

두 어노테이션은 Spring IOC 컨테이너에 Bean을 등록하도록 하는 메타데이터를 기입하는 유사한 동작을 한다.

이 둘의 차이는 용도가 다르다는 것!

@Component

개발자가 직접 작성한/컨트롤이 가능한 **Class**를 Bean으로 등록하기 위한 어노테이션

@Component는 싱글톤 클래스 Bean을 생성한다. (싱글톤 클래스 : 한번에 하나의 객체(instance)만 가질 수 있는 클래스)

`@Scope("Prototype")` 어노테이션을 통해 **싱글톤이 아닌 빈**을 생성할 수도 있다.

@Service , @Repository 어노테이션 또한 이에 포함된다.

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

@Bean

개발자가 컨트롤이 불가능한 외부 라이브러리들을 Bean으로 등록하기 위한 어노테이션

> 예를 들어, 아래와 같은 MyClass의 경우 MyClass 클래스에 @Component를 선언할 수 없으니 MyClass의 인스턴스를 생성하는 메소드를 만들고 해당 메소드에 @Bean을 선언하여 Bean으로 등록한다.

→ 즉, 개발자가 생성한 **Class**에 @Bean 선언은 불가능하다.

```
@Configuration
public class ApplicationConfig {
  @Bean
  public MyClass myClass(String param){
      return new MyClass(param);
  }
}
```

@Autowired 어노테이션

@Component를 통해 등록된 빈을 다른 곳에서 의존성 주입할 때 사용하는 어노테이션

@Autowired를 통한 의존성 주입 방법

@Autowired를 통해 의존성을 주입하는 방법은 크게 세 가지이다.

1. **생성자 주입**
2. **수정자(Setter) 주입, 혹은 메서드 주입**
3. **필드 주입**

1. 생성자 주입

클래스의 생성자에 @Autowired를 붙여 해당 클래스 타입의 필드들을 찾아 등록하도록 한다.

아래의 경우 등록된 Class2의 빈을 알아서 찾아와 주입한다.

생성자가 하나일 경우 어노테이션을 붙이지 않아도 해당 생성자를 자동 주입의 대상으로 인식한다.

```
@Component
public class Class1 {

    private final Class2 field;

    // @Autowired (Optional) -> 생성자가 하나라서
    public Class1(final Class2 field) {
        this.field = field
    }
}
```

👍 **스프링에서 가장 권장되는 방법 !**

2. 수정자(Setter) 주입, 혹은 메서드 주입

메서드에 @Autowired를 붙여 해당 클래스 타입의 빈을 찾아와 의존성을 주입한다.

```
@Component
public class Class1 {

    private Class2 field;
```

```
    @Autowired
    public void setField(Class2 field) {
        this.field = field
    }
}
```

생성 시점 이후에 메서드를 호출하여 의존성을 변경할 수 있으나 !

⚠

- 필드가 외부에서 변경될 수 있다는 점,
- 의존성이 필요한 시점에 주입되지 않을 수 있다는 위험이 존재한다.

3. 필드 주입

가장 간단하게 주입하는 방법으로 필드에 @Autowired를 붙여 의존성을 주입하는 방법이다.

```
@Component
public class Class1 {

    @Autowired
    private Class1 field;

}
```

코드는 가장 간결하나 수정자 주입과 마찬가지로 필드가 final이 아니므로 변경될 여지가 있다.

메서드와 필드 주입 시의 메서드, 필드의 접근제어자는 private도 가능하다.

생성자 주입이 권장되는 이유

✨ **생성자 주입**은 수정자 주입, 필드 주입보다 여러 장점을 가진다.

- 필수적인 의존성의 주입 시점을 보장할 수 있다.
- 필드를 final로 선언이 가능하여 외부에서의 변경 가능성을 낮춘다.
- 의존성 주입이 한 곳(생성자)에서만 일어나므로 필요한 의존성을 한곳에서 모아서 확인할 수 있다.
- 순수 자바 코드로의 테스트가 용이하다.
- *Bean의 순환 참조 문제* 를 방지할 수 있다.

Bean의 순환 참조 문제?

스프링 자체적으로는 **생성자 주입에서만 순환 참조를 방지**하며, 스프링 부트 2.6 버전부터 필드 주입 또한 순환 참조를 발생시킨다.

```
@Component
public class A {

    @Autowired
    private B fieldB;

    private void methodA() {
        fieldB.methodB();
    }
}
```

```
@Component
public class B {

    @Autowired
    private A fieldA;

    private void methodB() {
        fieldA.methodA();
    }
}
```

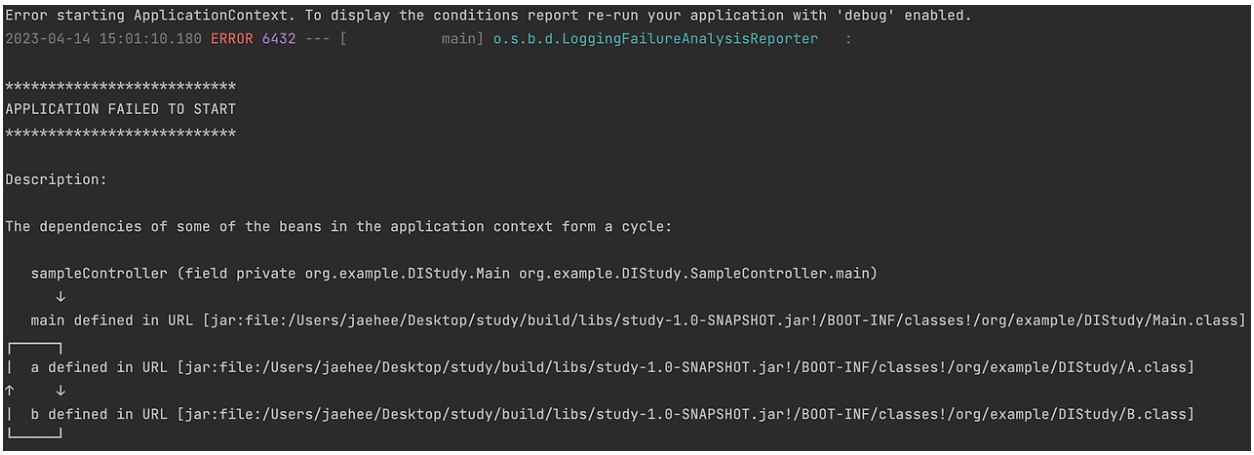
위 코드는 **컴파일 타임에서는 아무런 문제가 없지만,**

런타임에 `methodA` 혹은 `methodB`를 실행시킨다면 서로를 참조하며 메서드를 계속 실행시키면서 **StackOverFlow가 발생**, **런타임에 서버가 죽게 된다!**

이미 실행 중인 서버가 예상치 못하게 죽는 상황은 최악이다.

이러한 상황을 **생성자 주입을 통해 방지할 수 있다.**

위의 필드 주입을 모두 생성자 주입으로 바꾸고 서버를 실행시키면 아래와 같이 에러를 발생시키며 중단된다.



bean의 의존성이 순환 참조 문제를 발생시킨다는 내용

생성자 주입을 사용하면 **생성자를 호출하고 필요한 의존성을 주입하는 과정이 일련에 일어난다**.

따라서, 의존성 주입이 일어난 뒤 인스턴스를 생성하므로 서로를 의존하는 경우에는 **빈 등록 자체가 불가능**한 것이다.

반면, **필드 주입**의 경우 **인스턴스를 생성한 뒤에 의존성을 연결**하는데 이로 인해 **순환 참조**가 만들어진다.

따라서, 생성자 주입을 사용하면 아예 서버가 켜지지 않으므로 운영 중 서버가 꺼지는 사태를 방지할 수 있다 !

https://jaehee329.tistory.com/28

https://wildeveloperetrain.tistory.com/26