Spring week 5

유채린
chaerin.du.ub@gmail.com

해당 내용은 인프런 강의를 정리한 내용입니다.
잘못된 내용에 대하여 피드백주시면 감사하겠습니다!

(인프런 아이콘 클릭 시, 참고한 인프런 강의 사이트로 이동합니다.)

Index

- 싱글톤 컨테이너
- 컴포넌트 스캔
- 의존관계 자동 주입

싱글톤 컨테이너

싱글톤 패턴

- 클래스의 인스턴스가 JVM에 1개만 생성되는 걸 보장하는 디자인 패턴
  → private 생성자를 사용하여 외부에서 임의로 new 키워드를 사용하지 못하도록 한다.
  → static 영역에 객체 instance를 미리 하나 생성하고, public static 메서드인 getInstance()를 만들어 해당 메서드를 통해서만 조회하도록 허용한다.
- 문제점
  - 코드 자체가 많이 들어간다. (지저분한 코드)
  - 클라이언트가 구체 클래스에 의존한다. (getInstance() 호출): DIP 위반, OCP원칙 위반 가능성 증가
  - 테스트하기 어려움
  - 내부 속성 변경/초기화 어려움
  - private 생성자로 자식 클래스 만들기 어려움
  - 유연성이 떨어짐! (의존관계 주입(DI) 적용하기 어려움)

➔ 싱글톤 패턴은 안티패턴이라고 불리기도 한다.

싱글톤 컨테이너

- 스프링 컨테이너는 싱글톤 패턴의 문제점을 해결하면서 객체 인스턴스를 싱글톤으로 관리한다.

  (싱글톤 패턴 적용하지 않아도 가능)
  - 스프링 컨테이너는 싱글톤 컨테이너 역할을 하는데, 이렇게 싱글톤 객체를 생성하고 관리하는 기능을 싱글톤 레지스트리라고 한다.

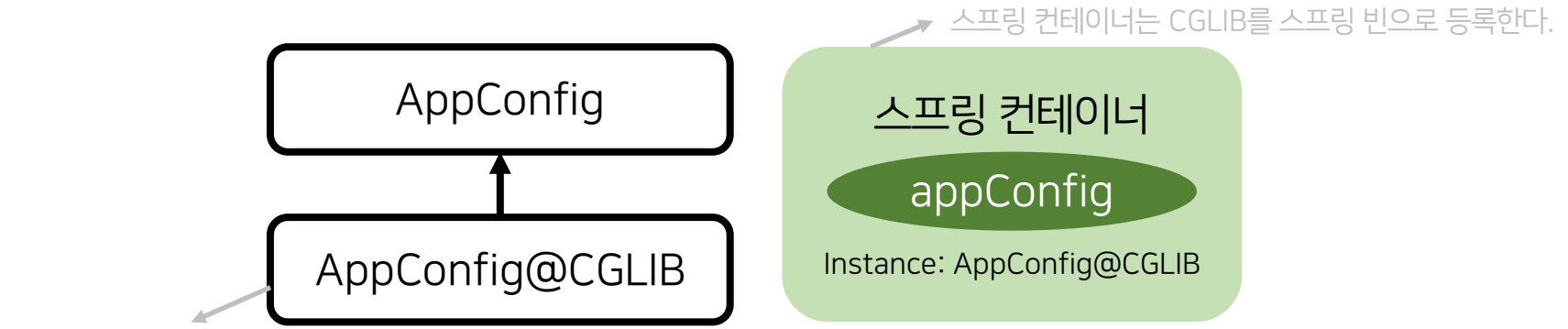
- 싱글턴 패턴 적용하지 않아도 객체 인스턴스를 싱글톤으로 관리한다.
  - 코드가 지저분하지 않고, DIP, OCP, 테스트, private 생성자로부터 자유롭게 사용가능!

- 스프링의 기본 빈 등록 방식 (95%)은 싱글톤

- 스프링 빈은 항상stateless(무상태)로 설계해야 한다.
  - 가급적 읽기만 하도록. 필드 대신 자바에서 공유되지 않는, 지역변수, 파라미터, ThreadLocal등을 사용해야 한다.

@Configuration@CGLIB

- @Configuration 애노테이션을 사용하면, 스프링이 CGLIB라는 바이트코드 조작 라이브러리를 사용해서 스프링 설정 정보 클래스(강의에서는 AppConfig)를 상속받은 임의의 다른 클래스를 만들고, 그 다른 클래스를 스프링 빈으로 등록한다.
  - 이 임의의 다른 클래스가 바로 싱글톤이 보장되도록 해준다.
- @Bean이 붙은 메서드마다 이미 스프링 빈이 존재하면 존재하는 빈을 반환하고, 스프링 빈이 없으면 생성해서 스프링 빈으로 등록하고 반환하는 코드가 동적으로 만들어진다.
- @Configuration 을 적용하지 않고, @Bean만 적용하게 되면 스프링 빈으로 등록되지만 싱글톤 보장은 되지 않는다.
  - 그러므로 스프링 설정 정보는 항상 @Configuration을 사용하자



컴포넌트 스캔

컴포넌트 스캔/의존관계 자동 주입

- @ComponentScan 설정 정보가 없어도 자동으로 스프링 빈을 등록하는 기능
  - @Component가 붙은 모든 클래스를 스프링 빈으로 등록
  - 이때 스프링 빈의 기본 이름은 클래스명을 사용하되 맨 앞 글자는 소문자로 바꿔준다.
  - ex) MemberServiceImpl -> memberServiceImpl
  - 스프링 빈 이름 직접 지정도 가능하다. @Component("mmm")

- @Autowired 의존관계 자동 주입하는 기능
  - 생성자에 @Autowired를 지정하면, 스프링 컨테이너가 자동으로 해당 스프링 빈을 찾아서 주입한다.
  - 이때 기본 조회 전략은 타입이 같은 빈을 찾아서 주입한다. (getBean(MemberRepository.class)와 동일)
  - 생성자에 파라미터가 많아도 다 찾아서 자동으로 주입한다.

- 같은 빈 이름 중복 시, 수동 빈이 자동 빈을 오버라이딩하게 되는데 잡기 어려운 버그이므로 application.properties에서 *spring.main.allow-bean-definition-overriding=true* 설정을 해주도록 한다.

컴포넌트 스캔 옵션

- basePackages

: 모든 자바 클래스 스캔 시, 시간이 오래 걸리므로 탐색 패키지 위치를 지정해준다.

  - 지정하지 않는 경우, @ComponentScan이 붙은 설정 정보 클래스의 패키지가 시작 위치가 된다.
  - 여러 시작 위치 지정할 수 있다.
  - 패키지 위치 지정하지 않고, 설정 정보 클래스의 위치를 프로젝트 최상단에 두는 걸 권장한다.

- 필터
  - includeFilters: 컴포넌트 스캔 대상을 추가로 지정 (거의 사용 안 함)
  - excludeFilters: 컴포넌트 스캔에서 제외할 대상을 지정

```
@Configuration
@ComponentScan(
        includeFilters = @Filter(type = FilterType.ANNOTATION, classes = MyIncludeComponent.class),
        excludeFilters = @Filter(type = FilterType.ANNOTATION, classes = MyExcludeComponent.class)
)
 static class ComponentFilterAppConfig {

}
```

- FilterType
  - ANNOTATION: 기본값, 애노테이션을 인식해서 동작한다. (생략 가능)
    ex) org.example.SomeAnnotation
  - ASSIGNABLE_TYPE: 지정한 타입과 자식 타입을 인식해서 동작한다.
    ex) org.example.SomeClass
  - ASPECTJ: AspectJ 패턴 사용
    ex) org.example..*Service+
  - REGEX: 정규 표현식
    ex) org₩.example₩.Default.*
  - CUSTOM: TypeFilter 이라는 인터페이스를 구현해서 처리
    ex) org.example.MyTypeFilter

ElementType
- TYPE: 클래스, 인터페이스, 열거타입에 애노테이션 붙일 수 있다.
- CONSTRUCTOR: 생성자에 애노테이션 붙일 수 있다.
- METHOD: 메소드에 애노테이션 붙일 수 있다.
- FIELD: 필드에 애노테이션 붙일 수 있다.

```
@Target(ElementType.TYPE)   // 해당 사용자가 만든 애노테이션이 붙일 수 있는 대상 지정 (클래스, 생성자, 메서드 등..)
@Retention(RetentionPolicy.RUNTIME) // 어느 시점까지 애노테이션의 타입을 어디까지 가져갈 지 설정
@Documented
@Indexed
public @interface Component {

    // annotation
    The value may indicate a suggestion for a logical component name, to be turned into a Spring bean in
    case of an autodetected component.
    Returns: the suggested component name, if any (or empty String otherwise)
    String value() default "";

}
```

```
/*
RetentionPolicy
SOURCE: 컴파일할 때, 해당 애노테이션의 메모리를 버림
CLASS: 컴파일 시 애노테이션의 메모리를 가져가지만 런타임시 사라짐 (디폴트)
RUNTIME: 런타임까지 사용 가능. JVM 런타임 종료까지 살아있음
*/
```

https://sas-study.tistory.com/329
https://blog.sunimos.me/12

컴포넌트 스캔 대상

- @Component
- @Controller

: 스프링 MVC 컨트롤러

- @Service

: 개발자에게 핵심 비즈니스 로직임을 인식

- @Repository

: 스프링 데이터 접근 계층(ex. JPA, JDBC)으로 인식, 데이터 계층의 추상화된 예외를 스프링 예외로 변환

- @Configuration

: 스프링 설정 정보 인식, 스프링 빈이 싱글톤 유지하도록 해 줌

- 애노테이션은 상속관계가 없으며, 스프링이 지원하는 기능이다.

의존관계 자동 주입

의존관계 주입 방법

- 생성자 주입
  - 생성자 호출 시점에서 딱 1번만 호출됨. 불편, 필수 의존관계
  - 생성자가 1개만 있는 경우, @Autowired 생략 가능
- 수정자 주입(setter 주입)
  - 선택, 변경 의존관계
- 필드 주입
  - 외부에서 변경이 불가능해서 테스트 하기 힘듦. 사용하지 말자
- 일반 메서드 주입
  - 일반적으로 잘 사용하지 않는다.

수정자 주입(Setter 주입) 옵션처리

```
1 @Autowired(required = false)
  public void setNoBean1(Member noBean1) { System.out.println("noBean1 = " + noBean1); }

2 @Autowired
  public void setNoBean2(@Nullable Member noBean2) { System.out.println("noBean2 = " + noBean2); }

3 @Autowired
  public void setNoBean3(Optional<Member> noBean3) { System.out.println("noBean3 = " + noBean3); }
```

1. 자동 주입할 대상이 없는 경우, 메서드 자체 호출이 안 되게 한다.
2. 자동 주입할 대상이 없는 경우, null이 입력된다.
3. 자동 주입할 대상이 없는 경우, Optional.empty가 입력된다.

```
noBean2 = null
noBean3 = Optional.empty

Process finished with exit code 0
```

최근에는 생성자 주입을 권장

- final 키워드를 사용하여, 생성자에서만 값을 넣어줄 수 있다.
- 값이 설정되지 않는 경우에는 컴파일 시점에서 막아주므로 오류를 빨리 잡아낼 수 있다.
- 오직 생성자 주입 방식만이 final 키워드를 사용할 수 있으므로 생성자 주입을 쓰도록 하자

- 프레임워크에 의존하지 않고, 순수한 자바 언어의 특징을 잘 살림
- 기본으로 생성자 주입을 사용하고, 필수 값이 아닌 경우에는 수정자 주입 방식을 옵션으로 부여하자
  (생성자 주입과 수정자 주입 동시에 사용 가능)

- Lombok library의 @RequiredArgsConstructor ➔ 생성자 1개일 때 생성자 선언 필요 없음
  - final이 붙은 필드(불변)를 모아서 생성자를 자동으로 만들어준다.
  - 롬복이 자바의 애노테이션 프로세서라는 기능을 이용해서 컴파일 시점에서 생성자 코드를 자동으로 생성

이름만 다르고 똑같은 타입의 스프링 빈 중복

1. @Autowired 필드명 매칭
   - 먼저 타입 매칭을 시도하고 그 결과 여러 빈이 있을 때 추가로 동작하는 기능으로, 필드 이름과 파라미터 이름으로 빈 이름을 추가 매칭한다.

2. @Qualifier
   - 추가 구분자를 붙여주는 방법, 빈 이름이 변경되는 건 아니다.
   - 못 찾을 경우, NoSuchBeanDefinitionException 예외 발생
   - 주입 받을 때, 모든 코드에 @Qualifier 붙여줘야 한다.

3. @Primary
   - 우선순위를 정하는 방법.
     @Autowired 시, 여러 빈 매칭되면 @Primary가 우선권을 가진다.

@Qualifier 예시

```
@Component
@Qualifier("mainDiscountPolicy")
public class RateDiscountPolicy implements DiscountPolicy {}
```

```
@Autowired
public OrderServiceImpl(MemberRepository memberRepository,
                        @Qualifier("mainDiscountPolicy") DiscountPolicy
discountPolicy) {
    this.memberRepository = memberRepository;
    this.discountPolicy = discountPolicy;
}
```

- 우선순위
  : @Qualifier > @Primary

감사합니다.

OCP Open/Closed Principle

개방-폐쇄 원칙

- 확장에는 열려 있지만 변경에는 닫혀 있어야 한다.
- **다형성** (Polymorphism) 활용해야 한다.

DIP dependency Inversion Principle

의존관계 역전 원칙

- 추상적인 것은 자신보다 구체적인 것에 의존하지 않고, 변화하기 쉬운 것에 의존해서는 안된다.
- 구현 클래스에 의존하지 말고, **인터페이스(추상화)에 의존하자**