Spring5
Chapter 03 스프링 DI

@스프링 의존 주입(DI)

- 의존이란 하나의 객체가 다른 객체없이 제대로 된 기능을 수행할 수 없는 것(객체의 변경에 의해 영향이 있는 것)
- 의존 주입(Dependency Injection)은 어떤 객체가 필요로 하는 객체를 외부에서 넣어 주는 것을 뜻한다

의존 주입을 왜 할까?

- 의존 주입을 하게 되면 주입을 받는 입장에서 어떤 객체가 들어오는지 신경 쓸 필요가 없다
- 외부에서 객체를 넣어 주기 때문에 주입할 객체를 변경해야 한다면 해당 객체를 생성하는 코드만 수정해주면 된다 (변경할 코드가 한 곳으로 집중됨)

스프링

- 의존 주입을 지원하는 조립기
- 필요한 객체를 생성하고, 생성한 객체에 의존을 주입한다

스프링에서 의존 주입

- **1) 스프링 설정 파일을 작성한다**

- 자바코드 혹은 XML 사용 (주로 자바코드로 설정 클래스 작성)
- 어노테이션 사용
- @Configuration – 해당 클래스가 스프링 설정 클래스임을 표시
- @Bean – 해당 메소드가 생성한 객체를 스프링 빈임을 표시

스프링에서 의존 주입

- **2) 스프링 컨테이너(IOC 컨테이너)를 생성한다**

- 스프링에서 실질적으로 빈을 생성하고, 사용하고, 생명 주기를 관리하고, 보관하는 공간이 스프링 컨테이너
- 책 48페이지~ '스프링은 객체 컨테이너'
- 일반적으로 BeanFactory, ApplicationContext를 포괄하여 스프링 컨테이너라 부른다
- 어노테이션을 이용해서 설정 클래스를 작성했기 때문에 AnnotationConfigApplicationContext를 사용하여 컨테이너 생성

스프링 컨테이너-BeanFactory

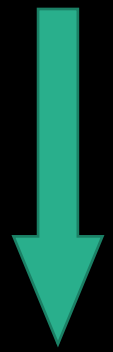
- 단순히 컨테이너에서 객체를 생성하고 의존 주입을 처리하는 기능을 제공한다
- 팩토리 디자인 패턴을 사용하며, 빈을 생성하고 분배를 책임진다

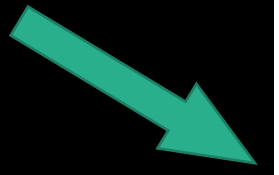
스프링 컨테이너-ApplicationContext

- BeanFactory의 기능을 포함하여 추가적인 기능을 제공
- 많이 사용되는 스프링 컨테이너
- 자바 코드 기반의 스프링 설정, 어노테이션을 사용한 빈 설정, 스프링 웹 개발, 트랜잭션 처리, 메시지 처리 등

의존 주입 방식-생성자

- 클래스가 생성될 때 의존 객체를 주입받는다



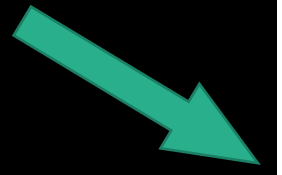


```
@Bean
public MemberDao memberDao() {
        return new MemberDao();
}
```

```
public MemberRegisterService(MemberDao memberDao) {
        this.memberDao = memberDao;
}
```

의존 주입 방식-setter

- Setter 메소드를 통해 의존 객체를 주입받는다



```java
@Bean
public MemberDao memberDao() {
        return new MemberDao();
}
```

```java
public void setMemberDao(MemberDao memberDao) {
        this.memDao = memberDao;
}
```

@Configuration 설정 클래스의 @Bean

```java
@Configuration
public class AppCtx {

	@Bean
	public MemberDao memberDao() {
		return new MemberDao();
	}

	@Bean
	public MemberRegisterService memberRegSvc() {
		return new MemberRegisterService(memberDao());
	}

	@Bean
	public ChangePasswordService changePwdSvc() {
		ChangePasswordService pwdSvc = new ChangePasswordService();
		pwdSvc.setMemberDao(memberDao());
		return pwdSvc;
	}
```

```java
@Configuration
public class AppCtx {

	@Bean
	public MemberDao memberDao() {
		return new MemberDao();
	}

	@Bean
	public MemberRegisterService memberRegSvc() {
		return new MemberRegisterService(memberDao());
	}

	@Bean
	public ChangePasswordService changePwdSvc() {
		ChangePasswordService pwdSvc = new ChangePasswordService();
		pwdSvc.setMemberDao(memberDao());
		return pwdSvc;
	}
```

- 스프링 컨테이너가 생성한 빈은 싱글톤 객체(단 하나의 객체)이다
- 즉, 다른 설정 메소드에서 memberDao()를 여러 번 호출하더라도 항상 같은 객체를 리턴한다는 것을 뜻한다
- 이는 스프링이 설정 클래스를 상속한 새로운 설정 클래스를 만들어서 사용하기 때문이다

두 개 이상의 설정 파일 사용하기

- 1) @Autowired

```java
@Configuration
public class AppConf1 {

	@Bean
	public MemberDao memberDao() {
		return new MemberDao();
	}

	@Bean
	public MemberPrinter memberPrinter() {
		return new MemberPrinter();
	}

}
```

```java
@Configuration
public class AppConf2 {
	@Autowired
	private MemberDao memberDao;
	@Autowired
	private MemberPrinter memberPrinter;
```

- 스프링 설정 클래스의 필드에 @Autowired 어노테이션을 붙이면 해당 타입의 빈을 찾아서 필드에 할당한다 (자동 주입 기능)

두 개 이상의 설정 파일 사용하기

- 1) @Autowired

- 스프링 컨테이너 생성자의 인자는 가변 인자이기 때문에 콤마로 구분해서 전달해준다

```
AbstractApplicationContext ctx =
                    new AnnotationConfigApplicationContext(AppConf1.class, AppConf2.class);


AppConf1 appConf1 = ctx.getBean(AppConf1.class);
```

두 개 이상의 설정 파일 사용하기

- 2) @Import
- 설정 클래스 AppConfImport.java

```
@Configuration
@Import({AppConf2.class})
public class AppConfImport {
```

- 배열을 이용해서 두 개 이상의 설정 클래스를 import할 수 있다
- @Import 어노테이션으로 AppConf2도 설정 클래스로 함께 사용할 수 있다
- 이 경우 스프링 컨테이너를 생성할 때 AppConf2 설정 클래스를 따로 지정해줄 필요가 없다

주입할 객체가 꼭 스프링 빈이어야 할까?

- 스프링 컨테이너가 제공하는 관리 기능이 필요 없고, getBean() 메소드로 구할 필요가 없으면 스프링 빈이 아니어도 된다

- 하지만 최근 스프링 부트가 나오면서 의존 자동 주입을 프로젝트 전반에 걸쳐 사용하기 때문에, 주입 대상은 모두 스프링 빈으로 등록하는 것이 보통이다