스프링 프레임워크 입문-백기선

2021년 7월 26일 월요일 오후 1:15

초기 실습 세팅 과정
이클립스에서 진행하려다가 뭔가에 막혀서 강의에서 사용하는 intellij 를 따라서 사용함
JDK 구버전 (1.8 or 11) 로 변경, 강의에서는 1.8버전으로 진행하라고 권장 (1.8인 경우 이클립스 실행 안되는 버전이라 타 블로그의 글을 보고 11로 진행하였는데 문제없었다)
프로젝트 경로에서 메이븐 빌드해줌
ex) C:\Users\skaja\IdeaProjects\spring-petclinic>mvnw package
main 폴더안에 들어있는 PetClinicApplication의 main run하여 정상 작동확인하였음

IOC 컨테이너
AOP
추상화 계층 (MVC, JDBC, Resource, O/X M, O/J M 등등)

일반적인 경우

```
class OwnerController{
        private OwnerRepository repo = new OwnerRepository();
}
```

=> OwnerController 는 OwnerRepository 이 있어야 작동

=> Inversion of Control (IOC)

```
class OwnerController{
        private OwnerRepository repo

        public OwnerController(OwnerRepository repo){
        this.repo = repo;
        }

        //repo 사용한다
}

class OwnerControllerTest{
        @Test
        public void create(){
                OwnerRepository repo = new OwnerRepository();
                OwnerController controller = new OwnerController(repo);
        }
}
```

누군가 생성자로 주겠지 라는 가정하

의존성 주입

```
        OwnerController controller = new OwnerController(repo);
    }
}
```

IOC

OwnerController 가 OwnerRepository를 사용할 때 OwnerRepository를 직접 만들어서 사용하는 것이 아니고, 누군가로 부터 주입받는다고 가정하고 생성자 작성 (주입해주는건 스프링, ApplicationContext 가 해줌)

왜 why? 테스트 코드를 작성하기 더 쉬움

IOC Container는 IOC 개념이 함유된 OwnerController 같은코드가 동작하게 해주는 역할

Owner Controller가 IOC Container 내부에 객체로 들어오고

IOC Container 내부에서 OwnerController 객체를 만들어준다

OwnerRepository 객체도 만들어줌

빈(컨테이너 내부에 만든 객체)들의 의존성을 관리해준다 (빈만, intellij 기준 class 왼쪽에 초록색 콩 아이콘 있으면 빈)

빈은 보통 @Controller, @Repository, @Component, @Service 같은 Annotation이 달려있으면 빈으로 취급

AplicationContext 가 빈들을 만들고, 빈들의 의존성을 엮어준다.

빈(Bean): 스프링 IOC 컨테이너가 관리하는 객체

OwnerRepository의 경우 빈이됨

pet 은 아님

빈만 빈을 관리할 수 있음

빈 등록 방법

1. component scanning
    a. @Component
        i. @Controller (Controller에 @Componet 적혀있음 이하동문)
        ii. @Repository
        iii. @Service
2. 직접 일일히 XML이나 자바 설정 파일에 등록
    a. ex)

    @bean (=> @Configuration 이라는 Annotation 있는 class 안에 정의해야함, @StringBootApplication 에 @Configuration 이 포함되있는 형태 등으로 존재함)

```
public String method(){
        return "hello";
}
```

어떻게 꺼내쓰는가?

1. @Autowired 혹은 @Inject
2. ApplicationContext에서 getBean()으로 직접 꺼내거나

의존성 주입(Dependency Injection) : 필요한 의존성을 어떻게 받아올 것인가
@Autowired/ @Inject를 어디에 붙일까?

1. 생성자
2. 필드
3. Setter

어떤 빈에 생성자가 오직 하나만 있고 생성자의 매개변수로 받는 타입의 빈이 존재한다면
@Autowired 가 안 붙어있더라도 의존성 주입

의존성에 대한 Setter를 가지고있다면 Setter에 붙이고
없다면 필드에 붙이자
why? => Setter 가 있다는 것은 Setter를 통해 의존성을 변경하겠다는 의미
아예 없는 경우는 굳이 Setter를 만들어서 할 필요까진??

AOP(Aspect Oriented Programming) : 흩어진 코드를 한곳에 모으자

```
class A{
	method a(){
		AAA
		오늘은 ~~
		BBB
	}
	method b(){
		AAA
		내일은 ~~
		BBB
	}
}
class B{
	method c(){
		AAA
		어제는 ~~
		BBB
	}
}

====================>

class A{
	method a(){
```

```
            오늘은 ~~
        }
        method b(){
            내일은 ~~
        }
}
class B{
        method c(){
            어제는 ~~
        }
}
class  AAABBB{
        method aaabbb(JoinPoint point){
            AAA
            point.execute()
            BBB
        }
}
```

구현에는 크게 2가지 기법

바이트 코드 (.class 파일) 조작 : 컴파일된 코드안에 끼워넣기

프록시 패턴 사용 : 상속 이용하여 내부적으로 처리

ex:

```
        class Aproxy extends A{
        }
```

PSA (Portable Service Abstraction)

나의 코드

<= (잘 만든 인터페이스 =PSA) 를 끼워넣으면 테스트나 코드 변경이 용이해짐

확장성이 좋지 못한 코드 or 기술에 특화된 코드