1일차

AOP

Aspect Oriented Programming

Add functionality to objects

Declaratively

- Logging, security, transactions, etc…

Spring projects

단순 프레임워크를 돕는 애드온이라고 봐도 된다.

Inversion of Control

The approach of outsourcing the construction and management of objects

Spring Container

Primary functions

Create and manage objects

Inject object’s dependencies

Spring 개발 절차

Spring Beans를 설정한 후

Spring 컨테이너 생성

스프링 컨테이너에서 Beans를 얻는다?

**스프링 프레임워크는 뒤에서 직접 코드를 만든다.**

Constructor Injection 개발절차

1. Defendency interface와 class 를 정의한다.

2. class의 injections을 위한 Constructor를 만든다.

3. Spring config file에 dependency injection을 수정한다.

**Question**

Why do we specify the Coach interface in getBean()?

For example:

Coach theCoach = context.getBean("myCoach", Coach.class);

**Answer**

Behaves the same as getBean(String), but provides a measure of type safety by throwing a BeanNotOfRequiredTypeException if the bean is not of the required type. This means that ClassCastException can't be thrown on casting the result correctly, as can happen with getBean(String).

<https://www.udemy.com/course/spring-hibernate-tutorial/learn/lecture/5215498#questions/3582594>

위에 링크가 시사하는 것 : 같은 property를 가진 파일은 동시에 로딩될 수 없다.

**Q. Setter injection vs Constructor injection**

Constructor-based or setter-based DI?

Since you can mix constructor-based and setter-based DI, it is a good rule of thumb to use constructors for mandatory dependencies and setter methods or configuration methods for optional dependencies. Note that use of the [@Required](https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#beans-required-annotation) annotation on a setter method can be used to make the property be a required dependency; however, constructor injection with programmatic validation of arguments is preferable.

The Spring team generally advocates constructor injection, as it lets you implement application components as immutable objects and ensures that required dependencies are not null. Furthermore, constructor-injected components are always returned to the client (calling) code in a fully initialized state. As a side note, a large number of constructor arguments is a bad code smell, implying that the class likely has too many responsibilities and should be refactored to better address proper separation of concerns.

Setter injection should primarily only be used for optional dependencies that can be assigned reasonable default values within the class. Otherwise, not-null checks must be performed everywhere the code uses the dependency. One benefit of setter injection is that setter methods make objects of that class amenable to reconfiguration or re-injection later. Management through [JMX MBeans](https://docs.spring.io/spring/docs/current/spring-framework-reference/integration.html#jmx) is therefore a compelling use case for setter injection.

Use the DI style that makes the most sense for a particular class. Sometimes, when dealing with third-party classes for which you do not have the source, the choice is made for you. For example, if a third-party class does not expose any setter methods, then constructor injection may be the only available form of DI.

Bean’s lifecycle : scope

명시적인 스코프가 없으면 singleton으로 가진다.

Singleton

즉, 우리가 코드상으로 getBean을 하더라도 똑 같은 레퍼런스 bean을 return한다.

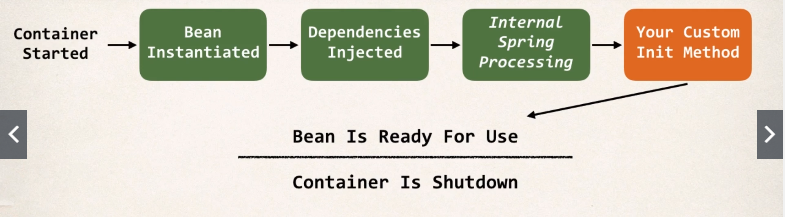
Prototype

매번 계속해서 새로운 class object reference를 return한다.

남은 타입은 웹서버를 위한 것이라 다음 시간ㅇ,,

Spring bean == Java Object

Bean lifecycle



Bean의 생명주기 중 파과와 탄생에서 custom code를 삽입 가능하다.

**Special Note about init and destroy Method Signatures**

When using XML configuration, I want to provide additional details regarding the method signatures of the init-method  and destroy-method .

메소드의 modifier는 아무 상관없음.

리턴 타입도 상관 없으나 return한 값을 확인하지 못하기 때문에 보통 “void”타입을 사용

이름 상관 없음.

Argument는 어떤 값도 받지 못하기 때문에 반드시 없어야 한다!

Autowriting example

코치 implementation에 필요한 ForutneService중에서

@Component를 Scan한다.

아무거나 닥치는데로 서비스를 찾아 넣는다.

Constructor Injection

4.3에서 생성자가 하나만 존재한다면 @Autowired를 명시하지 않아도 사용이 가능하다.

//@Autowired

1. public TennisCoach(FortuneService theFortuneService) {
2. System.out.println(" theFortuneService " + theFortuneService);

fortuneService = theFortuneService;

}

Autowired한 method순서는 코드 밑에 있는 것부터 시작하는 것으로 보아 stack으로 이루어진 것 같다.

Field Injection

에서는 JAVA Reflection이라는 것을 사용한다.

스프링을 공부하다가 보면 BeanFactory 라는 Spring Container 개념을 학습하게 된다.

이 BeanFactory는 어플리케이션이 실행한 후 객체가 호출 될 당시 객체의 인스턴스를 생성하게 되는데

그 때 필요한 기술이 Reflection이다.

자바는 스크립트 언어가 아닌 컴파일 언어이다. 물론 .java -> .class -> 실행이라는 2단계의 메커니즘을 가지고 있지만 컴파일 언어로 분리하는 게 옳다. 원래 자바에서는 동적으로 객체를 생성하는 기술이 없었다. 그리고 동적으로 인스턴스를 생성하는 Reflection으로 그 역활을 대신하게 된다.

출처: <https://gyrfalcon.tistory.com/entry/Java-Reflection> [Minsub's Blog]

Private field라 해도 Field injection이 가능하다.

Reflextion을통해 set한다.

Reflextion 사용에 대한 블로그

<https://codechacha.com/ko/reflection/>

3개의 injection 모드 같은 능력을 가지니 아무거나 원하는 거 일관서 있게 사용해라.

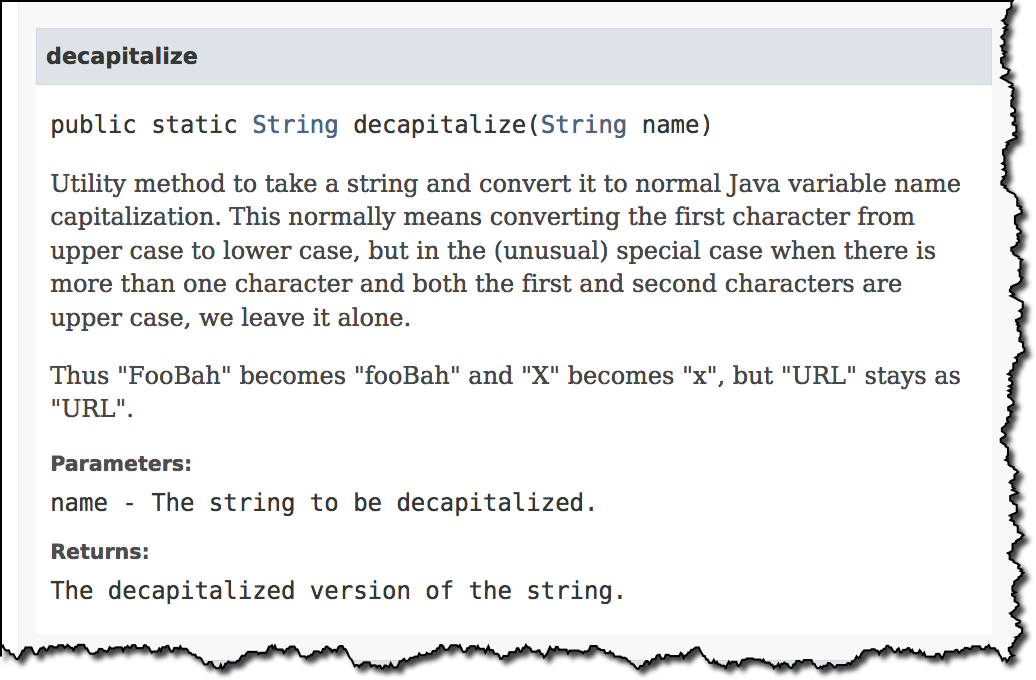
@Component  
public class TennisCoach implements Coach {  
 @Autowired  
 @Qualifier("happyFortuneService")  
 private FortuneService fortuneService;  
// So, the first thing Spring will do  
// is it will actually construct the class  
// by calling the default constructor  
// and then it will actually inject  
// a fortuneService implementation directly into this class  
// making use of some Java technology called Reflection.

Qualifier은 우리가 원하는 service 객체가 여러 개일 경우 여러 객체가 상속 받았을 경우 서브클래스 중 어떤 것을 결정할지 unique하게 해주는 Annotation이다.

Default Bean Names –

Default Bean Names는 특별한 rule을 따른다.

**Java Beans Introspector** 뒤에서는 이를 사용하여 작업한다. 아래는 key method에 대한 documentation 문서이다.



Constructor Injection에서는 특별한 @Qualifier 사용 룰이 주어지는데 이는 아래와 같은 코드로 진행되니 참고하자.

**@Autowired  
    public TennisCoach(@Qualifier("randomFortuneService") FortuneService theFortuneService) {**

**System.out.println(">> TennisCoach: inside constructor using @autowired and @qualifier");  
          
        fortuneService = theFortuneService;  
    }**

How to inject properties file using Java annotations

Config File

 <context:property-placeholder location="classpath:sport.properties"/>

Java class

@Value("${foo.email}")

private String email;

@Value("${foo.team}")

private String team;

Bean Scope

Singelton: bean defalult

Prototype 여러 개 생성 가능.

@Scope(“singleton”)

Prototype의 Lifecycle일 경우 preDestroy 불가ㅓ능.

이에 대한 stack overflow 의견

 It doesn't know when it is cleaned up, destroyed or what so ever. As such the @PreDestroy method isn't callable for prototype beans (as they do not have a clearly defined lifecycle like singletons or request scoped beans).

<https://stackoverflow.com/questions/20487685/spring-predestroy-method>

*Thus, although initialization lifecycle callback methods are called on all objects regardless of scope,****in the case of prototypes, configured destruction lifecycle callbacks are not called****. The client code must clean up prototype-scoped objects and release expensive resources that the prototype bean(s) are holding.*

*To get the Spring container to release resources held by prototype-scoped beans, try using a custom*[*bean post-processor*](https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#beans-factory-extension-bpp)*, which holds a reference to beans that need to be cleaned up.*

Java Config에서 @Bean Method는 이름 자체가 id로 인식된다는 것을 명심하자.

asdasdasd