Spring Framework 시작하기 2️⃣

Dependency Injection

IoC는 다양한 방법으로 만들 수 있습니다. 전략 패턴, 서비스 로케이터 패턴, 팩토리 패턴 그리고 의존관계 주입패턴들이 있습니다. 객체를 주입받는 패턴을 생성자 주입 패턴 (Dependency Injection) 이라고 부릅니다. 그 외에도 스프링은 세터 기반의 의존관계 주입도 지원합니다.

Dependency Resolution Process

Dependency Resolution Process

The container performs bean dependency resolution as follows:

- The `ApplicationContext` is created and initialized with configuration metadata that describes all the beans. Configuration metadata can be specified by XML, Java code, or annotations.
- For each bean, its dependencies are expressed in the form of properties, constructor arguments, or arguments to the static-factory method (if you use that instead of a normal constructor). These dependencies are provided to the bean, when the bean is actually created.
- Each property or constructor argument is an actual definition of the value to set, or a reference to another bean in the container.
- Each property or constructor argument that is a value is converted from its specified format to the actual type of that property or constructor argument. By default, Spring can convert a value supplied in string format to all built-in types, such as `int`, `long`, `String`, `boolean`, and so forth.

https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-dependency-resolution

Circular dependencies

A → B를 참조하고 B → A를 참조할경우 순환 의존관계가 형성되면서 `BeanCurrentlyInCreationException` 예외가 발생할 수 있습니다.

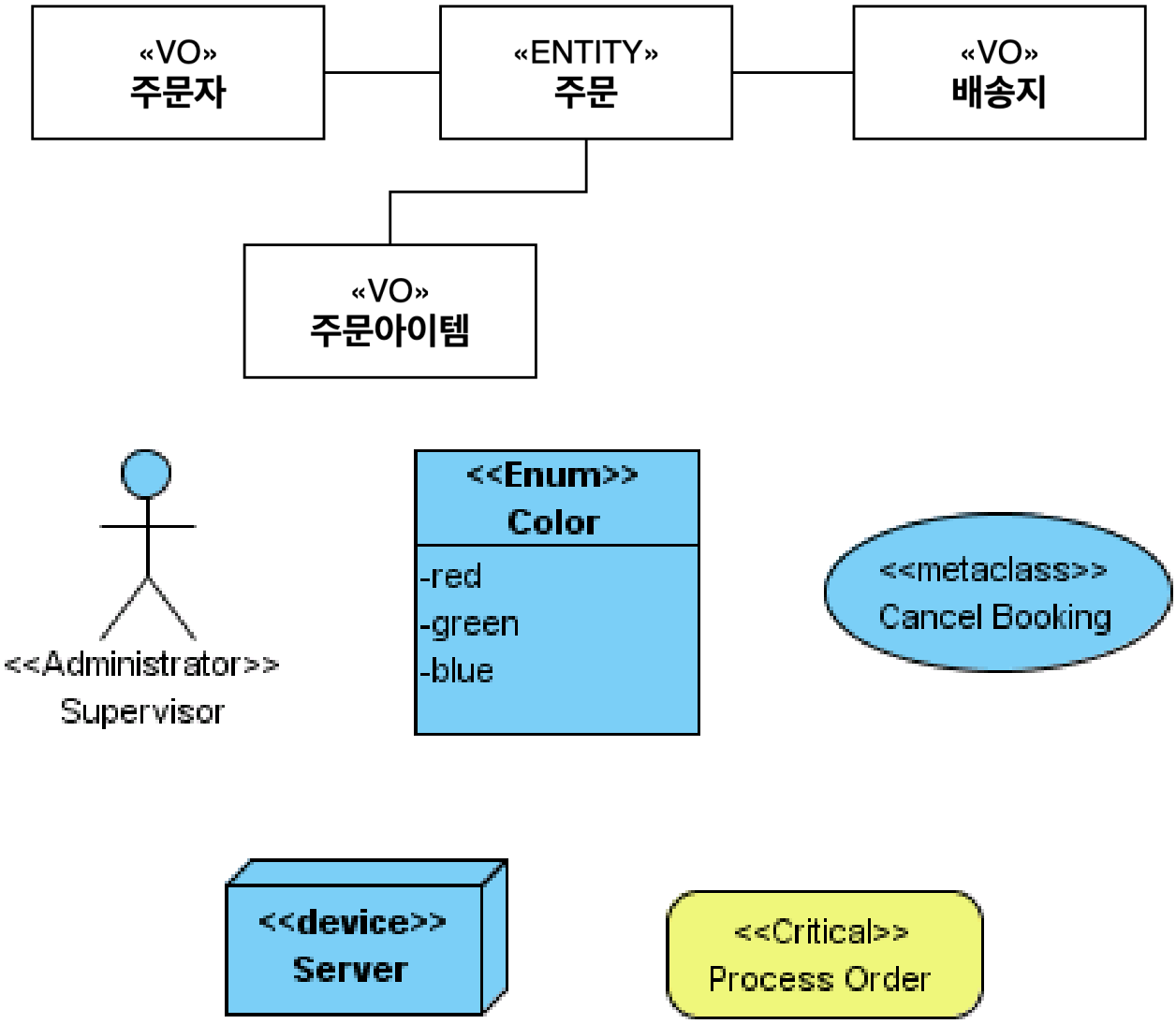
컴포넌트 스캔으로 빈 등록하기

컴포넌트 스캔은 스프링이 직접 클래스를 검색해서 빈으로 등록해주는 기능입니다. 설정 클래스에 빈으로 직접 등록하지 않아도 원하는 클래스를 빈으로 등록학 수 있습니다.

그러면 스프링은 자동으로 등록될 빈을 어떻게 찾을까요? 바로 Stereotype 애노테이션을 이용하면 스캔대성을 지정할 수 있습니다.

스트레오타입?

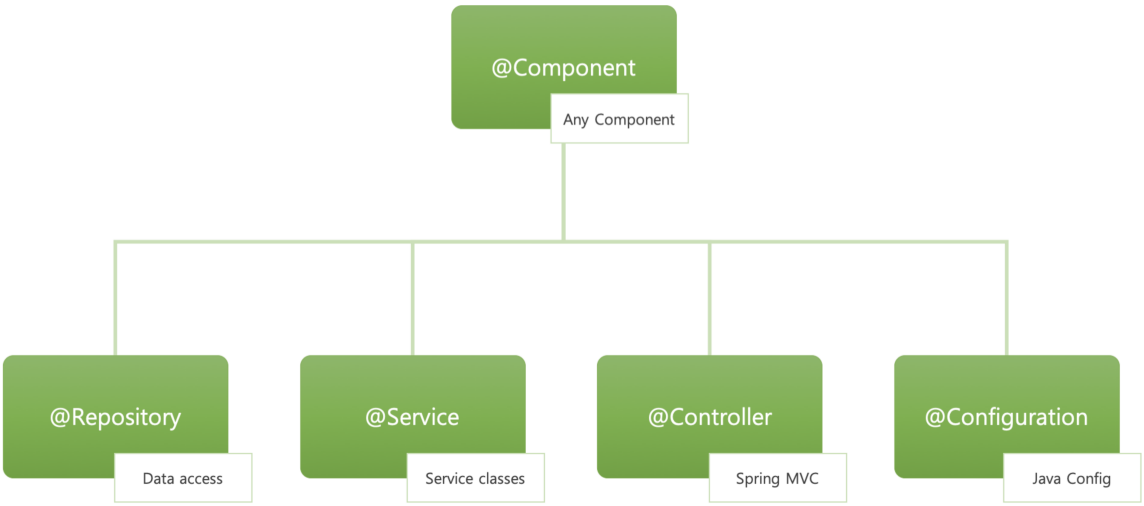
스트레오타입을 번역하면 고정관념인데 사실 이 용어는 UML에서 온거라고 볼 수 있습니다. 그림에서 보는것 처럼 UML다이어그램을 확장시켜주는 도구로서 특정 요소를 상황이나 도메인에 맞게 분류해주는 것입니다.



https://www.visual-paradigm.com/tutorials/how-to-create-stereotyped-model-element.jsp

스프링에서도 다양한 스트레오타입의 애노테이션이 있는데요 UML에서 처럼 모든 빈을 동일시 하지 않고 용도에 맞게 분류시켜주는 거고 스프링이 이미 여러 스트레오타입으로 분류를 해놓은 거라고 볼수있습니다.

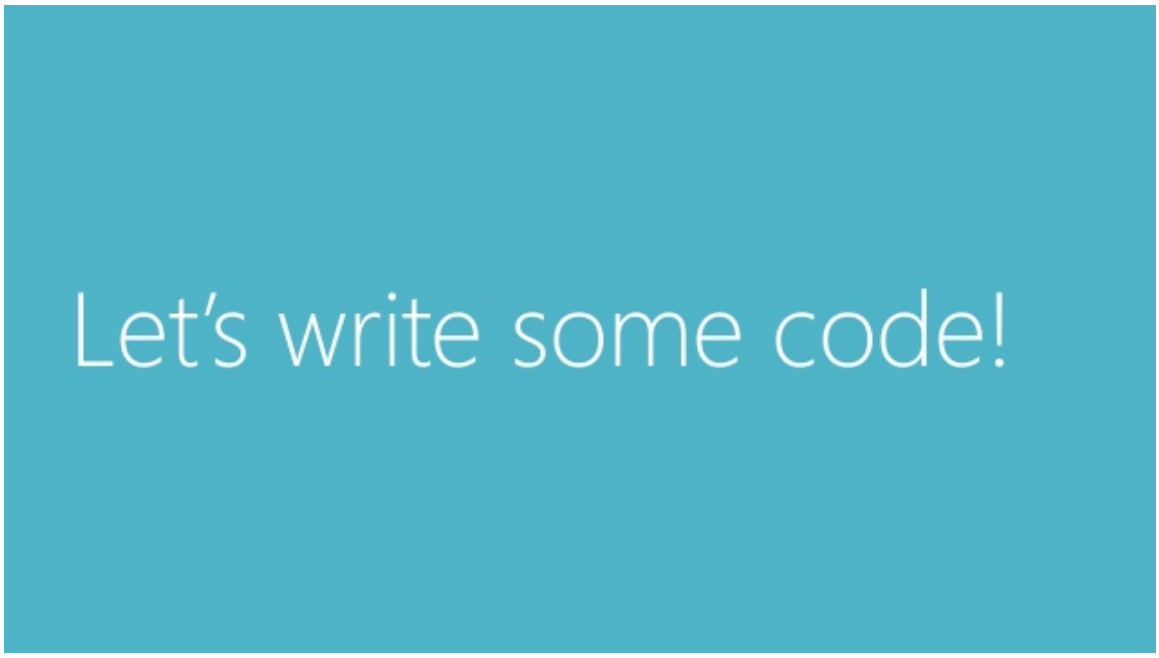
Spring 2.0 에서부터 @Repository 애노테이션이 등장하였고 Spring 2.5 부터 범용적으로 사용되었고 이때 @Component, @Controller, @Service, @Configuration 등이 등장하였습니다.



https://incheol-jung.gitbook.io/docs/q-and-a/spring/stereo-type

그럼 이제 스트레오타입 애노테이션을 써서 컴포넌트 스켄을 하도록 코드를 수정해 보겠습니다.

@Autowired 를 이용항 의존관계 자동주입



생성자 기반 의존관계 주입을 선택해야 하는 이유?

- 초기화시에 필요한 모든 의존관계가 형성되기 때문에 안전합니다.
- 잘못된 패턴을 찾을 수 있게 도와줍니다.
- 테스트를 쉽게 해줍니다.
- 불변성을 확보합니다.

스프링 공식문서에도 스프링 팀에서 조차 생성자 주입을 응호한다고 하고있습니다.

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 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.

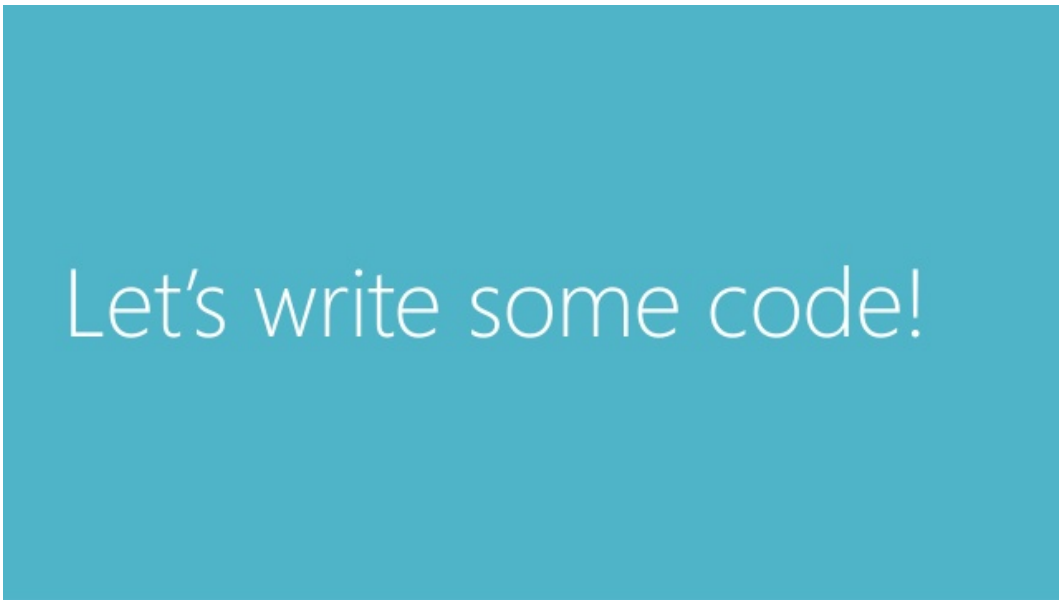
Setter injection should primarily only be used for optional dependencies that can be assigned reasonable default values within the class. Otherwise, not-null checks must be performed everywhere the code uses the dependency. One benefit of setter injection is that setter methods make objects of that class amenable to reconfiguration or re-injection later. Management through JMX MBeans 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.

https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-setter-injection

@Primary와 @Qualifier 애노테이션을 이용한 의존관계 설정

자동 주입이 가능한 빈이 두개 이상이면 어떤 빈이 주입되야하는지 선택해야합니다. 안그러면 스프링은 무슨 객체를 주입해야하는지 모르기 때문에 애러가 발생합니다.



복수 개의 빈 설정하기

@Configuration 애노테이션 또한 컴포넌트 스캔의 대상이 되기 때문에 설정을 여러 파일로 분리하고 메인이 되는 곳에서 컴포넌트 스캔으로 불러올 수도있다. 실제로 복잡한 상용 스프링 어플리케이션을 만들다보면 설정할 내용이 많기 때문에 여러 파일로 나누게 됩니다.

Bean Scope

스프링 공식문세어 보면 총 6개의 빈 스콥을 볼 수 있습니다. 기본적으론 싱글톤 스콥을 가집니다.

Table 3. Bean scopes

Scope	Description
singleton	(Default) Scopes a single bean definition to a single object instance for each Spring IoC container.
prototype	Scopes a single bean definition to any number of object instances.
request	Scopes a single bean definition to the lifecycle of a single HTTP request. That is, each HTTP request has its own instance of a bean created off the back of a single bean definition. Only valid in the context of a web-aware Spring `ApplicationContext`.
session	Scopes a single bean definition to the lifecycle of an HTTP `Session`. Only valid in the context of a web-aware Spring `ApplicationContext`.
application	Scopes a single bean definition to the lifecycle of a `ServletContext`. Only valid in the context of a web-aware Spring `ApplicationContext`.
websocket	Scopes a single bean definition to the lifecycle of a `WebSocket`. Only valid in the context of a web-aware Spring `ApplicationContext`.

https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-factory-scopes

Bean Lifecycle Callbacks

Bean 생성 생명주기 콜백

1. @PostConstruct 애노테이션이 적용된 메소드 호출
2. Bean이 InitializingBean 인터페이스 구현시 afterPropertiesSet 호출
3. @Bean 애노테이션의 `initMethod`에 설정한 메소드 호출

Baen 소멸 생명주기 콜백

1. @PreDestory 애노테이션이 적용된 메소드 호출
2. Bean이 DisposableBean 인터페이스 구현시 destroy 호출
3. @Bean 애노테이션의 `destroyMethod`에 설정한 메소드 호출