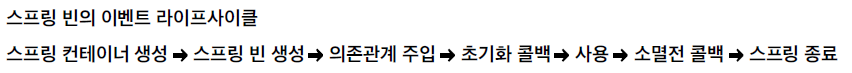
**3. Bean LifeCycle**

(김영한-스프링핵심원리 기본편 강의자료 8장 빈 생명주기 콜백, 9장 빈 스코프 및 spring 공식문서를 참조하여 작성하였습니다.)

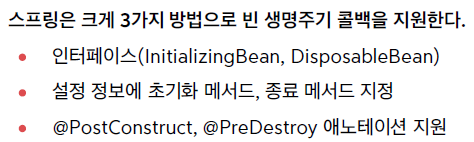
데이터베이스 커넥션 풀이나, 네트워크 소켓처럼 애플리케이션 시작시점에 필요한 연결을 미리 해두고, 애플리케이션 종료 시점에 연결을 모두 종료하려면 객체의 초기화와 종료작업이 필요.

스프링 빈의 초기화와 종료작업은 어떻게 진행되는가?

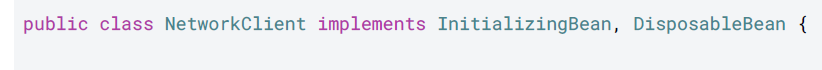


스프링 빈은 객체를 생성하고, 의존관계 주입이 다 끝나야 객체가 필요한 데이터를 사용할 수 있는 준비가 완료된다. 따라서 초기화 작업은 의존관계 주입이 모두 완료되고 난 다음에 호출해야한다. 개발자가 의존관계 주입이 모두 완료된 시점을 어떻게 알 수 있을까?

스프링은 의존관계 주입이 완료되면 스프링 빈에게 콜백 메서드를 통해서 초기화 시점을 알려주는 다양한 기능을 제공한다. 또한 스프링은 스프링 컨테이너가 종료되기 직전에 빈에게 소멸 콜백을 준다.



1) 인터페이스 구현 방식



위와 같이 InitializingBean, DisposableBean 인터페이스를 구현하도록 선언하고



afterPropertiesSet(), destroy() 메서드를 오버라이드하여 초기화와 종료작업을 수행할 수 있다.

더 나은 방법들이 있어서 지금은 사용되지 않는다.

2) @Bean 설정 정보 지정 방식

@Bean 어노테이션을 통해 수동으로 빈 등록 할 시 사용 가능한 방법.



위와 같이 개발자가 임의의 초기화, 종료작업 메서드를 구현하고,

’’’’

@Bean 어노테이션 내에 initMethod, destroyMethod 항목을 지정하여 자유롭게 설정하는 방법.

@Configuration

@Component(클래스) vs @Bean(메서드 위에)

3) @PostConstruct, @PreDestroy 어노테이션

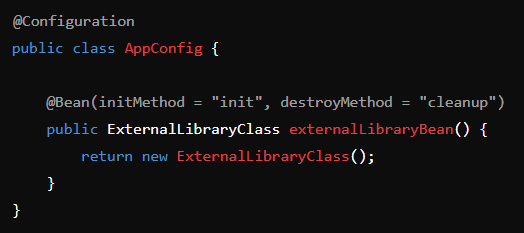
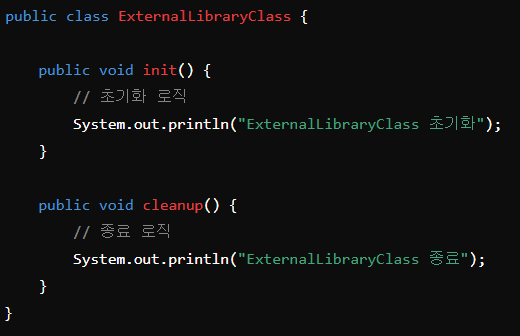


최신 스프링에서 가장 권장하는 방법. @Component 및 컴포넌트 스캔과 잘 어울림.

단점은 코드 수정이 불가능한 외부 라이브러리에는 적용하지 못한다는 것이다.

@Component 와 @Bean 어노테이션의 차이점?

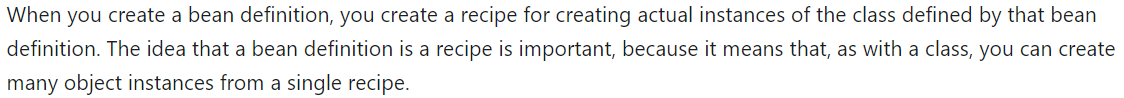
예를 들어, 내 어플리케이션에서 외부 라이브러리의 A라는 클래스를 사용하기 위해 스프링 빈으로 등록하는 상황 -> @Bean을 통해 수동 등록을 해야한다.



**4. Bean Scope**

Bean Scope란? 스프링 빈의 lifetime 즉, 스프링 빈이 존재할 수 있는 범위를 뜻한다.

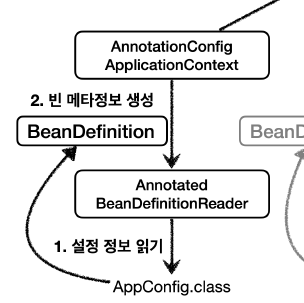
Spring Bean의 Default scope는 무엇일까요?



우리가 스프링 어노테이션을 활용하여 클래스를 작성하는 것

= Bean Definition을 생성하는 것

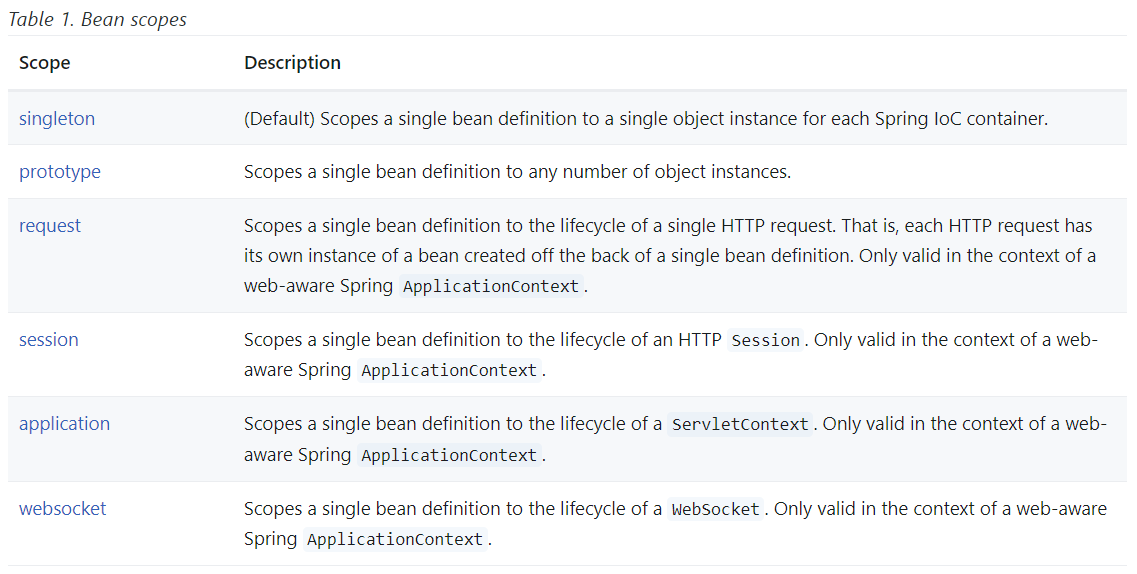
= 실제 객체의 recipe를 만드는 것

(DI 발표 내용 中)

You can control not only the various dependencies and configuration values that are to be plugged into an object that is created from a particular bean definition but also control the scope of the objects created from a particular bean definition. This approach is powerful and flexible, because you can choose the scope of the objects you create through configuration instead of having to bake in the scope of an object at the Java class level.

개발자는 Bean Definition을 통해 다양한 의존관계 및 설정 값 뿐만 아니라 Scope 또한 컨트롤 할 수 있다. 자바 클래스 수준에서 다루는 것이 아닌 Configuration을 통해 다룬다는 점은 굉장히 강력하고 유연하다.

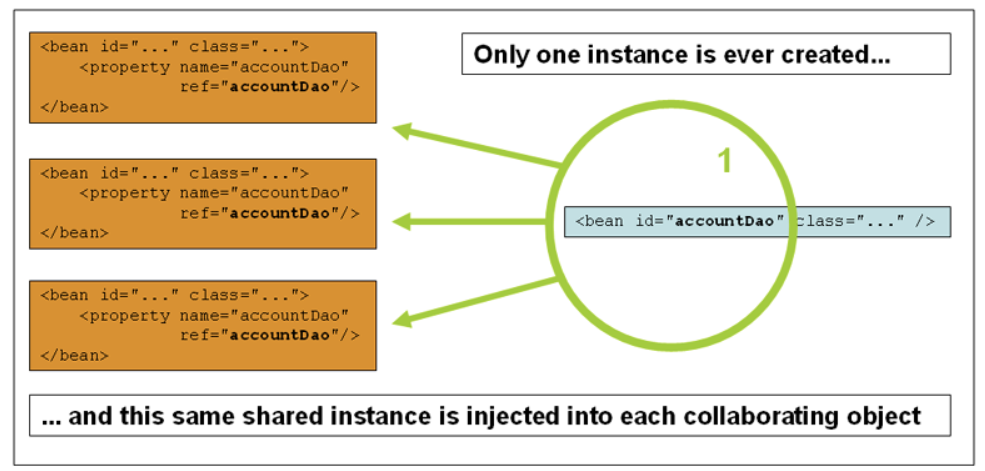
<빈 스코프의 종류>



Singleton과 Prototype을 제외한 나머지 스코프는 웹 기반의 애플리케이션에서만 사용이 가능하다. (정확히는 웹 인식 spring 컨테이너)

Singleton(싱글톤) Scope

: 각각의 Spring IoC 컨테이너에 대해 하나의 객체 만을 유지하며, 컨테이너의 시작과 종료까지 유지되는 가장 넓은 범위의 스코프이다.

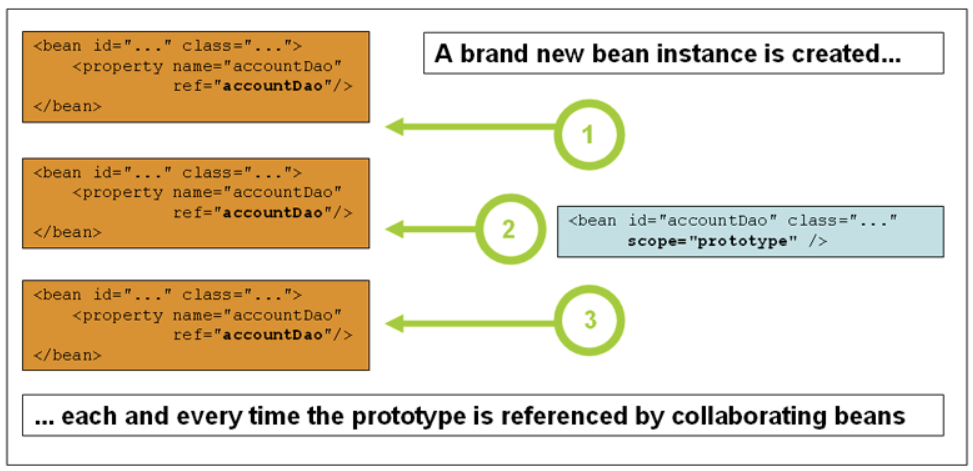


This single instance is stored in a cache of such singleton beans, and all subsequent requests and references for that named bean return the cached object.

컨테이너의 생성과 종료시점까지 싱글톤 빈을 관리하므로, 초기화 콜백 및 소멸전 콜백을 통한 초기화/종료 작업이 가능하다.

Prototype(프로토타입) Scope

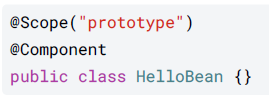
: 각각의 Spring IoC 컨테이너가 임의의 개수의 객체를 보유할 수 있다. 컨테이너는 빈의 생성과 의존관계 주입 및 초기화까지만 관여하고 더는 관리하지 않는 매우 짧은 범위의 스코프이다.



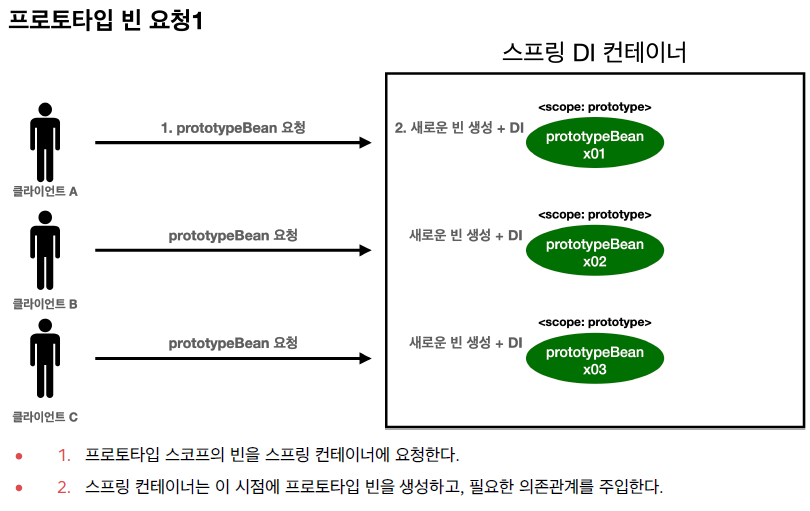
빈에 대한 요청이 존재할 때마다, 새로운 객체를 생성하여 반환한다. 요청할 때마다 새로운 빈이 주입되는 것이다. As a rule, you should use the prototype scope for all stateful beans and the singleton scope for stateless beans.

상태가 존재하는 빈은 prototype으로, 상태가 없는 빈은 singleton으로 사용하는 것이 좋다.

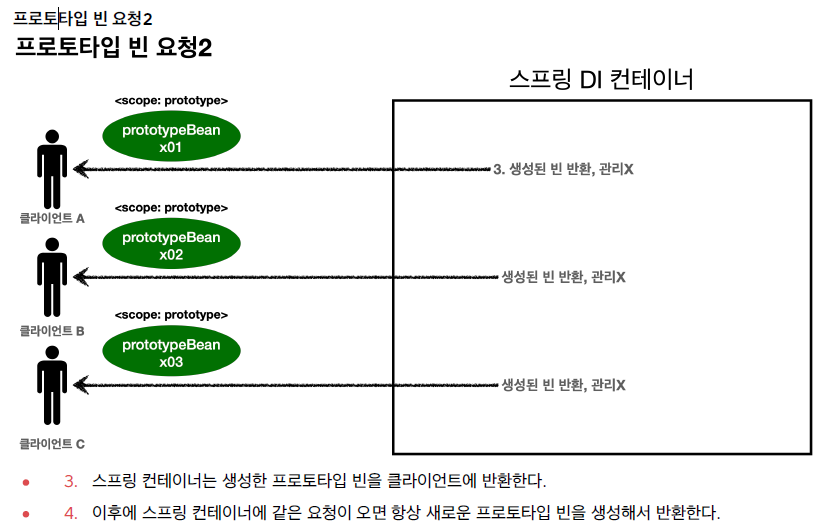
컨테이너가 빈의 생성과 의존관계 주입 그리고 초기화까지만 관여하므로 초기화 콜백 함수는 사용이 가능하나 소멸전 콜백 함수에의한 종료 작업은 불가능하다.







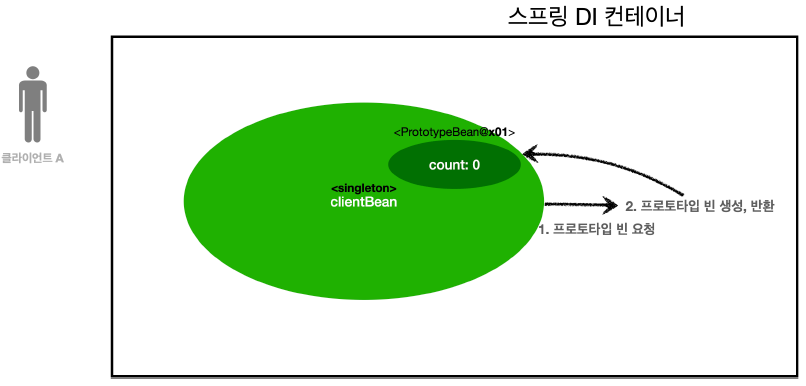
프로토타입 빈 생성에 필요한 의존관계도 매번 주입된다.



컨테이너에서 클라이언트에 빈을 반환한 후 더 이상 관리하지 않는다.

따라서 프로토타입 빈은 빈을 조회한 클라이언트가 관리해야하며, 종료작업 메서드 호출도 클라이언트가 해줘야한다.

**싱글톤 빈과 프로토타입 빈을 함께 사용할 때 발생할 수 있는 문제점**

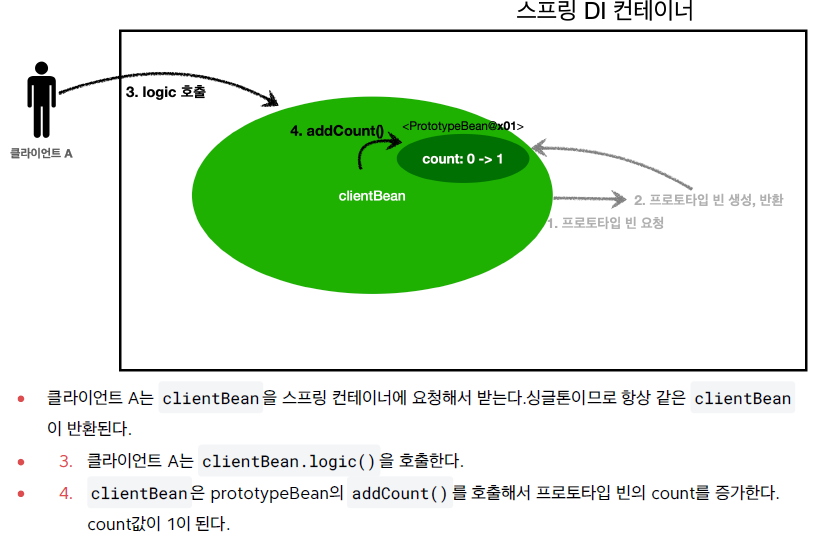


위와 같이 Singleton scope를 가진 clientBean 빈 내부에 Prototype Scope인 빈을 가지고 있는 상황을 생각해보자.

ClientBean은 싱글톤이므로, IoC 컨테이너 생성시점에 함께 생성되고, 의존관계 주입도 발생한다.

의존관계 주입 시점에 clientBean은 프로토타입 빈을 요청하고, 이 때 컨테이너는 새로운 빈을 생성하여 반환한다.

clientBean은 프로토타입 빈을 내부 필드에 보관한다. 정확히는 PrototypeBean의 참조값을 보관한다.



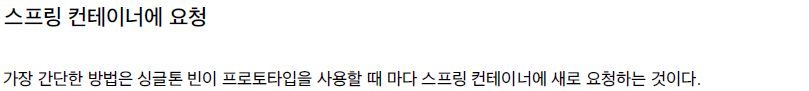
만약 클라이언트가 clientBean을 요청하여 컨테이너로부터 반환 받은 뒤, 프로토타입 빈을 사용하는 로직을 실행하면, 프로토타입 빈의 state가 변경된다.

이후에 또 다른 클라이언트가 clientBean을 반환받고, 프로토타입과 관련된 로직이 실행된다면, 변경된 state로 인한 의도치 않은 문제가 발생할 수 있다.

즉, stateless한 싱글톤 빈을 stateful하게 만드는 문제가 발생한다.

프로토타입 스코프 빈을 컨테이너에 요청하면 컨테이너는 매번 새로운 객체를 생성하여 반환해주지만, 싱글톤 타입 객체 내부에 존재하는 프로토타입 빈은 이미 과거에 주입이 끝난 빈으로, 매번 새롭게 생성되어 주입되지 않는다.

이를 해결하는 방법은 뭐가 있을까?





위와 같이 싱글톤 객체 내부의 프로토타입 빈을 사용하는 부분에서 매번 컨테이너에 새롭게 빈을 요청하여 처리하면 된다. 이러한 방식을 Dependency Lookup(DL)이라고 부른다.

하지만 이를 위해 결국 스프링 컨테이너인 ApplicationContext 객체를 주입받아야 한다.

이는 스프링 종속적인 코드가 되고, 단위테스트가 어렵다는 문제가 발생한다.

