스프링핵심원리-빈 스코프

2021년 8월 14일 토요일 오후 7:36

빈 스코프란

스프링 컨테이너의 시작과 함께 생성되어서 스프링 컨테이너가 종료될때까지 유지된다
why? => 스프링 빈이 싱글톤 스코프로 생성되기 때문, (스코프는 말 그래도 빈이 존재할 수 있는 범위)

스코프 종류

싱글톤 : 기본 스코프, 스프링 컨테이너의 시작과 종료까지 유지되는 가장 넓은 범위
프로토 타입 : 스프링 컨테이너는 프로토 타입 빈의 생성과 의존관계 주입까지만 관여, 매우 짧은 범위
웹 관련 스코프:
"request" : 웹 요청이 들어오고 나갈때 까지 유지되는 스코프
"session" : 웹세션이 생성되고 종료될 때까지 유지되는 스코프
"application" : 웹의 서블릿 컨텍스와 같은 범위로 유지되는 스코프

싱글톤 스코프 vs 프로토 타입 스코프

싱글톤 스코프의 빈 조회시 스프링 컨테이너는 항상 같은 인스턴스의 빈 반환
반면, 프로토 타입 스코프를 조회시, 항상 새로운 인스턴스 생성해서 반환

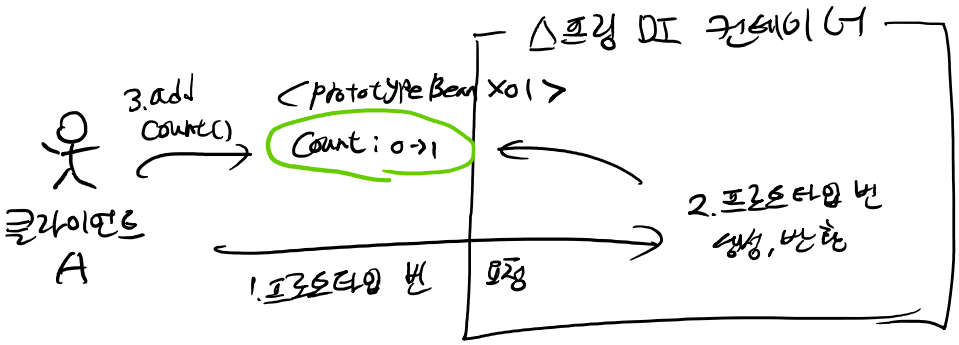
핵심은 **스프링 컨테이너는 프로타타입 빈을 생성하고, 의존관계 주입, 초기화까지만 처리한다는 것**
클라이언트에 빈을 반환하고, 이후 컨테이너는 빈을 관리하지 않음, 즉 관리 책임은 클라이언트에 있다는 것
(@PreDestory 같은 스프링 컨테이너가 호출하는 종료 메소드 자체가 호출 안됨)

싱글톤은 스프링 컨테이너 생성 시점에 초기화 메소드 실행
프로토 타입 스코프의 빈은 스프링 컨테이너에서 빈 조회될 때 생성되고 초기화 메소드도 실행됨

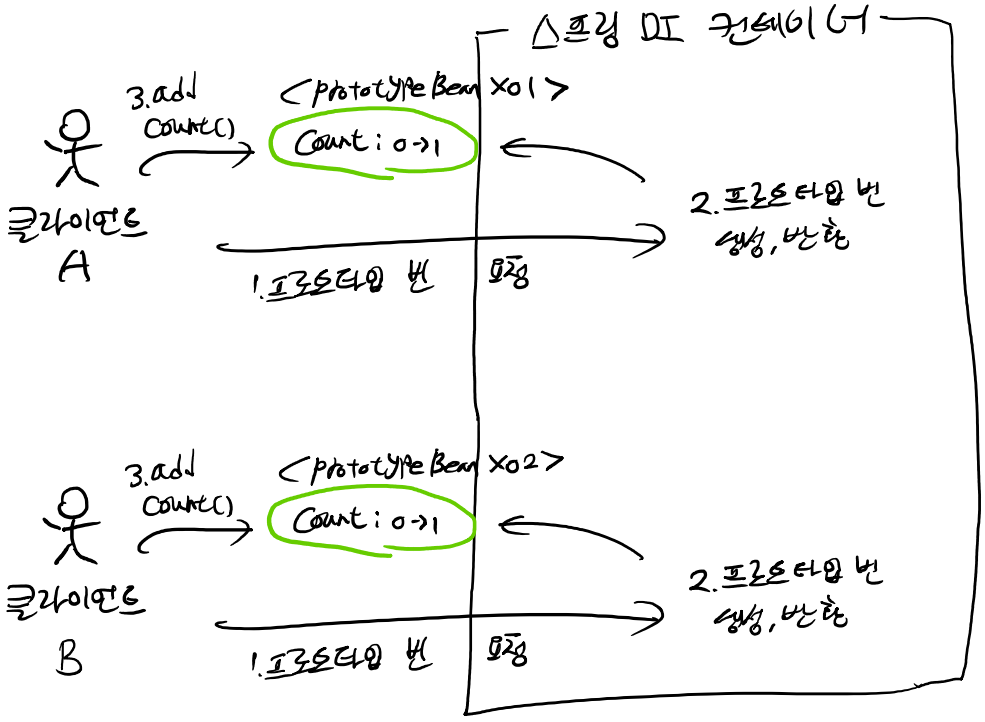
싱글톤 빈은 컨테이너가 관리하기 때문에 컨테이너 종료 시 빈의 종료 메소드 실행
프로토 타입 빈은 생성, 의존관계 주입, 초기화 까지만 관여하고 관리X 고로 종료메소드 실행X

프로토 타입 스코프와 싱글톤 빈을 함께 사용시 문제점

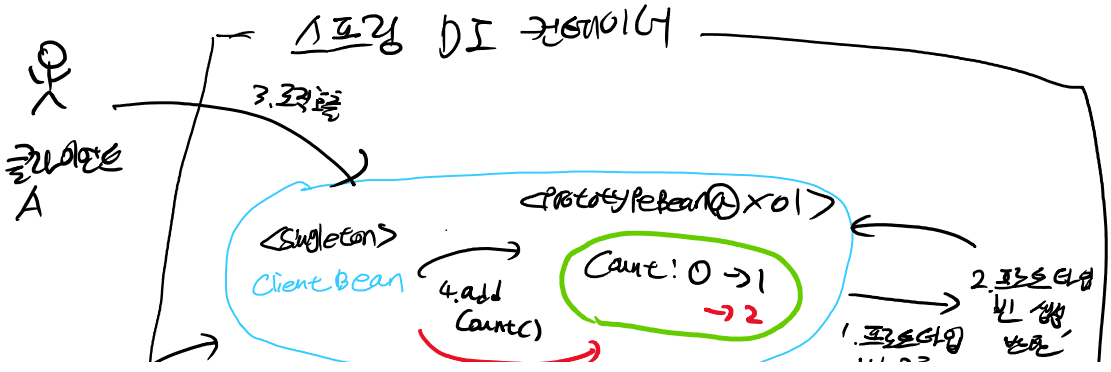
1. 프로토 타입 빈 직접 요청1

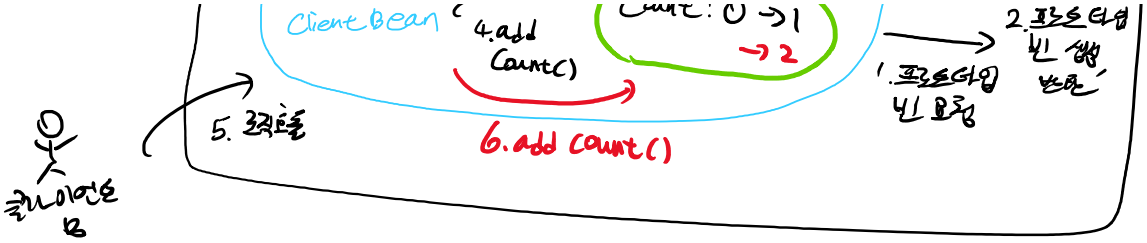


2. 프로토 타입 빈 직접 요청2



3. 싱글톤 빈에서 프로토 타입 빈 사용





clientBean 이 내부에 가지고 있는 프로토 타입 빈은 과거에 주입이 끝난 빈이기에 count 유지되어 있어 1에서 2로 증가함

의존관계를 외부에서 주입받는게 아니라 이렇게 직접 필요한 의존관걔를 찾는 것을 Dependency Lookup(DL) 의존관계 조회(탐색) 이라 함
이렇게 어플리케이션 컨텍스트 전체를 주입받게 되면, 스프링 컨테이너에 종속적인 코드가 되고 단위 테스트도 어려워짐
=> ObjectFactory, ObjectProvider(최신) : 지정한 빈을 컨테이너에서 대신 찾아주는 DL 서비스 제공
ObjectFactory : 기능이 단순, 별도의 라이브러리 필요X, 스프링에 의존
ObjectProvider : ObjectFactory + 편의 기능(상속, 옵션, 스트림 처리 등)

JSR -330 Provider - javax.inject.Provider 라는 자바 표준 사용하는 방법
자바 표준이고, 기능 단순 => 단위 테스트를 만들거나 mock 코드 만들기 편함
별도의 라이브러리 필요하다
자바표준이라서 스프링이 아닌 다른 컨테이너에서도 사용 가능

자바 표준인 JSR -330 Provider vs 스프링 제공인 ObjectProvider
ObjectProvider 이 DL을 위한 편의 기능을 많이 제공하고 스프링 외에 별도의 의존관계 추가 필요 없어서 편리함
다만 스프링이 아닌 다른 컨테이너에서도 사용할 수 있어야 하면 JSR-330 Provider 사용

특별한 이유 없으면 스프링이 제공하는 기능 사용하자

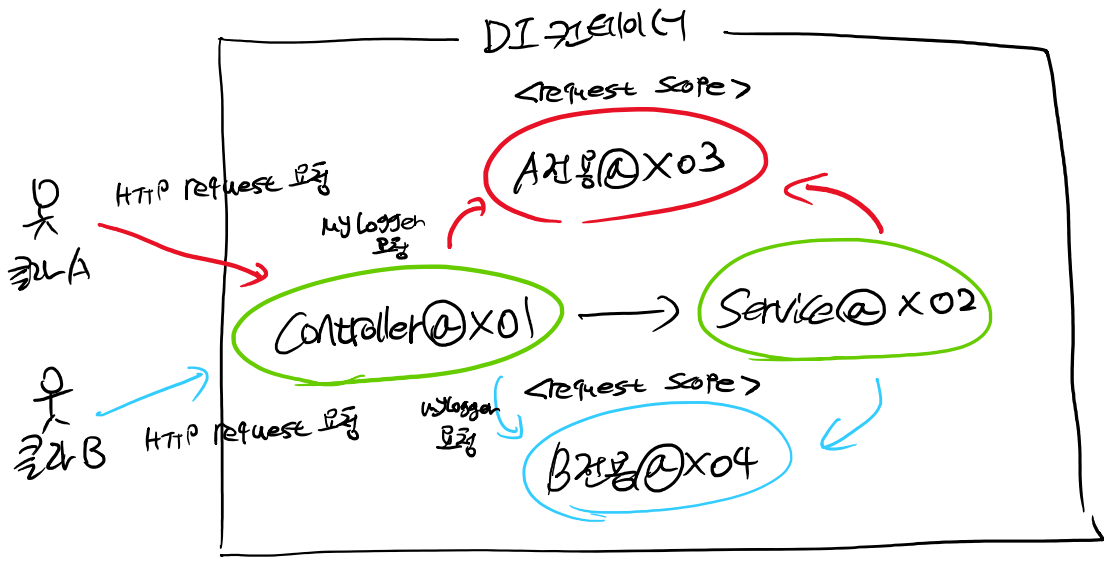
빈 스코프

웹 스코프 :

웹 환경에서만 동작
프로토 타입과 달리 스프링이 스코프의 종료 시점 까지 관리함 고로 종료 메소드 호출됨

웹 스코프 종류

- request : HTTP 요청 하나 들어오고 나갈 때 까지 유지되는 스코프, 각각의 HTTP 요청마다 별도의 빈 인스턴스가 생성되고, 관리된다.
- session : HTTP Session 과 동일한 생명 주기를 가지는 스코프
- application : 서블릿 컨텍스트와 동일한 생명 주기를 가지는 스코프
- websocket : 웹 소켓과 동일한 생명주기를 가지는 스코프



HTTP request 맞추어 각각 할당되는 것

```
@PostConstruct
public void init() {
    uuid = UUID.randomUUID().toString();
    System.out.println("[" + uuid + "]" + "request scope bean create : " + this);
}
```

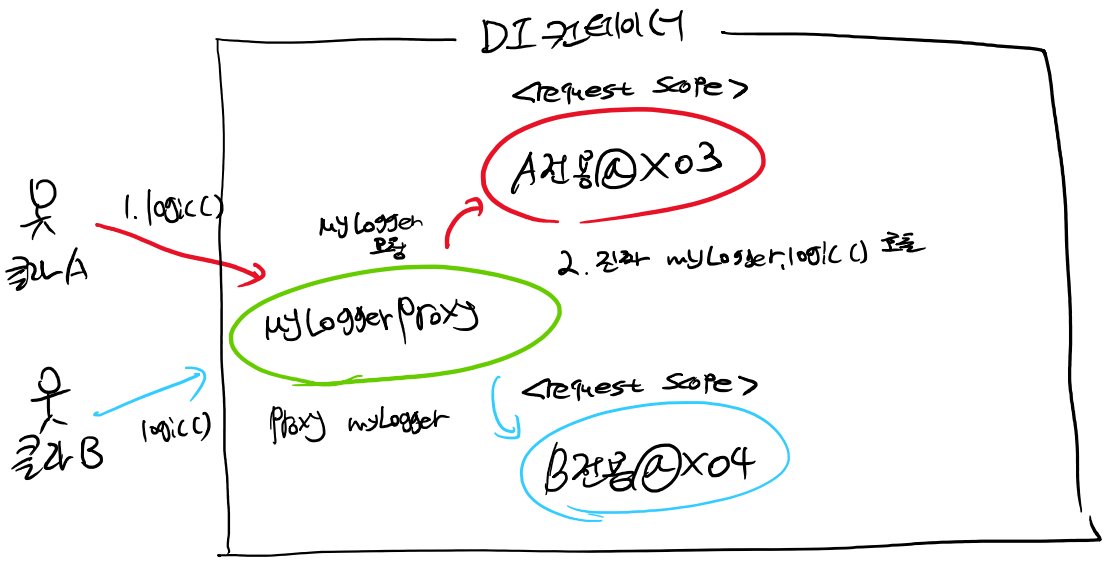
빈이 생성되는 시점에 자동으로 @PostConstruct 초기화 메소드를 사용해서 uuid를 생성해서 저장해둔다. 이 빈은 HTTP 요청당 하나씩 생성되므로, uuid를 저장해두면 다른 HTTP 요청과 구분 가능

request scope를 사용하지 않고 파라미터로 모든 정보를 서비스계층에 넘긴다면 파라미터가 너무 많아져서 지저분해짐 + 웹 정보가 웹에 관련없는 서비스 계층까지 넘어가게 됨
=> 웹과 관련된 부분은 컨트롤러까지만 사용해야... 서비스 계층은 웹 기술에 종속되지 않고 순수하게 유지해야 유지보수 관점에서 좋음

스코프와 프록시
proxyMode = ScopedProxyMode.TARGET_CLASS
적용 대상이 인터페이스가 아닌 클래스면 TARGET_CLSSS
인터페이스면 INTERFACES 선택
가짜 프록시 클래시 만들어두고 HTTP request 와 상관없이 가짜 프록시 클래스를 다른 빈에 미리 주입해 둘 수 있다.

proxyMode = ScopedProxyMode.TARGET_CLASS 를 설정하면 스프링 컨테이너는 CGLIB 라는 바이트 코드를 조작하는 라이브러리를 사용해서 가짜 프록시 객체를 생성한다

가짜 프록시 객체는 요청이 오면 그때 내부에서 진짜 빈을 요청하는 위임 로직이 들어있다
가짜 프록시 객체는 실제 request scope 와는 관계 없음, 싱글톤처럼 동작



프록시 객체 덕분에 클라이언트는 싱글톤 빈을 사용하듯 편리하게 request scope 사용할 수 있음
핵심 아이디어 = 진짜 객체 조회가 필요한 시점까지 지연처리한다는 점
단지 어노테이션 설정 변경만으로 원본 객체를 프록시 객체로 대체가능 (DI컨테이너와 다형성의 장점)
웹 스코프 아니라도 프록시는 사용 가능