**빈 생명주기 콜백**

스프링이 생성되거나 스프링이 죽기 전에 메소드를 호출해줄 수 있는 기능이 있음.

생성되어 초기화될 때 호출하고, 스프링이 소멸하기 직전에 안전하게 종료할 수 있는 메소드를 호출해주고 간단한 내용임.

세 가지 방식이 있고, 각 특징 별로 배울 것이 있음.

**1. 빈 생명주기 콜백 시작**

데이터베이스 커넥션 풀이나 네트워크 소켓처럼 애플리케이션 시작할 때 사전 연결을 미리하거나, 종료할 때 안전하게 종료를 해야함.

커넥션 풀 : 애플리케이션이 시작될 때 데이터베이스와 연결을 미리 맺어놓음. 그렇게 해야 tcp/핸드쉐이킹 할 때 걸리는 시간이 줄어들음. 커넥션을 미리 여러 개 만들어서 요청들어오면 바로 응답할 수 있게 함.

네트워크 소켓 : 서버가 뜰 때 다른 쪽과 네트워크를 미리 열어놔야함. -> 응답을 바로 줄 수 있음.

서버가 종료될 때 기타 안전 종료를 위한 작업들을 스프링이 해준다. -> 객체의 초기화와 종료 작업이 필요하다.

네트워크는 서버가 실행될 때 미리 연결해둬야하고, 서버가 종료될 때 안전하게 끊어져야하는 네트워크 객체가 있다고 가정하자. 네트워크 연결과 종료과정을 단순히 문자열로 표현할 것.

NetworkClient

connect() : 애플리케이션 시작 시점에 호출되어 네트워크 연결을 하는 메소드

disconnect() : 애플리케이션 종료 직전에 호출되어 네트워크 연결을 끊는 메소드

package hello.core.lifecyle;  
  
public class NetworkClient {  
 private String url;  
  
 public NetworkClient() {  
 System.*out*.println("생성자 호출, url = " + url);  
 connect();  
 call("초기화 연결 메시지");  
 }  
  
 public void setUrl(String url) {  
 this.url = url;  
 }  
  
 // 서비스 시작 시 호출  
 public void connect() {  
 System.*out*.println("connect:" + url);  
 }  
  
 public void call(String message) {  
 System.*out*.println("call: " + url + " message = " + message);  
 }  
  
 // 서비스 종료시 호출  
 public void disconnect() {  
 System.*out*.println("cloase: " + url);  
 }  
}

---

---

package hello.core.lifecyle;  
  
import org.junit.jupiter.api.Test;  
import org.springframework.context.annotation.AnnotationConfigApplicationContext;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
  
public class BeanLifeCycleTest {  
  
 @Test  
 public void lifeCycleTest() {  
 AnnotationConfigApplicationContext ac = new AnnotationConfigApplicationContext(LifeCycleConfig.class);  
 // ConfigurableApplicationContext 인터페이스를 사용하거나, AnnotationConfigApplicationContext 를 사용해야 .close메소드를 쓸 수 있다.  
 NetworkClient client = ac.getBean(NetworkClient.class);  
 ac.close();  
  
 }  
  
 @Configuration  
 static class LifeCycleConfig {  
 @Bean  
 public NetworkClient networkClient() {  
 NetworkClient networkClient = new NetworkClient();  
 networkClient.setUrl("http://hello-spring.dev");  
 return networkClient;  
 }  
 }  
}

NetworkClient 생성자 부분을 보면 url 정보 없이 connect 가 호출되는 것을 볼 수 있다.

객체를 생성하는 단계에는 url이 없고, 객체를 생성한 다음 외부에서 수정자 주입을 통해서 setUrl이 호출되어야 url 이 존재하게 된다.

스프링 빈은 간단하게 다음과 같은 라이프사이클을 가진다.

‘객체 생성’ -> 의존관계 주입.

(생성자 객체 생성은 예외, 생성자를 통한 객체 생성은 이미 만들어져있는 객체를 파라미터로 쓰기 때문에 주입과 동시에 객체가 생성이 된다.)

setter나 필드 주입과 같은 경우에는 객체 생성이후 의존관계 주입이 일어난다.

스프링 빈은 객체를 생성하고, 의존관계 주입이 다 끝난 다음에야 온전한 작업을 할 수 있다. 만약 객체를 생성하는 단계에서 필요한 데이터를 설정하거나 초기화하거나 사용하는 등의 작업을 하면, 오류가 날 수 있다. 객체 생성 및 의존 관계 주입이 모두 완료된 다음에 초기화 작업이 이루어져야 한다.

개발자는 의존관계 주입이 모두 완료된 시점을 어떻게 알지?

-> 스프링은 의존 관계 주입이 완료되면 **스프링 빈에게 콜백 메서드를 통해서 초기화 시점을 알려주는 기능을 제공**한다.

또한 **스프링은 스프링 컨테이너가 종료되기 직전에 소멸 콜백 메소드**를 호출한다.

-> 안전하게 종료작업을 진행할 수 있음.

스프링 빈의 이벤트 라이프사이클 (싱글톤의 경우)

스프링 컨테이너 생성 -> 스프링 빈 생성 -> 의존관계주입 -> 초기화 콜백 -> 사용 -> 소멸전 콜백 -> 스프링 종료

초기화 콜백 : 빈이 생성되고, 빈의 의존관계 주입이 완료된 후 호출

소멸전 콜백 : 빈이 소멸되기 직전에 호출

객체의 생성과 초기화 작업을 한 번에 하는 것이 좋지 않을까?

생성자 객체 생성으로 객체 생성과 주입, 초기화 작업을 한 번에 하는 것은 좋지 않다.

생성자는 필수 정보를 받고, 메모리를 할당해서 객체를 생성하는 책임을 가짐.

초기화는 이렇게 생성된 값들을 활용해서 외부 커넥션을 연결하는(실제 동작 과정이 들어감.) 등 무거운 동작을 수행한다. - ex) 외부 커넥션을 만들거나, 커넥션 풀을 만들거나 등.

유지보수 측면에서는 생성자 안에서 무거운 초기화 과정까지 한 번에 다 하는 것보다, 필수파라미터만 받고 객체의 생성에만 집중할 수 있게 만드는 것이 좋다.

단일 책임 원칙을 지키자.

객체를 생성하는 것과 초기화를 분리했을 때의 장점

외부 커넥션을 맺는 작업을 최초의 어떤 행위가 올 때까지 최대한 미룰 수 있음. 생성만 하고 기다리다가 액션이 오면 만드는 등의 작업을 할 수 있다.

스프링의 빈 생명주기 콜백 지원 3가지 방법

1) 인터페이스 (InitializingBean, DisposableBean)

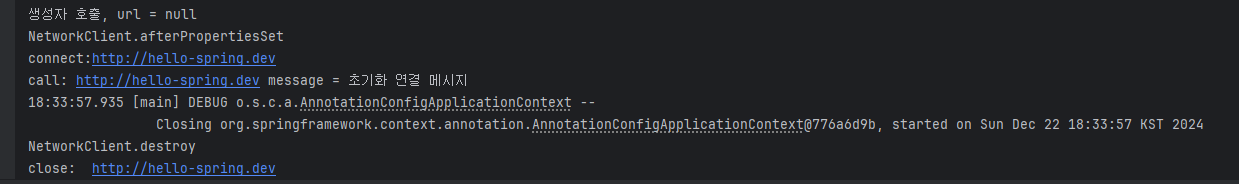
2) 설저 정보에 초기화 메소드, 종류 메소드 지정

3) @PostConstruct, @PreDestory 애노테이션 지원

1. 인터페이스로 콜백 지원 받음. -> 스프링 초창기에 나온 것이고, 거의 사용하지 않음.

InitializingBean 과 DisposableBean을 구현함.

-> initalizingBean의 afterPropertiesSet DisposableBean의 destroy를 overriding함



초기화, 소멸 인터페이스의 단졈

: 스프링 전용 인터페이스여서, 해당 코드가 스프링 전용 인터페이스에 의존하게 된다.

초기화, 소멸 메소드의 이용을 변경할 수 없다.

내가 코드를 고칠 수 없는 외부 라이브러리에 적용할 수 없다.

2. 빈등록 초기화 소멸 메소드.

@Bean(initMethod = "init", destroyMethod = "close") 을 통해 init 메소드와 destory 메소드 등록 가능

: 메소드 이름을 자유롭게 줄 수 있다는 장점이 있음.

스프링 빈이 스프링 코드에 의존하지 않음.

코드가 아니라 설정 정보를 사용하기 때문에 코드를 고칠 수 없는 외부 라이브러리에도 초기화 ,종료 메소드를 적용할 수 있다.

@Bean으로 등록할 때만 발생함.

Bean의 destroyMethod 속성에는 아주 특별한 기능이 있다.

라이브러리는 대부분 close나 shutdown이라는 이름의 종료 메서드를 사용한다.

@Bean의 destroyMethod는 기본값이 {inferred} 으로 등록되어있는데,

이 추론 기능은 ‘close’,’shutdown’이라는 이름의 메서드를 자동으로 호출해준다.

따라서 직접 스프링 빈으로 등록하면 종료 메서드는 따로 적어주지 않아도 잘 동작할 때도 있다.

추론 기능을 사용하기 싫으면 destoryMethod=”” 처럼 빈 공백을 지정하면 된다.

AutoCloseable 클래스

try-with-resources는 try()문에 선언된 객체들에 대해서 try가 종료될 때 자동으로 자원을 해제해주는 기능. try에 선언된 객체가 AutoCloseable 을 구현했더라면 try구문이 종료될 때 객체의 close 메소드를 호출해준다.



Try catch finally사용



Try with resources 사용

Try-with-resources가 모든 객체의 close()를 호출해주지는 않습니다. AutoCloseable을 구현한 객체만 close()가 호출됩니다.

3. 애노테이션 @PostConstruct, @PreDestroy 사용 ( 이것을 쓰면 됨.)

@PostContruct와 @PreDestory는 javax.annotation(네임스페이스가 jakatra로 변경됨) 에 속해있는 것으로 자바에서 지원해주기 때문에 스프링이 아닌 다른 컨테이너를 사용하더라도 호환이 된다.

<https://www.samsungsds.com/kr/insights/java_jakarta.html> (Jakatra 참고 스프링 3버전부터는 javax가 아닌 jakatra 사용

최신 스프링에서 가장 권장하는 방법이다.

컴포넌트 스캔과 잘 어울린다. (빈 등록하는 것이 아니기 때문에 수동 빈 등록이 아니더라도 자동 등록 가능.)

유일한 단점은 외부 라이브러리에는 적용하지 못한다. 외부 라이브러리를 초기화, 종료 해야하면 @Bean의 기능을 사용하자.

정리 : PostContruct 와 @PreDestroy를 사용하자. 외부 라이브러리의 경우는 @Bean의 initMethod, destroyMethod를 사용하자.