**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

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