[ 05 ] 생명주기와 범위

1. 스프링 컨테이너 생명 주기
   * 스프링 컨테이너 생성 : GenericXmlApplicationContext ctx = new GenericXmlApplicationContext();
   * 스프링 컨테이너 설정 : ctx.load(“classpath:applicationCTX.xml”); ctx.refresh();
   * 스프링 컨테이너 사용 : Student st = ctx.getBean(“student”,Student.class); st.getName();
   * 스프링 컨테이너 소멸(자원해제) : ctx.close();

(예제)

Student.java

**package** com.ch.ex;

**public** **class** Student {

**private** String name;

**private** **int** age;

**public** Student(String name, **int** age){

**this**.name = name;

**this**.age = age;

}

**public** String getName() {**return** name;}

**public** **int** getAge() {**return** age;}

}

applicationCTX.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd"*>

<bean id=*"student"* class=*"com.ch.ex.Student"*>

<constructor-arg value=*"홍길동"*/>

<constructor-arg value=*"30"*/>

</bean>

</beans>

MainClass.java

**package** com.ch.ex;

**import** org.springframework.context.support.GenericXmlApplicationContext;

**public** **class** MainClass {

**public** **static** **void** main(String[] args) {

GenericXmlApplicationContext ctx = **new** GenericXmlApplicationContext(); // 생성

ctx.load("classpath:applicationCTX.xml"); //설정

ctx.refresh();

Student student = ctx.getBean("student", Student.**class**);//사용

System.***out***.println("이름 : " + student.getName());

System.***out***.println("나이 : " + student.getAge());

ctx.close(); //소멸

}

}

1. 스프링 빈 생명 주기
   * Implements EnvironmentAware – 인터페이스를 이용한 생명주기
     + 1. EnvironmentAware를 implements하면 오버라이드한 setEnvironment() 생성 : 빈이 만들어지기 전 수행 – 환경변수에 대한 변경이 생길 시에만 호출됨
   * Implements InitializingBean, DisposableBean – 인터페이스를 이용한 생명주기 관리 방법
     1. InitializingBean을 implements하면 오라바이드한 afterPropertiesSet() 생성 : 빈 초기화 과정(ctx.refresh())에서 호출됨.
     2. DisposableBean 을 implements하면 오버라이드한 destroy() 생성 : 빈 소멸과정(ctx.close())에서 호출됨.

* ctx.close()의 경우 컨테이너가 소멸 하는 단계. 컨테이너가 소멸 하면, 빈은 자동 소멸 됩니다.
  + @PostConstruct, @preDestroy – 어노테이션을 이용한 생명주기 관리 방법

(예제)

Student.java

**package** com.ch.ex;

**import** org.springframework.beans.factory.DisposableBean;

**import** org.springframework.beans.factory.InitializingBean;

**public** **class** Student **implements** InitializingBean, DisposableBean{

**private** String name;

**private** **int** age;

**public** Student(String name, **int** age){

**this**.name = name;

**this**.age = age;

}

**public** String getName() {**return** name;}

**public** **int** getAge() {**return** age;}

@Override

**public** **void** afterPropertiesSet() **throws** Exception {

System.***out***.println("afterPropertiesSet"); // 빈이 생성될 때(초기화될 때) 호출되는 메소드

}

@Override

**public** **void** destroy() **throws** Exception {

System.***out***.println("destroy"); //빈이 소멸하는 과정에서 호출되는 메소드

}

}

**OtherStudent.java**

**package** com.ch.ex;

**import** javax.annotation.\*;

**public** **class** OtherStudent {

**private** String name;

**private** **int** age;

**public** OtherStudent(String name, **int** age){

**this**.name = name;

**this**.age = age;

}

**public** String getName() {**return** name;}

**public** **int** getAge() {**return** age;}

@PostConstruct

**public** **void** initMethod(){

System.***out***.println("OtherStudent - initMethod");

}

@PreDestroy

**public** **void** destroyMethod(){

System.***out***.println("OtherStudent - destroyMethod");

}

}

applicationCTX.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context-3.2.xsd"*>

<context:annotation-config/>

<bean id=*"student"* class=*"com.ch.ex.Student"*>

<constructor-arg value=*"홍길동"*/>

<constructor-arg value=*"30"*/>

</bean>

<bean id=*"otherStudent"* class=*"com.ch.ex.OtherStudent"*>

<constructor-arg value=*"홍길순"*/>

<constructor-arg value=*"50"*/>

</bean>

</beans>

**MainClass.java**

**package** com.ch.ex;

**import** org.springframework.context.support.GenericXmlApplicationContext;

**public** **class** MainClass {

**public** **static** **void** main(String[] args) {

GenericXmlApplicationContext ctx = **new** GenericXmlApplicationContext();

ctx.load("classpath:applicationCTX.xml");

ctx.refresh();

ctx.close();

}

}

1. 스프링 빈 범위
   * 스프링 컨테이너가 생성되고, 스프링 빈이 생성될 때, 생성된 스프링 빈은 scope를 가지고 있다. 범위란 객체가 어디까지 영향을 미치는지 결정하는 것.

(예제)

<bean id=*"student"* class=*"com.ch.ex.Student"* scope=*"singleton"*>

<constructor-arg value=*"길동"*/>

<constructor-arg value=*"10"*/>

</bean>

**package** com.ch.ex;

**import** org.springframework.context.support.GenericXmlApplicationContext;

**public** **class** MainClass {

**public** **static** **void** main(String[] args) {

GenericXmlApplicationContext ctx = **new** GenericXmlApplicationContext();

ctx.load("classpath:applicationCTX.xml"); //컨테이너 설정

ctx.refresh();

Student student1 = ctx.getBean("student", Student.**class**);

student1.out();

Student student2 = ctx.getBean("student", Student.**class**);

student2.setName("내이름");

student2.setAge(20);

System.***out***.println("student2 수정후 student1과 2");

student1.out();

**if**(student1.equals(student2)){

System.***out***.println("student1==student2");

}**else**{

System.***out***.println("student1!=student2");

}

MainClass

student1 student2

**IOC 컨테이너**

= spring

ctx.close(); }

}

s빈: 길동 10