[ 04 ] DI(Dependency Injection) 자동 의존 설정과 빈 생명주기와 범위 (webproject로 생성)

1. **객체간 의존 자동 연결 : @Autowired 어노테이션을 이용**

프로젝트의 규모가 조금만 커져도 한 개의 어플리케이션에서 생성하는 스프링 빈 객체는 수백개 이상으로 증가하게 되는데 이 경우 스프링 빈 간의 의존관계를 xml 설정이나 자바 기반 설정을 관리하는데 시간을 뺏길 수 있다. 또는 특정 타입의 빈 객체가 한 개밖에 존재하지 않는 경우가 많아서 의존 객체가 너무 뻔할 때가 있다. 만약 일일이 의존관계를 설정할 필요없이 자동으로 프로퍼티나 생성자 파라미터 값으로 동일 타입의 빈 객체를 전달해 주는 기능이 있다면 설정 코드가 많이 줄어들 것이다.

(예제)

Student.java

@Data

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

**private** String name;

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

**private** **double** height;

**private** **double** weight;

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

**this**.name = name;

**this**.age = age;

}

}

StudentInfo.java

@Data

**public** **class** StudentInfo {

@Autowired

**private** Student student;

// @Autowired

// public void setStudent(Student student) {

// this.student = student;

// }

}

Family.java

@Data

**public** **class** Family {

**private** String papaName;

**private** String mamiName;

**private** String sisterName;

**private** String brotherName;

**public** Family(String papaName, String mamiName) {

**this**.papaName = papaName;

**this**.mamiName = mamiName;

}

}

ex1/applicationCTX1.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.lec.ch04.ex1.Student"*>

<constructor-arg value=*"홍길동"*/>

<constructor-arg value=*"11"*/>

<property name=*"height"* value=*"150"*/>

<property name=*"weight"* value=*"40"*/>

</bean>

<bean id=*"studentInfo"* class=*"com.lec.ch04.ex1.StudentInfo"*/>

<!-- <property name="student" ref="student"/>

</bean> -->

</beans>

ex1/applicationCTX2.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:c=*"http://www.springframework.org/schema/c"*

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

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

<bean id=*"family"* class=*"com.lec.ch04.ex1.Family"* c:papaName=*"홍아빠"* c:mamiName=*"김엄마"* p:brotherName=*"홍아들"* p:sisterName=*"홍딸램"*/>

</beans>

**public** **class** AutoWairedTestMain {

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

AbstractApplicationContext ctx = **new** GenericXmlApplicationContext("classpath:META-INF/ex1/CTX1.xml", "classpath:META-INF/ex1/CTX2.xml");

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

StudentInfo studentInfo = ctx.getBean("studentInfo", StudentInfo.**class**);

Student student2 = studentInfo.getStudent();

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

System.***out***.println("studentInfo의 student : " + student2);

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

System.***out***.println("두 객체는 같다");

}**else** {

System.***out***.println("두 객체는 다르다");

}

Family family = ctx.getBean("family", Family.**class**);

System.***out***.println(family);

ctx.close();

}

}

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)

Person.java

@AllArgsConstructor

@Data

**public** **class** Person {

**private** String name;

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

}

OtherPerson.java

@AllArgsConstructor

@Data

**public** **class** OtherPerson {

**private** String name;

**private** String tel;

}

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=*"person"* class=*"com.lec.ch03.ex2lifeCycle.Person"*>

<constructor-arg value=*"홍길동"* />

<constructor-arg value=*"70"* />

</bean>

<bean id=*"otherPerson"* class=*"com.lec.ch03.ex2lifeCycle.OtherPerson"*>

<constructor-arg value=*"성춘향"* />

<constructor-arg value=*"02-555-5555"* />

</bean>

</beans>

MainClass.java

<?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=*"person"* class=*"com.lec.ch03.ex2lifeCycle.Person"*>

<constructor-arg value=*"홍길동"* />

<constructor-arg value=*"70"* />

</bean>

<bean id=*"otherPerson"* class=*"com.lec.ch03.ex2lifeCycle.OtherPerson"*>

<constructor-arg value=*"성춘향"* />

<constructor-arg value=*"02-555-5555"* />

</bean>

</beans>

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)

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

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

**import** lombok.AllArgsConstructor;

**import** lombok.Data;

@AllArgsConstructor

@Data

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

**private** String name;

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

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

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

}

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

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

}

}

**import** lombok.AllArgsConstructor;

**import** lombok.Data;

**import** javax.annotation.PostConstruct;

**import** javax.annotation.PreDestroy;

@AllArgsConstructor

@Data

**public** **class** OtherPerson {

**private** String name;

**private** String tel;

@PostConstruct

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

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

}

@PreDestroy

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

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

}

}

* + Xml에 추가

<context:annotation-config/>

1. 스프링 빈 범위
   * 스프링 컨테이너가 생성되고, 스프링 빈이 생성될 때, 생성된 스프링 빈은 scope를 가지고 있다. 범위란 객체가 어디까지 영향을 미치는지 결정하는 것. 기본이 scope=*"singleton"*(cf. scope=*" prototype "*)

(예제)

<bean id=*"family"* class=*"com.lec.ch03.ex1xml.Family"* scope=*"singleton"*>

<constructor-arg value=*"홍아빠"* />

<constructor-arg value=*"홍엄마"* />

<property name=*"sisterName"* value=*"홍딸램"*/>

<property name=*"brotherName"* value=*"홍아들"*/>

</bean>

<bean id=*"familyPrototype"* class=*"com.lec.ch03.ex1xml.Family"* scope=*"prototype"*>

<constructor-arg value=*"홍아빠"* />

<constructor-arg value=*"홍엄마"* />

<property name=*"sisterName"* value=*"홍딸램"*/>

<property name=*"brotherName"* value=*"홍아들"*/>

</bean>

**public** **class** ScopeTextMain {

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

// GenericXmlApplicationContext ctx = new GenericXmlApplicationContext();

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

// ctx.refresh();

GenericXmlApplicationContext ctx = **new**

GenericXmlApplicationContext("ex3/applicationCTX.xml");

Family family1 = ctx.getBean("familyPrototype", Family.**class**);

Family family2 = ctx.getBean("familyPrototype", Family.**class**);

family1.setPapaName("김아빠");

family1.setMamiName("이엄마");

family1.setBrotherName("김아들");

family1.setSisterName("김딸램");

System.***out***.println(family1);

System.***out***.println(family2);

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

Family family3 = ctx.getBean("family", Family.**class**);

Family family4 = ctx.getBean("family", Family.**class**);

family3.setPapaName("김아빠");

family3.setMamiName("이엄마");

family3.setBrotherName("김아들");

family3.setSisterName("김딸램");

System.***out***.println(family3);

System.***out***.println(family4);

ctx.close();

}

}

family

MainClass

family3 family4

**IOC 컨테이너**

= spring