

[03] DI(Dependency Injection) 자동 설정과 빈 생명주기와 범위

1. XML 파일을 이용한 DI 설정방법 ; XML파일을 이용한 DI설정 방법은 그 동안 우리가 살펴본 방식 입니다. 이미 학습한 기본적인 사항들과 추가적인 새로운 사항들에 대해서 알아보니다.

@Autowired 어노테이션을 이용한 객체간 의존 자동 연결

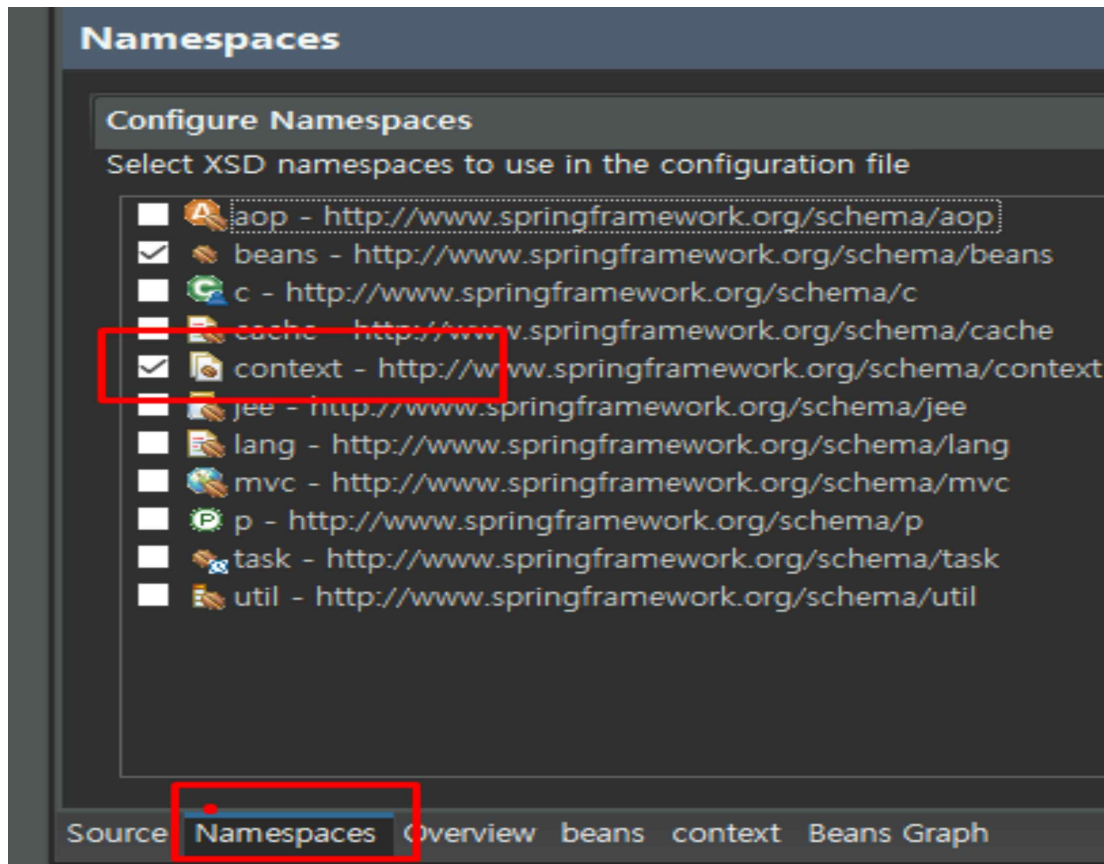
프로젝트의 규모가 조금만 커져도 한 개의 어플리케이션에서 생성하는 스프링 빈 객체는 수백개 이상으로 증가하게 되는데 이 경우 스프링 빈 간의 의존관계를 xml 설정이나 자바 기반 설정을 관리하는데 시간을 뺏길 수 있다. 또는 특정 타입의 빈 객체가 한 개밖에 존재하지 않는 경우가 많아서 의존 객체가 너무 뻥할 때가 있다. 만약 일일이 의존관계를 설정할 필요 없이 자동으로 프로퍼티나 생성자 파라미터 값으로 동일 타입의 빈 객체를 전달해 주는 기능이 있다면 설정 코드가 많이 줄어들 것이다.

Ch03_diLife

Spring MVC Project 로 웹프로젝트용

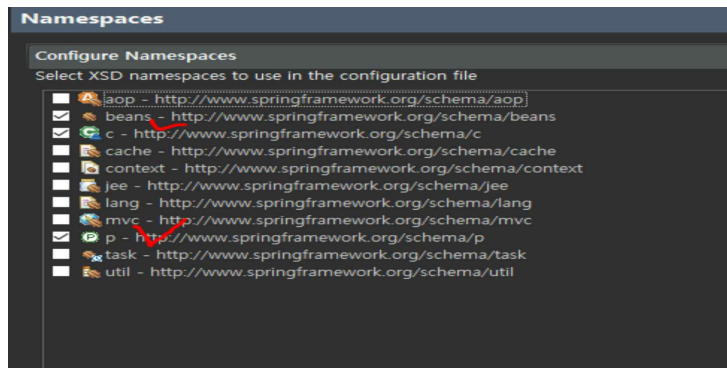
Ex1 Student.java

```
<!-- studentInfo 는 student 를 의존 받고 바로 달라붙게  
하는 @Autowired 요거해줬음-->
```



```
http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.1.xsd">
<context:annotation-config/> <!-- 어노테이션 있는지 확인해라 -->
<bean id="student" class="com.lec.ch03.ex1.Student">
  <constructor-arg value="오동준"/>
  <constructor-arg value="11"/>
  <constructor-arg>
    <list>
      <value>골프</value><value>야호</value>
    </list>
  </constructor-arg>
  <property name="height" value="180"/>
  <property name="weight" value="60"/>
</bean>
<bean id="studentInfo" class="com.lec.ch03.ex1.StudentInfo"> <!-- studentInfo 는 student 를 의존 받고 바로 달라볼게 하는 @Autowired 요거해줬음-->
  <!-- <property name="student" ref="student"></property> 이제는 이렇게 했으면 보들은 늙! -->
</bean>
```

ex1.Family.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"
       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
                           http://www.springframework.org/schema/c http://www.springframework.org/schema/c
                           http://www.springframework.org/schema/p http://www.springframework.org/schema/p">
  </beans>
```

2. 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();`
- ✓

Ex2.Person , OtherPerson

- Implements InitializingBean, DisposableBean – 인터페이스를 이용한 생명주기 관리 방법

```
@Data
@AllArgsConstructor //모든 생성자함수
public class Person implements InitializingBean, DisposableBean, EnvironmentAware{ //Initia
    private String name;
    private String tel;
    @Override
    public void setEnvironment(Environment environment) {
        System.out.println("Person 형 빈 객체 생성하자마자 실행 1 : setEnvironment()");
    }
    @Override
    public void afterPropertiesSet() throws Exception { // afterPropertiesSet : 빈을 생성하자마자 실행
        System.out.println("Person 형 빈 객체 생성하자마자 실행 2 : afterPropertiesSet()");
    }
    @Override
    public void destroy() throws Exception { // destroy :
        System.out.println("Person 형 빈 소멸 바로 전 실행 : destroy() 호출" );
    }
}
```

InitializingBean : afterPropertiesSet

DisposableBean : destroy

EnvironmentAware : setEnvironment

-OtherPerson.java

```
OtherPerson.java x Person.java applicationCTD.xml TextMain.java applicationCTD.txt
1 package com.lec.ch03.ex2;
2
3 import javax.annotation.PostConstruct;
4 import javax.annotation.PreDestroy;
5
6 import lombok.AllArgsConstructor;
7 import lombok.Data;
8 @Data
9 @AllArgsConstructor
10 public class OtherPerson {
11     private String name;
12     private int age;
13     @PostConstruct
14     public void initMethod() {
15         System.out.println("OtherPerson 형 빈 생성하자마자 자동 호출");
16     }
17     @PreDestroy
18     public void destroyMethod() {
19         System.out.println("OtherPerson 형 빈 소멸 전 자동 호출");
20     }
21 }
22 }
```

Console x
 INFO : org.springframework.beans.factory.xml.XmlBeanDefinitionParser:100-331:beans.xml:2022-7-13 9:51:11:42:42 - 9:51:11
 INFO : org.springframework.context.support.GenericXmlApplicationContext:100-331:beans.xml:2022-7-13 9:51:11:42:42 - 9:51:11
 INFO : org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor:100-331:beans.xml:2022-7-13 9:51:11:42:42 - 9:51:11
 INFO : org.springframework.beans.factory.support.DefaultListableBeanFactory:100-331:beans.xml:2022-7-13 9:51:11:42:42 - 9:51:11
 Person 형 빈 객체 생성하자마자 실행 1 : setEnvironment()
 Person 형 빈 객체 생성하자마자 실행 2 : afterPropertiesSet()
 OtherPerson 형 빈 생성하자마자 자동 호출

 Person(name=다재이통학, tel=010-9999-9999)
 OtherPerson(name=오디젤, age=30)

 INFO : org.springframework.context.support.GenericXmlApplicationContext:100-331:beans.xml:2022-7-13 9:51:11:42:42 - 9:51:11
 INFO : org.springframework.beans.factory.support.DefaultListableBeanFactory:100-331:beans.xml:2022-7-13 9:51:11:42:42 - 9:51:11
 OtherPerson 형 빈 소멸 전 자동 호출
 Person 형 빈 소멸 바로 전 실행 : destroy() 호출
 빈 소멸 !

Ex3 SingletonTestMain.java 요약

- Ex3/ 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"
       >
    <bean id="family" class="com.lec.ch03.ex1.Family" scope="singleton"> <!-- 싱글톤 -->
        <constructor-arg value="오아빠"/>
        <constructor-arg value="오엄마"/>
        <property name="sisterName" value="오딸랑"/>
        <property name="brotherName" value="오아들랑"/>
    </bean>
    <bean id="familyPrototype" class="com.lec.ch03.ex1.Family" scope="prototype"> <!-- 프로토타입 -->
        <constructor-arg value="동아빠"/>
        <constructor-arg value="동엄마"/>
        <property name="sisterName" value="동딸랑"/>
        <property name="brotherName" value="동아들랑"/>
    </bean>
</beans>
```

위에는 싱글톤 아래는 proto type 밑에서 실행 확인 해보자

```
1 package com.lec.ch03.ex3;
2
3 import org.springframework.context.support.GenericXmlApplicationContext;
4
5 import com.lec.ch03.ex1.Family;
6
7 public class SingletonTestMain {
8     public static void main(String[] args) {
9         String location = "classpath:META-INF/ex3/applicationCtx.xml";
10        GenericXmlApplicationContext ctx = new GenericXmlApplicationContext(location);
11        Family family1 = ctx.getBean("family", Family.class);
12        Family family2 = ctx.getBean("family", Family.class);
13        family1.setPapaName("박아빠");
14        family1.setMamiName("박엄마");
15        System.out.println("family1 : " + family1);
16        System.out.println("family2 : " + family2);
17        System.out.println("family1 : " + family1);
18        Family family3 = ctx.getBean("familyPrototype", Family.class);
19        Family family4 = ctx.getBean("familyPrototype", Family.class);
20        family3.setBrotherName("집나갔나아들아");
21        family3.setSisterName("집에있나 딸랑아");
22        System.out.println("family3 : " + family3);
23        System.out.println("family4 : " + family4);
24    }
25 }
```

Console Output:

```
INFO : org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loading bean definitions from classpath:META-INF/ex3/applicationCtx.xml
INFO : org.springframework.context.support.GenericXmlApplicationContext - Loading bean definitions from classpath:META-INF/ex3/applicationCtx.xml
family1 : Family(papaName=박아빠, mamiName=박엄마, sisterName=오딸랑, brotherName=오아들랑)
family2 : Family(papaName=박아빠, mamiName=박엄마, sisterName=오딸랑, brotherName=오아들랑)
family1 : Family(papaName=박아빠, mamiName=박엄마, sisterName=오딸랑, brotherName=오아들랑)
family3 : Family(papaName=동아빠, mamiName=동엄마, sisterName=집에있나 딸랑아, brotherName=집나갔나아들아)
family4 : Family(papaName=동아빠, mamiName=동엄마, sisterName=동딸랑, brotherName=동아들랑)
```

1. 싱글톤이여서 하나만 바뀌어도 전체 다 바뀐다
2. Prototype 이여서 부분으로 바꿀수 있다 .