++++++++++++++++++++++++++++++++++++

+Chapter 04 Annotation 기반 설정 및 Java Code 기반 설정+

++++++++++++++++++++++++++++++++++++

===============================================================================

: Java5부터 추가된 Annotation은 Code에 설정 정보를 입력함으로써 설정 파일을 사용하지 않거나 또는

설정 파일의 크기를 줄이고 싶을 때 유용하게 사용되고 있다. SPRING2 버전부터 Annotation을 이용하여

빈과 관련된 정보를 설정할 수 있도록 하고 있다. 본 절에서는 설정 정보와 관련된 Annotation의 사용 방법에

대해서 살펴보도록 하자.

===============================================================================

1. Annotation 기반 설정

1.1 @Required Annotation을 이용한 필수 프로퍼티 검사

: org.springframework.beans.factory.annotation 패키지에 위치한 @Required Annotation은 SPRING2 부터 제공되고

있으며 필수 프로퍼티를 명시할 때 사용된다. 필수 프로퍼티를 지정하려면 먼저 프로퍼티 설정 메서드에

@Requied Annotation을 붙여야 한다.

import org.springframework.beans.factory.annotation.Required

public class Camera{

private int number;

@Required

public void setNumber(int number){

this.number=number;

}

}

@Required Annotation을 추가했다고 해서 해당 프로퍼티를 필수 프로퍼티로 인식하지는 않는다. SPRING 설정 파일에

RequiredAnnotationBeanPostProcessor Class를 빈으로 등록해야 비로서 SPRING이 @Required Annotation을 알맞게

처리하게 된다.

<bean class="org.springframework.beans.factory.annotaction.RequiredAnnotationBeanPostProcessor"/>

<bean id="camera" class="madvirus.spring.ch04.homecontrol.Camera">

<!--number 프로퍼티에 @Required Annotation 적용

따라서, number 프로퍼티를 설정하지 않을 경우 예외 발생-->

<property name="number" value="1"/>

</bean>

RequiredAnnotationBeanPostProcessor Class는 SPRING Container에 등록된 빈 객체를 조사하여 @Requied Annotation이

적용된 프로퍼티의 값이 설정되어 있는 지의 여부를 검사한다. 만약 프로퍼티 값이 설정되어 있지 않다면 빈 생성시

예외를 발생시킨다.

RequiredAnnotationBeanPostProcessor Class를 빈으로 등록하는 대신에 <context:annotation-config> 태그를 사용해도

@Requied Annotation이 적용된다.

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

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

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-3.0.xsd

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

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<context:annotation-config />

..

</beans>

<context:annotation-config> 태그는 Annotation과 관련해서 다음의 BeanPostProcessor를 함께 등록해준다.

-RequiredAnnotationBeanPostProcessor : @Required Annotation 처리

-AutowiredAnnotationBeanPostProcessor : @Autowired Annotation 처리

-CommonAnnotationBeanPostProcessor : @Resource, @PostConstruct, @PreDestroy Annotation 처리

-ConfigurationClassPostProcessor : @Configuration Annotation 처리

따라서, 특별히 특정 기능을 사용하지 않아야 하는 경우가 아니라면 <context:annotation-config> 태그를 사용하는 것이

설정 파일을 단순하게 만들어 준다.

---------------------------------------------------------------------------------------------------------------------------------------------------------------

1.2 @Autowired Annotation을 이용한 자동 설정

: org.springframework.beans.factory.annotation 패키지에 위치한 @Autowired Annotation은 의존 관계를 자동으로 설정할 때

사용된다. @Autowired Annotation은 SPRING2.5에 추가된 기능으로 타입을 이용하여 의존하는 객체를 삽입해 준다.

@Autowired Annotation은 생성자, 필드, 메서드의 세 곳에 적용이 가능한데, 예를 들어, 프로퍼티 설정 메서드에 다음과 같이

@Autowired Annotation을 적용할 수 있다.

import org.springframework.beans.factory.annotation.Autowired;

public class MonitorViewer implements Viewer {

private DisplayStrategy displayStrategy;

@Autowired

public void setDisplayStrategy(DisplayStrategy displayStrategy) {

this.displayStrategy = displayStrategy;

}

...

}

@Autowired Annotation은 타입을 이용한 프로퍼티 자동 설정 기능을 제공한다. 따라서서, 위 Code는 displayStrategy 프로퍼티에

DisplayStrategy 타입의 빈 객체를 전달한다.

@Autowired Annotation을 이용한 자동 설정을 적용하려면 AutowiredAnnotationBeanPostProcessor Class를 SPRING 설정 파일에

빈 객체로 등록해 주어야 한다.

<bean class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor"/>

또는, 다음과 같이 <context:annotation-config> 태그를 사용해도 된다.

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

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

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-3.0.xsd

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

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<context:annotation-config />

..

</beans>

@Autowired Annotation은 프로퍼티 설정 메서드 뿐만 아니라 일반 메서드에도 적용 가능하다. 즉, 메서드 이름이

setXXX() 형식이 아니어도 @Autowired Annotation을 적용할 수 있다. 아래 Code는 prepare() 메서드에서

@Autowired Annotation을 적용했는데, 이 경우 SPRING은 HomeController 객체를 생성할 때 prepare() 메서드에

AlarmDevice 타입의 빈 객체와 Viewer 타입의 빈 객체를 전달한다.

import org.springframework.beans.factory.annotation.Autowired;

public class HomeController {

private AlarmDevice alarmDevice;

private Viewer viewer;

@Autowired

public void prepare(AlarmDevice alarmDevice, Viewer viewer) {

this.alarmDevice = alarmDevice;

this.viewer = viewer;

}

...

}

멤버 필드에 직접 @Autowired Annotation을 적용해도 된다. 아래 Code는 적용 예이다.

public class MonitorViewer implements Viewer{

@Autowired

private DisplayStrategy displayStrategy;

...

}

@Autowired Annotation을 배열에 적용하게 되면, 해당 타입의 모든 빈 객체를 배열로 전달 받는다. 예를 들어, 아래 Code와

같이 @Autowired Annotation을 적용하면 InfraredRaySensor 타입의 빈 객체가 배열로 setSensor() 메서드에 전달된다.

메서드 뿐만 아니라 배열 타입의 멤버 필드에 @Autowired Annotation을 적용해도 동일하게 동작한다.

public class HomeController {

private InfraredRaySensor[] sensors;

@Autowired

public void setSensors(InfraredRaySensor[] sensors){

this.sensors = sensors;

}

...

}

제너릭이 적용된 컬렉션 타입을 사용하는 경우 java.util.List 타입이나 java.util.Set 타입을 이용하여 특정 타입의 빈 객체

목록을 전달 받을 수 있다.

public class HomeController {

...

private List<InfraredRaySensor> sensors;

@Autowired

public void setSensors(List<InfraredRaySensor> sensors){

this.sensors = sensors;

}

...

}

(1) @Autowired Annotation 적용 프로퍼티의 필수 여부 지정

: @Autowired Annotation은 타입을 이용해서 자동적으로 프로퍼티 값을 설정하기 때문에, 해당 타입의 빈 객체가

존재하지 않거나 또는 빈 객체가 두 개 이상 존재할 경우 SPRING은 @Autowired Annotation이 적용된 빈 객체를

생성할 때 예외를 발생시키게 된다.

@Autowired Annotation을 적용한 프로퍼티를 반드시 설정할 필요가 없는 경우도 있는데, 이런 경우에는

@Autowired Annotation에 required 속성의 값을 false로 지정해 주면 된다.

public class HomeController {

...

@Autowired(required=false)

public void setSensors(List<InfraredRaySensor> sensors){

this.sensors = sensors;

}

...

}

required 속성의 값을 false로 지정할 경우, 해당 타입의 빈 객체가 존재하지 않더라도 SPRING은 예외를 발생하지 않는다.

참고로 @Autowired Annotation의 required 속성의 기본값은 true이다.

(2) @Qualifier Annotation을 이용한 자동 설정 제한

: @Autowired Annotation은 타입을 기반으로 하기 때문에, @Autowired Annotation을 이용하여 자동 설정할 타입이

두 개 이상 존재할 경우 예외를 발생시킨다. 이런 경우 @Qualifier Annotation을 이용하여 동일한 타입의 빈 객체들 중에서

특정 빈을 사용하도록 설정할 수 있다.

@Qualifier Annotation은 orag.springframework.beans.factory.annotation 패키지에 정의되어 있으며, @Autowired Annotation과

함께 사용된다. @Qualifier Annotation은 자동 연결될 빈 객체의 수식어를 값으로 갖는다.

public class HomeController {

@Autowired

@Qualifier('main')

private Recorder recorder;

...

}

위 Code는 recorder 멤버 필드에 Recorder 타입의 빈 객체를 자동 연결하는데, 수식어가 'main'인 빈 객체를 연결한다는 것을 의미한다.

빈 객체의 수식어는 설정 파일에서 <qualifier> 태그를 이용하여 설정할 수 있다. 아래 Code는 <qualifier> 태그를 이용하여 빈 객체에

수식어를 지정하는 예를 보여 주고 있다.

<bean id="recorder" class="madvirus.spring.ch04.homecontrol.Recorder">

<qualifier value="main"/>

</bean>

<qualifier> 태그의 value 속성의 값을 @Qualifier Annotation 값으로 사용하게 된다.

생성자나 메서드가 2개 이상의 파라미터를 갖는 경우 @Qualifier Annotation을 각 파라미터에 적용함으로써 각 파라미터에 전달되는

빈 객체를 한정할 수도 있다.

@Autowired

public void prepare(AlarmDevice alarmDevice, @Qualifier("center") Viewer viewer) {

this.alarmDevice = alarmDevice;

this.viewer = viewer;

}

---------------------------------------------------------------------------------------------------------------------------------------------------------------

1.3 @Resource Annotation을 이용한 프로퍼티 설정

: javax.annotation 패키지에 위치한 @Resource Annotation은 Java6 버전 및 JEE5 버전에 추가된 Annotation으로서

Application에서 필요로 하는 Resource을 자동 연결할 때 사용된다. SPRING2.5부터 @Resource Annotation을 지원하는데,

SPRING에서는 의존하는 빈 객체를 전달할 때 사용된다. SPRING에서 @Resource Annotation을 사용하려면 다음과 같이

name 속성에 자동으로 연결할 빈 객체의 이름을 입력하면 된다.

import javax.annotation.Resource;

public class HomeController{

@Resource(name="camera1")

private Camera camera1;

private Camera camera4;

@Resource(name="camera4")

public void setCamera4(Camera camera4){

this.camera4=camera4;

}

}

@Resource Annotation이 적용되려면 CommonAnnotationBeanPostProcessor Class를 빈 객체로 등록해야 한다.

<bean class="org.springframework.beans.factory.annotation.CommonAnnotationBeanPostProcessor"/>

또는 <context:annotation-config /> 태그를 이용해도 된다.

만약 @Resource Annotation에서 지정한 빈 객체가 존재하지 않으면, SPRING은 Container를 초기화하는 과정에서

NoSuchBeanDefinitionException 예외를 발생시킨다.

---------------------------------------------------------------------------------------------------------------------------------------------------------------

1.4 @PostConstruct Annotation 및 @PreDestroy Annotation과 라이프 사이클

: @PostConstruct Annotation 및 @PreDestroy Annotation은 javax.annotation 패키지에 위치하며, @Resource Annotation과

마찬가지로 Java6 및 JEE5에 추가된 Annotation으로서 라이프 사이클의 초기화 및 제거 과정을 제공한다. 이 두 Annotation은

SPRING2.5부터 지원한다.

@PostConstruct Annotation은 의존하는 객체를 설정한 이후에 초기화 작업을 수행할 메서드에 적용되며, @PreDestroy Annotation은

Container에서 제거하기 전에 호출 될 메서드에 적용된다. 즉, SPRING 설정 파일에서 init-method 속성과 destroy-method 속성을

이용하여 명시한 메서드와 동일한 시점에 실행된다.

import javax.annotation.PostConstruct;

import javax.annotation.PreDestroy;

public class HomeController{

...

@PostConstruct

public void init(){

//초기화 처리

}

@PreDestroy

public void destroy(){

//Resource 반환 등 종료 처리

}

}

@PostConstruct Annotation 및 @PreDestroy Annotation은 @Resource Annotation과 마찬가지로 CommonAnnotationBeanPostProcessor

Class를 빈 객체로 등록해 주거나 <context:annotation-config> 태그를 사용하면 적용된다.

===============================================================================

2. 빈 객체 스캔

: SPRING2.5 버전부터 Class패스에 위치한 Class를 검색하여 특정한 Annotation이 붙은 Class를 자동으로 빈으로 등록하는 기능을 제공하고 있다.

이 기능을 사용하면 XML 설정 파일에 다양한 빈 정보를 추가하지 않고 특정한 Class를 빈으로 등록할 수 있게 된다.

2.1 빈 객체 스캔

: SPRING 2.0부터 @Repository Annotation을 제공하고 있었고, SPRING 2.5부터 @Component Annotation, @Service Annotation 그리고

@Controller Annotation이 새롭게 추가되었다. 이들 Annotation은 Class 선언 부분에 붙게 된다. 아래는 @Component Annotation의 적용 예이다.

import org.springframework.stereotype.Comonent;

...

@Component

public class HomeController{

@Resource(name="camera1")

private Camera camera1;

...

@Autowired

public void prepare(AlarmDevice alarmDevice, Viewer viewer){

this.alarmDevice = alarmDevice;

this.viewer = viewer;

}

...

}

@Component Annotation을 Class에 적용했다면, <context:component-scan> 태그를 이용해서 SPRING이 Class를 검색할

패키지를 지정하면 된다. <context:component-scan> 태그를 추가하면 SPRING은 지정한 패키지에서 @Component Annotation이

(또는 하위 Annotation) 적용된 Class를 검색하여 빈으로 등록하게 된다.

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

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

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-3.0.xsd

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

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<context:component-scan base-package="madvirus.spring.chap04.homecontrol" />

..

</beans>

---------------------------------------------------------------------------------------------------------------------------------------------------------------

2.2 자동 검색된 빈의 이름과 범위

: SPRING은 기본적으로 검색된 Class를 빈으로 등록할 때, Class의 (첫 글자를 소문자로 바꾼) 이름을 빈의 이름으로 사용한다.

예를 들어, 아래 Code의 경우 자동으로 등록되는 빈의 이름은 "homeController"가 된다.

@Component

public class HomeController{

...

}

따라서, ApplicationContext를 사용하는 Code에서는 다음과 같이 "homeController"를 이용하여 SPRING Container로부터 빈 객체를

가져오면 된다.

ApplicationContext context = ...;

HomeController controller = (SystemMonitor)context.getBean("homeController");

만약 특정한 이름을 명시해 주고 싶다면 다음과 같이 Annotation의 속성에 빈의 이름을 입력하면 된다.

@Component("homeController")

public class HomeController{

...

}

SPRING은 기본적으로 빈의 범위를 "singleton"으로 설정한다. 스캔을 통해서 검색된 빈 역시 "singleton" 범위를 갖게 된다.

"singleton"이 아닌 다른 범위를 지정하고 싶다면 다음과 같이 @Scope Annotation을 이용하여 범위를 지정하면 된다.

import org.springframework.context.annotation.Scope;

import org.springframework.stereotype.Component;

@Component

@Scope("prototype")

public class Worker{

...

}

2장에서 살펴봤었던 <aop:scoped-proxy> 태그와 동일하게 프록시 객체를 생성하고 싶다면 다음과 같이 proxyMode 속성의 값으로

ScopedProxyMode 열거형 값을 할당해 주면 된다.

import org.springframework.context.annotation.Scope;

import org.springframework.context.annotation.ScopedProxyMode;

import org.springframework.stereotype.Component;

@Component

@Scope(value="prototype", proxyMode=ScopedProxyMode.TARGET\_CLASS)

public class Worker{

...

}

ScopedProxyMode에 정의된 열거값은 아래와 같다.

-NO : 프록시를 생성하지 않음.

-INTERFACES : Interface에 대한 프록시를 생성한다.(JDK 다이나믹 프록시를 이용해서 프록시 생성)

-TARGET\_CLASS : Class에 대해 프록시를 생성한다.(CGLIB 이용)

-DEFAULT : 기본 값. 별도의 설정을 하지 않았다면, NO와 동일하다.

<context:component-scan>태그에서 설정을 변경할 수 있다.

<context:component-scan> 태그의 scoped-proxy 속성의 값을 사용하면 기본적으로 프록시 객체를 생성할 지의 여부를

지정할 수 있다.

<context:component-scan base-package="madvirus.spring.ch04.homecontrol" scoped-proxy="no"/>

scoped-proxy 속성에 올 수 있는 값은 "no","interfaces", 그리고 "targetClass"이다.

---------------------------------------------------------------------------------------------------------------------------------------------------------------

2.3 스캔 대상 Class 범위 지정하기

: <context:include-filter> 태그와 <context:exclude-filter> 태그를 사용하면 자동 스캔 대상에 포함시킬 Class와 포함시키지 않을

Class를 구체적으로 명시할 수 있다. 아래 Code는 필터 태그의 사용 예이다.

<context:component-scan base-package="madvirus.spring.ch02.scan">

<context:include-filter type="regex" expression=".\*HibernateRepository"/>

<context:exclude-filter type="aspectj" expression="..\*IBatisRepository"/>

</context:conponent>

<context:include-filter> 태그와 <context:exclude-filter> 태그는 각각 type 속성과 expression 속성을 갖는다. type 속성에 따라

expression 속성에 올 수 있는 값이 달라지는데, type 속성에 입력 가능한 값은 아래와 같다.

-annotation : Class에 지정한 Annotation이 적용되었는지의 여부. expression 속성에는 "org.example.SomeAnnotation"와 같이 Annotation 이름을 입력한다.

-assignable : Class가 지정한 타입으로 할당 가능한지의 여부. expression 속성에는 "org.example.SomeAnnotation"와 같이 Annotation 이름을 입력한다.

-regex : Class 이름이 정규 표현식에 매칭되는 지의 여부. expression 속성에는 "org\.example\.Default.\*"와 같이 정규 표현식을 입력한다.

-aspectj : Class 이름이 AspectJ의 표현식에 매칭되는지의 여부. expression 속성에는 "org.example..\*Service+"와 같이 AspectJ의 표현식을 입력한다.

===============================================================================

3. Java Code 기반 설정

: Spring JavaConfig 프로젝트는 XML이 아닌 Java Code를 이용해서 Container를 설정할 수 있는 기능을 제공하는 프로젝트로서,

이를 사용하면 XML이 아닌 Java Code를 이용해서 생성할 빈객체와 각 빈 간의 연관등을 처리하게 된다. Java Code를 이용하기 때문에

이클립스와 같은 IDE가 제공하는 Code 완성 기능을 이용해서(XML에 비해 상대적으로) 오타 없이 빈 객체를 설정할 수 있게 된다.

SPRING 3버전은 Spring JavaConfig 프로젝트가 제공하는 기능을 핵심 기능으로 추가하였으며, 따라서 SPRING 3버전부터는

별도의 확장 Module 없이 Java Code를 이용해서 Container를 설정할 수 있게 되었다. 본 절에서는 SPRING 3에서 제공하는 Java Code

기반의 설정 방법을 살펴보도록 하겠다.

3.1 @Configuration Annotation과 @Bean Annotation을 이용한 Code 기반 설정

: org.springframework.context.annotation 패키지의 @Configuration Annotation과 @Bean Annotation을 이용해서

SPRING Container에 새로운 빈 객체를 제공할 수 있다. 다음은 Java Code를 이용해서 새로운 빈 객체를 SPRING Container에

제공하는 Class의 예를 보여주고 있다.

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class SpringConfig {

@Bean

public AlarmDevice alarmDevice() {

return new SmsAlarmDevice();

}

}

@Bean Annotation은 새로운 빈 객체를 제공할 때 사용되며, @Bean이 적용된 메서드의 이름을 빈의 식별값으로 사용한다.

따라서, 위 Code는 다음의 SPRING XML 설정과 동일한 빈을 정의한다.

<bean id="alarmDevice" class="madvius.spring.ch04.homecontrol.SmsAlarmDevice"/>

메서드의 이름이 아닌 다른 이름을 빈 객체의 이름으로 사용하고 싶다면 @Bean Annotation의 name 속성을 사용하면 된다.

@Bean(name="smsAlarmDevice")

public AlarmDevice alarmDevice() {

return new SmsAlarmDevice();

}

(1) @Bean 객체 간의 의존 설정

: 의존 설정은 매우 간단하다. 다음과 같이 의존할 빈 객체에 대한 메서드를 호출하는 것으로 의존 관계를 설정할 수 있다.

@Configuration

public class SpringConfig {

@Bean

public Camera camera1() {

return new Camera();

}

@Bean

public InfraredRaySensor windowSensor() {

return new InfraredRaySensor("창 센서");

}

@Bean

public Viewer viewer() {

MonitorViewer viewer = new MonitorViewer();

viewer.setDisplayStrategy(displayStrategy());

return viewer;

}

@Bean

public DisplayStrategy displayStrategy() {

return new DefaultDisplayStrategy();

}

...

@Bean(initMethod = "init")

public HomeController homeController() {

HomeController homeController = new HomeController();

List<InfraredRaySensor> sensors = new ArrayList<InfraredRaySensor>();

sensors.add(windowSensor());

sensors.add(doorSensor());

homeController.setSensors(sensors);

...

homeController.setCamera1(camera1());

homeController.setDisplayStrategy(displayStrategy());

return homeController;

}

}

위 Code에서 각 빈 객체들은 의존할 빈에 해당하는 메서드를 호출함으로써 의존 관계를 설정하고 있다. 여기서

눈여겨 볼 부분은 displayStrategy() 메서드가 viewer() 메서드와 homeController() 메서드에서 각각 한번씩 호출된다는

점이다. 즉 displayStrategy() 메서드가 두 번 호출된다. displayStrategy() 메서드가 호출될 때마다 새로운 빈 객체가 생성될거라고

생각할 수 있는데, 실제로 displayStrategy() 메서드는 호출 횟수에 상관없이 매번 동일한 객체를 리턴한다.

SPRING은 CGLIB를 이용해서 @Configuration이 적용된 Class의 프록시 객체를 생성한다. 이 프록시 객체는 @Bean Annotation이

적용된 메서드가 호출될 때 생성할 빈 객체의 범위에 따라서 알맞은 객체를 제공한다. 예를 들어, 설정한 빈의 범위가

singleton이면 메서드가 리턴하는 객체는 매번 동일한 객체가 되고, 빈의 범위가 prototype이면 메서드가 매번

새로운 빈 객체를 리턴하게 된다.

(2) @Bean Annotation의 autowire 속성을 이용한 연관 자동 설정

: XML 설정 파일의 경우 아래와 같이 <bean> 태그의 autowire 속성을 이용해서 자동 연관 처리 여부를 설정할 수 있다.

<bean name="writeArticleService" class="madvirus.spring.chap02.WriteArticleServiceImpl" autowire="byName"/>

@Bean Annotation 사용하는 경우에도 autowire 속성을 이용해서 자동 연관 여부를 설정할 수 있다. 아래 Code는 설정 적용 예이다.

import org.springframework.beans.factory.Autowire;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class SpringConfig{

@Bean(autowire=Autowire.BY\_NAME)

public HomeController homeController(){

HomeController homeController = new HomeController();

...

return homeController;

}

...

}

autowire 속성에는 Autowire 열거 타입의 값이 올 수 있으며, Autowire 열거 타입이 제공하는 값은 아래와 같다.

-Autowire.BY\_NAME : 이름을 이용해서 자동 연관을 처리한다.

-Autowire.BY\_TYPE : 타입을 이용해서 자동 연관을 처리한다.

-Autowire.NO : 자동 연관을 처리하지 않는다.

---------------------------------------------------------------------------------------------------------------------------------------------------------------

3.2 @Configuration Annotation 기반 설정 정보 사용

: Class에 @Configuration Annotation을 적용한다고 해서 SPRING Container가 해당 Class로부터 빈 정보를

구할 수 있는 것은 아니다. @Configuration Annotation이 적용된 Class를 이용해서 SPRING 빈을 생성하는

방법은 AnnotationConfigApplicationContext를 이용하는 방법과 XML 설정에서 ConfigurationClassPostProcessor를

이용하는 방법이 존재하는데, 이 두 방법에 대해서 살펴보도록 하자.

(1) AnnotationConfigApplicationContext를 이용한 @Configuration Class 사용

: @Configuration Annotation이 적용된 Class를 설정 정보로 이용하는 첫 번째 방법은 AnnotationConigApplicationContext

를 이용하는 것이다. 다음과 같이 생성자에 @Configuration Annotation Class를 전달해 주기만 하면 된다.

import org.springframework.context.ApplicationContext;

import org,springframework.context.annotation.AnnotationContextApplicationContext;

import org.springframework.context.support.AbstractApplicationContext;

import.org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainConfig{

public static void main(String[] args){

ApplicationContext context = new AnnotationConfigApplicationContext(SpringConfig.class);

HomeController homeControl = context.getBean("homeController",HomeController.class);

...

}

}

한 개 이상의 @Configuration Annotation 적용 Class로부터 ApplicationContext를 생성하고 싶다면 다음과 같이

@Configuration Annotation Class의 목록을 지정하면 된다.

ApplicationContext context = new AnnotationConfigApplicationContext(ArticleServiceConfig.class, ArticleRepositoryConfig.class);

AnnotationConfigApplicationContext Class는 <context:annotation-config> 태그를 사용했을 때와 마찬가지로 각 빈Class에서

@Autowired Annotation과 @Resource Annotation 등 앞서 'Annotation 기반 설정'절에서 살펴봤던 Annotation들이

기본으로 적용된다.

(2) XML 설정 파일에서 @Configuration Annotation Class 사용하기

: XML 설정 파일에서 @Configuration Annotation Class에 정의된 빈 정보를 이용할 수도 있다.

XML 설정 파일에 @Configuration Annotation Class를 사용하려면 ConfigurationClassPostProcessor Class와

@Configuration Annotation이 적용된 Class를 SPRING 설정 파일에 빈 객체로 등록해 주면 된다.

<bean class="org.springframework.context.annotation.ConfigurationClassPostProcessor"/>

<bean class="madvirus.spring.ch04.config.SpringConfig"/>

ConfigurationClassPostProcessor Class는 @Configuration Annotation이 적용된 빈 객체에서 @Bean Annotation이

적용된 메서드로부터 빈 객체를 가져와 SPRING Container에 등록한다.

@Configuration Annotation이 적용된 Class로부터 생성한 빈 객체들도 다른 빈과 동일하게 @Required Annotation,

@Resource Annotation을 이용해서 의존 관계를 설정할 수 있다. 단, 이들 Annotation 기반 의존 설정 기능을 사용하려면

ConfigurationClassPostProcessor뿐만 아니라 RequiredAnnotationBeanPostProcessor 등을 비롯한 관련 BeanPostProcessor들을

추가로 등록해 주어야 한다. 또는 다음과 같이 <context:annotation-config/> 태그를 사용해도 된다.

<context:annotation-config />

<bean class="madvirus.spring.chap04.config.SpringConfig"/>

@Configuration Annotation이 적용된 Class는 @Component Annotation이 적용된 Class와 마찬가지로 컴포넌트 스캔 대상이다.

따라서, 아래와 같이 컴포넌트 스캔 기능을 이용해서 @Configuration Annotation이 적용된 Class를 자동으로 빈으로 등록할 수 있다.

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

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

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-3.0.xsd

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

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<context:component-scan base-package="madvirus.spring.chap04.config" />

..

</beans>

(3) @ImportResource를 통해 @Configuration 설정 Class에서 XML 사용하기

: 앞서 XML 설정 파일에서 @Configuration Class를 사용하는 방법을 살펴봤는데, 반대로 @Configuration Class에서

XML 설정 정보를 사용할 수도 있다. @Configuration Class에서 XML 설정 정보를 함께 사용하고 싶다면

@ImportResource Annotation을 사용하면 된다.

import org.springframework.context.annotation.Configuration;

import org.springframework.context.annotation.ImportResource;

@Configuration

@ImportResource("classpath:/article-repository.xml")

public class ArticleConfigWithImportResource{

@Autowired

private ArticleRepository articleRepository;

@Bean

public ArticleService articleService{

return new ArticleServiceImpl(articleRepository);

}

}

두 개 이상의 XML 설정 파일을 사용하고 싶다면 배열 형태로 설정 파일 목록을 지정해주면 된다.

@Configuration

@ImportResource("classpath:/article-repository.xml", "classpath:/article-datasource.xml")

public class ArticleConfigWithImportResource{

...

}

---------------------------------------------------------------------------------------------------------------------------------------------------------------

3.3 서로 다른 @Configuration Annotation Class 간의 의존 설정

: @Configuration Annotation Class를 작성하다 보면, 다른 @Configuration Annotation Class에서 설정한 빈 객체를

의존하는 경우가 있다. 이런 경우는 주로 layer 별로 설정 정보를 구분할 때 발생한다. 예를 들어, 아래 Code와 같이

서비스 layer와 리포지토리 layer에 대한 설정을 별도 Class로 제공한다고 해 보자.

@Configuration

public class ArticleServiceConfig{

@Bean

public ArticleService articleService(){

//다른 설정에서 정의한 빈에 의존

ArticleRepository articleRepository = ...;

return new ArticleServiceImpl(articleRepository);

}

}

@Configuration

public class ArticleRepositoryConfig{

@Bean

public ArticleRepository articleRepository{

return new ArticleRepositoryImpl();

}

}

위 Code에서 ArticleServiceConfig에 설정된 articleService 빈은 ArticleRepositoryConfig에 설정된 articleRepository 빈에

의존하고 있다. 이렇게 서로 다른 설정 Class에 존재하는 빈 객체 간의 의존을 처리할 때에는 아래와 같이 @Autowired

Annotation이나 @Resource Annotation을 이용해서 의존에 필요한 빈 객체를 전달받을 수 있다.

@Configuration

public class ArticleServiceConfig{

@Autowired

private ArticleRepository articleRepository;

@Bean

public ArticleService articleService(){

return new ArticleServiceImpl(articleRepository);

}

}

@Configuration

public class ArticleRepositoryConfig{

@Bean

public ArticleRepository articleRepository{

return new ArticleRepositoryImpl();

}

}

\*\*\*@Configuration Class에서 @Autowired Annotation을 이용한 의존 처리

위 Code에서 ArticleServiceConfig Class에서 articleRepository 필드에 @Autowired Annotation이 적용되었기 때문에

articleRepository 필드에는 ArticleRepositoryConfig Class에 정의된 articleRepository() 메서드가 생성한 빈 객체가

할당된다. 따라서 ArticleServiceConfig.articleService() 메서드에서 ArticleServiceImpl 객체를 생성할 때 사용되는

articleRepository 객체는 ArticleRepositoryConfig Class에서 생성한 ArticleRepositoryImpl 객체가 된다.

의존 객체를 참조하는 또 다른 방법은 @Configuration Class를 @Autowired 로 전달받는 것이다. 아래 Code는

@Configuration Class를 전달받아 의존을 처리하는 경우의 예를 보여주고 있다.

@Configuration

public class ArticleServiceConfig{

@Autowired

private ArticleRepositoryConfig repositoryConfig;

@Bean

public ArticleService articleService(){

return new ArticleServiceImpl(repositoryConfig.articleRepository());

}

}

(1) @Import를 이용한 @Configuration Annotation 적용 Class의 조합

: @Import Annotation을 이용하면 하나의 @Configuration Class에서 다수의 @Configuration Class를 묶을 수 있다.

아래와 같이 @Import Annotation에 설정 정보를 가져올 @Configuration Class 목록을 지정해주면 된다.

import org.springframework.context.annotation.Configuration;

import org.springframework.context.annotation.Import;

@Configuration

@Import({ArticleServiceConfig.class, ArticleRepositoryConfig.class})

public class ArticleConfigWithImport{

...

}

@Import Annotation을 이용할 경우의 장점은 개발자가 모든 @Configuration Class 목록을 기억할 필요 없이

@Import Annotation이 적용된 Class만 기억하면 손쉽게 설정 정보 추적이 가능하다는 점이다.

---------------------------------------------------------------------------------------------------------------------------------------------------------------

3.4 라이프 사이클 처리

: @Bean Annotation을 통해서 생성되는 빈 객체도 XML 설정이나 컴포넌트 스캔을 통해서 생성되는 빈 객체와 마찬가지로

ApplicationContextAware Interface, InitializingBean Interface 등의 SPRING Container가 제공하는 라이프 사이클을 따른다.

또한, @Bean Annotation의 initMethod 속성과 destroyMethod 속성을 이용해서 커스텀 초기화/종료 메서드를 지정할 수 있다.

아래 Code는 initMethod 속성의 사용 예를 보여주고 있다.

@Bean(initMethod-"init")

public HomeController homeController(){

HomeController homeController = new HomeController();

...

return homeController;

}

---------------------------------------------------------------------------------------------------------------------------------------------------------------

3.5 @Scope Annotation을 이용한 범위 설정

: @Scope Annotation을 이용하면 @Bean Annotation을 통해서 생성할 빈 객체의 범위를 지정할 수 있다.

아래 Code는 적용 예를 보여주고 있다.

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.annotation.Scope;

@Configuration

public class SpringConfig{

@Bean

@Scope(value="prototype")

public Worker worker(){

return new Worker();

}

...

}

XML설정에서 <aop:scoped-proxy> 태그를 이용해서 빈 객체에 대한 프록시 객체를 생성해서 범위에 알맞게 연관을 맺을 수

있는데, @Configuration Annotation을 사용하는 경우 @Scope Annotation의 proxyMode 속성을 이용해서 프록시 생성 여부를

지정해 주면 된다.

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.annotation.Scope;

import org.springframework.context.annotation.ScopedProxyMode;

@Configuration

public class SpringConfig{

@Bean

@Scope(value="prototype", proxyMode=ScopedProxyMode.TARGET\_CLASS)

public Worker worker(){

return new Worker();

}

...

}

proxyMode 속성에는 ScopedProxyMode 열거 타입을 값으로 지정할 수 있으며, 설정할 수 있는 값은 아래와 같다.

-ScopedProxyMode.NO : 프록시를 적용하지 않는다.

-ScopedProxyMode.INTERFACES : Interface에 대한 프록시를 생성한다.

-ScopedProxyMode.TARGET\_CLASS : Class에 대한 프록시를 생성한다.

-ScopedProxyMode.DEFAULT : 기본 설정을 사용한다.

---------------------------------------------------------------------------------------------------------------------------------------------------------------

3.6 @Qualifier Annotation을 이용한 한정자 지정.

: 앞서 @Autowied Annotation이 적용된 프로퍼티가 두 개 이상의 빈 객체와 매칭되는 경우 @Qualifier Annotation을 이용해서

연관될 수 있는 빈을 한정시키는 내용을 살펴봤었다. 이때, XML 설정에서는 <qualifier> 태그를 이용해서 연관 대상이 되는

빈에 대한 한정자를 설정하였다.

<bean id="recorder" class="madvirus.spring.ch04.homecontrol.Recorder">

<qualifier value="main"/>

</bean>

Java Code 기반 설정을 사용할 경우에는 @Qualifier Annotation을 이용해서 한정자를 설정한다.

import org.springframework.context.annotation.Qualifier;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class SpringConfigAutowired{

@Bean

@Qualifier("main")

public Recorder recorder(){

return new Recorder();

}

...

}