Spring Framework

9. DI 애플리케이션 작성(4)

Tacademy

CONTENTS

1. Bean 등록 메타정보 구성 전략
2. Bean 등록 및 의존관계 설정 Annotation
3. 프로퍼티(Property) 파일을 이용한 값 설정 방법

학습 목표

- Bean 등록 메타정보 구성 전략에 대하여 이해할 수 있습니다.
- Bean 등록 및 의존관계 설정 Annotation에 대하여 이해할 수 있습니다.
- 프로퍼티(Property) 파일을 이용한 설정 방법에 대하여 이해할 수 있습니다.

1. Bean 등록 메타정보 구성 전략

전략(1) XML 단독 사용

- 모든 Bean을 명시적으로 XML에 등록하는 방법이다.
- 생성되는 모든 Bean을 XML에서 확인할 수 있다는 장점이 있으나 Bean의 개수가 많아지면 XML 파일을 관리하기 번거로울 수 있다.
- 여러 개발자가 같은 설정파일을 공유해서 개발하다 보면 설정파일을 동시에 수정하다가 충돌이 일어나는 경우도 적지 않다.
- DI에 필요한 적절한 setter 메서드 또는 constructor가 코드 내에 반드시 존재해야 한다.
- 개발 중에는 어노테이션 설정방법을 사용했지만, 운영 중에는 관리의 편의성을 위해 XML설정으로 변경하는 전략을 쓸 수도 있다.

전략(2) XML과 빈 스캐닝 (Bean Scanning)의 혼용

- Bean으로 사용될 클래스에 특별한 어노테이션(Annotation)을 부여해주면 이런 클래스를 자동으로 찾아서 Bean으로 등록한다.
- 특정 어노테이션이 붙은 클래스를 자동으로 찾아서 Bean으로 등록해주는 방식을 **빈 스캐닝(Bean Scanning)을 통한 자동인식 Bean 등록기능**이라고 한다.
- 어노테이션을 부여하고 자동 스캔으로 빈을 등록하면 XML 문서 생성과 관리에 따른 수고를 덜어주고 개발 속도를 향상시킬 수 있다.
- 애플리케이션에 등록될 Bean이 어떤 것들이 있고, Bean들 간의 의존관계가 어떻게 되는지를 한눈에 파악할 수 없다는 단점이 있다.

2. Bean 등록 및 의존관계 설정 Annotation

Bean 등록 Annotation

@Component	컴포넌트를 나타내는 일반적인 스테레오 타입으로 <bean> 태그와 동일한 역할을 함
@Repository	퍼시스턴스(persistence) 레이어, 영속성을 가지는 속성(파일, 데이터베이스)을 가진 클래스
@Service	서비스 레이어, 비즈니스 로직을 가진 클래스
@Controller	프리젠테이션 레이어, 웹 어플리케이션에서 웹 요청과 응답을 처리하는 클래스

- @Repository, @Service, @Controller는 더 특정한 유즈케이스에 대한 @Component의 구체화된 형태이다.

Bean 의존관계 주입 Annotation (1)

@Autowired, @Resource 어노테이션은 의존하는 객체를 자동으로 주입해 주는 어노테이션이다.

@Autowired

- 정밀한 의존관계 주입 (Dependency Injection)이 필요한 경우에 유용하다.
- @Autowired는 프로퍼티, setter 메서드, 생성자, 일반메서드에 적용 가능하다.
- 의존하는 객체를 주입할 때 주로 Type을 이용하게 된다.
- @Autowired는 <property>, <constructor-arg> 태그와 동일한 역할을 한다.

Bean 의존관계 주입 Annotation (2)

@Autowired, @Resource 어노테이션은 **의존하는 객체를 자동으로 주입해 주는 어노테이션**이다.

@Autowired는 타입으로, @Resource는 이름으로 연결한다는 점이 다르다.

@Resource

- 어플리케이션에서 필요로 하는 자원을 자동 연결할 때 사용된다.
- @Resource는 프로퍼티, setter 메서드에 적용 가능하다.
- 의존하는 객체를 주입할 때 주로 **Name**을 이용하게 된다.

Bean 의존관계 주입 Annotation (3)

@Value

- 단순한 값을 주입할 때 사용되는 어노테이션이다.
- @Value("Spring")은 <property .. value="Spring" /> 와 동일한 역할을 한다.

@Qualifier

- @Qualifier는 @Autowired 어노테이션과 같이 사용되어 진다.
- @Autowired는 타입으로 찾아서 주입하므로, 동일한 타입의 Bean객체가 여러 개 존재할 때 특정 Bean을 찾기 위해서는 @Qualifier를 같이 사용해야 한다.

Component Scan을 지원하는 태그

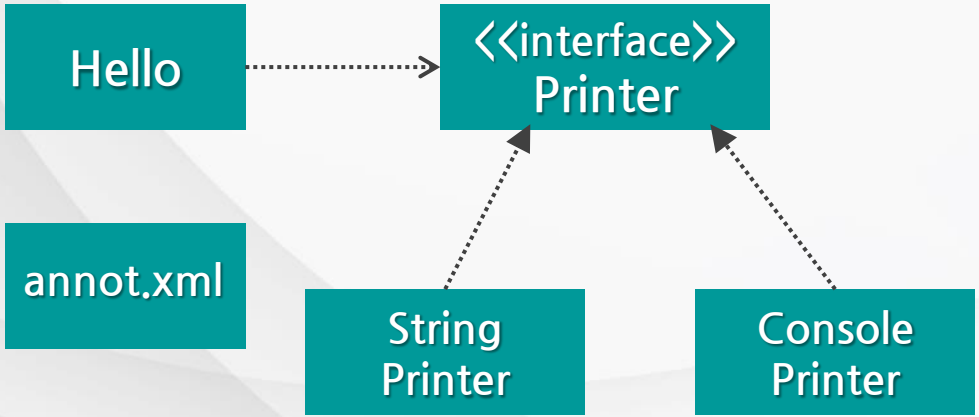
❖ <context:component-scan> 태그

- @Component를 통해 자동으로 Bean을 등록하고, @Autowired로 의존관계를 주입받는 어노테이션을 클래스에서 선언하여 사용했을 경우에는 해당 클래스가 위치한 특정 패키지를 Scan하기 위한 설정을 XML에 해주어야 한다.

```
<context:component-scan base-package="myspring.di.annot" />
```

- <context:include-filter>태그와 <context:exclude-filter>태그를 같이 사용하면 자동 스캔 대상에 포함시킬 클래스와 포함시키지 않을 클래스를 구체적으로 명시할 수 있다.

어노테이션을 사용한 POJO 클래스 작성 (1)



어노테이션을 사용한 POJO 클래스 작성 (2)

❖ StringPrinter.java

StringPrinter.java

```
1  package myspring.di.annot;
2
3  import org.springframework.stereotype.Component;
4
5  @Component("stringPrinter")
6  public class StringPrinter implements Printer {
7      private StringBuffer buffer = new StringBuffer();
8
9      public void print(String message) {
10         this.buffer.append(message);
11     }
12
13     public String toString() {
14         return this.buffer.toString();
15     }
16 }
17
```

어노테이션을 사용한 POJO 클래스 작성 (3)

❖ ConsolePrinter.java

ConsolePrinter.java

```
1  package myspring.di.annot;
2
3  import org.springframework.stereotype.Component;
4
5  @Component("consolePrinter")
6  public class ConsolePrinter implements Printer {
7      public void print(String message) {
8          System.out.println(message);
9      }
10 }
```

어노테이션을 사용한 POJO 클래스 작성 (4)

❖ Hello.java

Hello.java

```
1  package myspring.di.annot;
2
3  import org.springframework.beans.factory.annotation.Autowired;
7
8  @Component
9  public class Hello {
10     @Value("Spring")
11     String name;
12
13     @Autowired
14     @Qualifier("stringPrinter")
15     Printer printer;
16
17     public String sayHello() {
18         return "Hello " + name;
19     }
20
21     public void print() {
22         this.printer.print(sayHello());
23     }
24 }
```

Bean Configuration (빈 설정) XML 작성

❖annot.xml

annot.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:
4      xmlns:aop="http://www.springframework.org/schema/aop"
5      xsi:schemaLocation="http://www.springframework.org/schema/be
6          http://www.springframework.org/schema/context http://www
7          http://www.springframework.org/schema/aop http://www.spr
8
9      <!--어노테이션이 선언된 클래스들을 스캔하기 위한 설정 -->
10     <context:component-scan base-package="myspring.di.annot" />
11
12 </beans>
```

DI 테스트 클라이언트 수정

❖ HelloBeanSpringTest.java

HelloBeanSpringTest.java

```
1  package myspring.di.annot.test;
2
3  import static org.junit.Assert.assertEquals;
14
15 @RunWith(SpringJUnit4ClassRunner.class)
16 @ContextConfiguration(locations = "classpath:config/annot.xml")
17 public class HelloBeanSpringTest {
18     @Autowired
19     private ApplicationContext context;
20
21     @Test
22     public void bean1() {
23         Hello hello = (Hello) context.getBean("hello");
24         assertEquals("Hello Spring", hello.sayHello());
25         hello.print();
26
27         Printer printer = context.getBean("stringPrinter",Printer.class);
28         assertEquals("Hello Spring", printer.toString());
29     }
30 }
```

3. 프로퍼티(Property) 파일을 이용한 설정 방법

Properties 파일 및 Bean 설정파일 작성

value.properties

```
myname=Spring
myprinter=printer
value1=JUnit
value2=AOP
value3=DI
printer1=stringPrinter
printer2=consolePrinter
```

annot.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
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/be
        http://www.springframework.org/schema/context http://www
        http://www.springframework.org/schema/aop http://www.spr

    <!--어노테이션이 선언된 클래스들을 스캔하기 위한 설정 -->
    <context:component-scan base-package="myspring.di.annot" />

    <context:property-placeholder
         location="classpath:config/value.properties" />

</beans>
```

어노테이션을 사용한 POJO 클래스 수정 (2)

value.properties

```
myname=Spring
myprinter=printer
value1=JUnit
value2=AOP
value3=DI
printer1=stringPrinter
printer2=consolePrinter
```

Hello.java

```
package myspring.di.annot;

import javax.annotation.Resource;

@Component
public class Hello {
    @Value("${myname}")
    String name;

//  @Autowired
//  @Qualifier("stringPrinter")
    @Resource(name="${printer1}")
    Printer printer;

    public String sayHello() {
        return "Hello " + name;
    }

    public void print() {
        this.printer.print(sayHello());
    }
}
```

학습정리

지금까지 [DI 애플리케이션 작성(4)]에 대해서 살펴보았습니다.

Bean 등록 메타정보 구성 전략

XML 단독사용, XML과 빈 스캐닝의 혼용

Bean 등록 및 의존관계 설정 Annotation

- @Component, @Repository, @Service, @Controller
- @Autowired, @Qualifier, @Value, @Resource

프로퍼티(property) 파일을 이용한 값 설정방법

Properties 파일 작성, ${ } 치환자 사용, <context:property-placeholder>