



# 제6장 스프링의 애너테이션 기반 개발

- 스프링의 애너테이션 기반 개발 지원 방법 및 사용법

# 애너테이션(Annotation)이란?

- ▶ 자바 코드에 추가하여 사용할 수 있는 메타데이터의 일종
- ▶ @기호를 붙여 사용
- ▶ JDK1.5 버전 이상에서 사용
- ▶ 클래스 파일에 적용되어 컴파일러에 의해 생성
- ▶ XML파일을 대체할 수 있는 환경 조성



# 애너테이션 장점(1)



XML로 설정할 때의 모듈 변경 방법

```
<bean id="exam" class="소속클래스명" />
```

```
<bean id="console" class="소속클래스명">
```

```
    <property name="exam" ref="exam" />
```

```
</bean>
```



## 애너테이션 장점(2)

Annotation로 설정할 때의 모듈 변경 방법

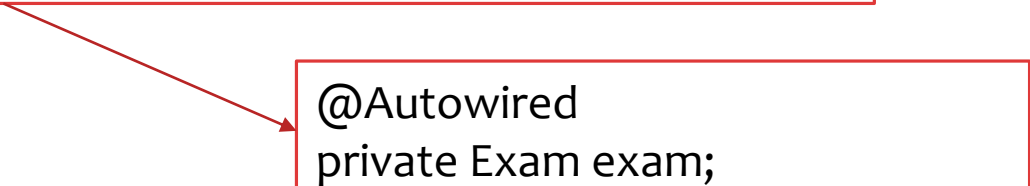
```

@Component
<bean id="exam" class="소속클래스명" />
<bean id="console" class="소속클래스명">
    <property name="exam" ref="exam" />
</bean>

```

// DI구조

`@Autowired`  
private Exam exam;



# 애너테이션 장점(3)

Annotation로 설정할 때의 모듈 변경 방법

...

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

...

**<context:annotation-config />** // @Autowired를 찾아 검색하세요.

<bean id="exam" class="소속클래스명" />

//<bean id="console" class="소속클래스명"> //자바 소스의 @Autowired 을 찾음

</bean>

# 애너테이션 장점(4)

Annotation로 설정할 때의 모듈 변경 방법

```
<bean id="exam" class="소속클래스명" />
```

```
<bean id="console" class="소속클래스명">
```

```
</bean>
```

// DI구조

```
@Autowired //자동으로 객체를 연결해 주세요.  
@Override  
public void setExam(Exam exam){  
    this.exam = exam;  
}
```

중요함

# 애너테이션 장점(5)

Annotation로 설정할 때의 모듈 변경 방법

<bean id="exam1" class="소속클래스명" />

<bean id="exam2" class="소속클래스명">

</bean>

@Autowired //자동으로 객체를 연결해 주세요.

@Qualifier("exam1") ← 서로 다른 객체명이 xml에 있을 경우 이름 지정

@Override

public void setExam(**Exam** exam){

    this.exam = exam;

}

# @Autowired

- ▶ 객체의 타입으로 의존 관계 자동 연결
- ▶ 스프링 빈 객체를 특정 참조 변수에 자동으로 매핑해 줌
- ▶ 생성자 수준, 메소드 수준, 필드 수준으로 사용
- ▶ 자동 생성을 원하는 위치의 상단에 표시
- ▶ `@Autowired(required=true)`
  - ▶ 의존 관계 자동 연결 필수
- ▶ `@Autowired(required=false)`
  - ▶ 의존 관계 연결 선택
  - ▶ 생성자가 여러 개 있어도 됨 : 스프링은 생성자 중 하나를 호출해 빈 클래스 인스턴스를 생성함



# @Component(1)

- ▶ 스프링 빈 식별
  - ▶ 클래스가 스프링 빈을 표현하다는 사실을 나타냄
  - ▶ @Component가 붙은 클래스는 스프링 빈 객체로 등록이 되어 객체를 생성 및 삭제를 스프링에서 관리할 수 있음
- ▶ “빈”을 “스프링 컴포넌트 Component”라고 부르기도 함
- ▶ 컨트롤러, 서비스, 데이터접근객체<sup>DAO</sup>에 따라 특별한 @Component를 사용하도록 권장
  - ▶ 컨트롤러 : @Controller
  - ▶ 서비스 : @Service
  - ▶ 데이터 접근 객체 : @Repository

# @Component(2)

## @Component

```
class 클래스명{  
    @Autowired(required=false)  
    public void setExam(Exam exam){  
        this.exam = exam;  
    }  
}
```

**<context:component-scan base-package="소속패키지" />**  
**<bean id="exam" class="소속클래스" />**

# @Component(3)

**@Component(“찾고자하는 id명 기입”)**

```
class 클래스명{  
    @Autowired(required=false)  
    public void setExam(Exam exam){  
        this.exam = exam;  
    }  
}
```

**<context:component-scan base-package=“소속패키지” />**

**<bean id=“exam” class=“소속클래스” />**

# @Bean 과 @Configuration(1)

## ▶ @Configuration

- ▶ 설정 하고자 하는 클래스 선언부 앞에 선언

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       ...>

  </bean>
}
```

## ▶ @Bean

- ▶ @Configuration으로 선언된 클래스 메소드에서 사용
- ▶ 메서드의 리턴 객체가 스프링 빈 객체임을 알려줌
- ▶ 빈의 식별명은 기본적으로 메소드명이 됨
- ▶ @Bean(name="빈이름")으로 식별명 변경가능
- ▶ @Scope를 통해 객체 생성을 조정할 수 있음
- ▶ @Component를 통해 @Configuration 선언이 없어도 빈 객체를 생성할 수 있음

↓

```
@Configuration
class 클래스명{
    private int a;
    private int b;
    .....
}
```

# @Bean 과 @Configuration(2)

```
<?xml version="" encoding=""?>
```

```
<beans xmlns="" ...>
```

```
    <context:component-scan base-package="소속패키지명, ... " />
```

```
</bean>
```

```
}
```

```
@ComponentScan("소속패키지명") // @ComponentScan({"소속패키지명", "... ", "... " })
```

```
@Configuration
```

```
class 클래스명{
```

```
    private int a;
```

```
    private int b;
```

```
    .....
```

```
}
```

# @Bean 과 @Configuration(3)

```
<?xml version="" encoding=""?>
<beans xmlns="" ...>
    <context:component-scan base-package="소속패키지명, ... " />
    <bean id="exam" class="소속클래스명" />
</bean>
}
```

```
@ComponentScan("소속패키지명") // @ComponentScan({"소속패키지명", "... ", "..."})
@Configuration
class 클래스명{
    ...
    @Bean
    public Exam exam(){
        return ....
    }
}
```

```
...main(String[] args){
    ApplicationContext context = new
        AnnotationConfigApplicationContext(xxx.class);

    .....
}
```

\* ApplicationContext의 종류

- 1) ClassPathXmlApplicationContext
- 2) FileSystemXmlApplicationContext
- 3) XmlWebApplicationContext
- 4) AnnotationConfigApplicationContext

# ○ ○ ○ @Bean 과 @Configuration(4) ○ ○ ○

- ▶ Bean에 이름을 지정하는 방법
  - ▶ 이름이 명시되지 않을 경우
    - ▶ @Component : 소문자로 시작된 클래스명이 자동 사용됨
    - ▶ Bean : 소문자로 시작된 메소드명이 자동으로 사용됨
  - ▶ 이름이 명시된 경우
    - ▶ @Component("식별명")
    - ▶ @Bean(name="식별명")

# @Bean 과 @Configuration(5)

## ▶ 스프링 Bean 생성 예제

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class AA{
    @Bean
    public A eSum(){
        Abc abc = new Abc();
        return abc;
    }

    @Bean
    public A eSum2(){
        Abc2 abc = new Abc2();
        return abc;
    }
}
```





# @Primary



- ▶ @Bean 또는 @Component와 함께 사용되며 객체 생성의 우선권을 부여함

```
@Bean(name="AA")
```

```
....
```

```
@Bean(name="BB")
```

```
@Primary
```

```
....
```

```
@Autowired
```

```
.... BB의 내용이 처리됨...
```

# @Qualifier

- ▶ 빈 이름으로 의존 관계 자동 연결, @Primary 보다 우선권이 높음
- ▶ 빈 이름을 자동연결하기 위해 필드 수준, 메소드 수준, 생성자 수준에서 사용
- ▶ @Autowired와 @Qualifier를 함께 사용하면 의존 관계를 빈 이름으로 자동 연결할 수 있음
  - ▶ 먼저 @Autowired를 설정한 필드, 생성자 인수, 메소드 인수의 객체 타입으로 후보 빈을 찾고, 이후 @Qualifier를 사용해 자동 연결 후보 목록에서 유일한 빈을 구별함

```
@Bean(name="AA")
....

@Bean(name="BB")
....
```

```
@Autowired
@Qualifier("BB")
.... BB의 내용이 처리됨...
```

- ▶ JSR330 자바의존관계 주입(JavaEE6 표준)부터 @Service와 @Qualifier대신 -> @Named 사용 javax.inject.\*
- ▶ JSR330 자바의존관계 주입 (JavaEE6 표준) 부터 @Autowired 대신-> @Inject사용 javax.inject.\*
  - ▶ 생성자 의존관계 주입이 가장 먼저 발생->필드주입->메서드 주입 순으로 일어남

# @Value

- ▶ 빈 클래스 설정
- ▶ 애너테이션을 설정한 스프링 빈은 XML파일에 정의되지 않기 때문에 <property>나 <constructor-arg>엘리먼트의 value와 같은 역할을 함
- ▶ 필드수준, 메소드 수준, 메소드 파라미터 수준, 생성자 인수 수준에서 사용할 수 있음
- ▶ @Value(value="sample") 와 @Value("sample")은 같다
- ▶ @Value 애너테이션에 SpEL<sup>expression language(표현언어)</sup>사용
  - ▶ @Value("#{con.getCountry()}") // 메소드 호출
  - ▶ @Value("#{con.environment}") //프로퍼티값

```
@Component
class 클래스명{
    @Value("10")
    private int a;
    @Value("20")
    private int b;
    .....
}
```

# 그 외

애너테이션	설명
@Scope	빈 스코프 지정(<bean>엘리먼트의 scope속성과 같다)
@Lazy	스프링 컨테이너가 빈을 지연해서 생성할지 지정(<bean>엘리먼트의 lazy-init속성과 같다)
@DependsOn	빈의 암시적 의존 관계를 설정(<bean>엘리먼트의 depends-on속성과 같다)
@Primary	빈을 자동 연결 제1후보로 지정(<bean>엘리먼트의 primary속성과 같다)

JSR380 애너테이션	설명
@NotNull	필드값이 null이 아니어야 함
@Min	지정한 필드값이 최소값보다 크거나 같아야 함
@Max	지정한 필드값이 최대값보다 크거나 같아야 함
@NotBlank	설정한 필드값이 null이나 빈 값이 아니어야 함
@Size	필드의 크기(길이)가 min과 max사이여야 함