Spring Framework
- 스프링 DI - 2

CONTENTS

- 의존관계 자동 설정
- @Autowired 어노테이션을 이용한 의존 자동 주입
- @Resource 어노테이션을 이용한 자동 의존 주입
- 자동 주입과 명시적 의존 주입 간의 관계

의존 관계 자동 설정

- **의존하는 빈 객체의 타입이나 이름을 이용하여 의존객체를 자동으로 설정할 수 있는 기능**
- **autowire속성을 이용**
- **자동 설정과 직접 설정의 혼합도 가능**
- **세 가지 방식**
  - **byName :** 프로퍼티의 이름과 같은 이름을 갖는 빈 객체를 설정
  - **byType :** 프로퍼티의 타입과 같은 타입을 갖는 빈 객체를 설정
  - **constructor :** 생성자 파라미터 타입과 같은 타입을 갖는 빈 객체를 생성자에 전달

의존 관계 자동 설정

- **특정방식을 이용해서 의존 객체를 설정하려면 <bean>태그에 autowire 속성값을 지정해주면 된다.**

```
<bean id="greeting" class="GreetingServiceImpl" autowire="byName"/>
```

```
<?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-3.0.xsd"
default-autowire="byName" >

<bean id="articleDao" class="service.chap01.MySQLArticleDao">
</bean>

</beans>
```

의존 관계 자동 설정

- **byName**
  - 프로퍼티 이름과 동일한 이름을 갖는 빈 객체를 프로퍼티 값으로 설정

```
…
private OutputService outputter;
public void setOutputter(OutputService outputter) {
        this.outputter = outputter;
}
…
```

```
…
<bean id="greeting" class="autosetting.GreetingServiceImpl"  autowire="byName"/>
<bean id="outputter" class="autosetting.OutputServiceImplConsole"/>
…
```

빈 객체 범위

- 기본적으로 컨테이너에 한 개의 빈 객체를 생성
- 빈의 범위를 설정할 수 있는 방법을 제공
- scope 속성을 이용 범위 설정

범위	설명
singleton	컨테이너에 한 개의 빈 객체만 생성한다.(기본값)
prototype	빈을 요청할 때마다 빈 객체를 생성한다.
request	HTTP 요청마다 빈 객체를 생성한다.(WebApplicationContext 에서만 적용)
session	HTTP 세션마다 빈 객체를 생성한다.(WebApplicationContext 에서만 적용)
global-session	클로벌 HTTP 세션에 대해 빈 객체를 생성한다. 포틀릿을 지원하는 컨텍스트에 대해서만 적용가능

빈 객체 범위

```
…
GreetingService bean = (GreetingService)factory.getBean("greeting");
GreetingService bean2 = (GreetingService)factory.getBean("greeting");
System.out.println(bean == bean2); // true
…
```

```
…
<bean id="greeting" class="beanscope.GreetingServiceImpl"  scope="singleton">
        <property name="greeting">
                <value>Hello</value>
        </property>
</bean>
…
```

빈 객체 범위

```
…
GreetingService bean = (GreetingService)factory.getBean("greeting");
GreetingService bean2 = (GreetingService)factory.getBean("greeting");
System.out.println(bean == bean2); // false
…
```

```
…
<bean id="greeting" class="beanscope.GreetingServiceImpl"  scope="prototype">
        <property name="greeting">
                <value>Hello</value>
        </property>
</bean>
…
```

어노테이션 기반 설정

- 어노테이션이란?
  - JDK5 버전부터 추가된 것으로 메타데이터를 XML등의 문서에 설정하는 것이 아니라 소스 코드에 "@어노테이션"의 형태로 표현하며 클래스, 필드, 메소드의 선언부에 적용 할 수 있는 특정 기능이 부여된 표현법
- 어노테이션 사용 이유
  - 프레임워크들이 활성화 되고 애플리케이션 규모가 커질수록 XML 환경 설정은 복잡해지는데, 이러한 어려움을 개선시키기 위하여 자바 파일에 어노테이션을 적용해서 코드를 작성함으로써 개발자가 설정 파일에 작업하게 될 때 발생시키는 오류의 발생 빈도를 낮춰주기도 함
  - 어노테이션을 사용하면 소스 코드에 메타데이터를 보관할 수 있으며 컴파일 타임의 체크뿐 아니라 어노테이션 API를 사용하여 코드의 가독성을 높일 수도 있음

어노테이션 기반 설정

- **주석과 어노테이션의 차이점**
  - //과, @ 은 개발자가 보기에는 똑같이 실행되지 않는 코드 같지만 실제로 //은 컴파일러가 실행은 안 해도 @은 컴파일러가 실행되기 전에 어노테이션으로 설정한 내용대로 코드가 작성되었는지 확인하기 위해 실행을 함
  - "@Override" 어노테이션이 선언된 메소드를 Eclipse 툴에서 재정의 할 경우 재정의 문법에 위배된 코드로 개발이 되면 소스 화면에 문법 오류 발생이라는 표현을 해 주기 때문에 개발자의 실수도 쉽게 알 수 있으며, 빠른 시간에 문제점에 대한 해결을 하게 됨
  - 개발자가 환경 설정에서의 실수를 했을 경우에도 수정해주는 역할을 하기도 하고, 메타정보를 선언하게 되는 XML 설정 파일이 점점 복잡해지지 않도록 자바 코드 안에서 설정하므로 개발자의 도움말이 되는 역할을 함

■ @(어노테이션)

- **@Component**
  - 스프링 설정 파일에 <context:component-scan> 태그를 추가하면 어노테이션이 적용된 클래스를 빈으로 등록됨.
    - @Required(RequiredAnnotationPostProcessor)
    - @Autowired(AutowiredAnnotationBeanPostProcessor)
    - @Resource
    - @PostCounstruct
    - @PreDestory(CommonAnnotationBeanPostProcessor)
    - @Configuration(ConfigurationClassPostProcessor)
    - 그 외 Repository, Service, Controller 포함
    - <context:component-scan base-package="xxx"/>
      xxx 패키지 하위에 @Component로 선언된 클래스를 bean으로 자동등록(bean 이름 : 해당클래스 이름, 첫글자 소문자)

@(어노테이션)

- **Spring framework Annotation**
  - **결합도를 낮추고 유지보수성을 높이기 위해 xml로 설정하였으나 xml이 너무 많아지면 오히려 유지보수성이 낮아지는 아이러니한 상황 발생**
  - **유지보수성을 목적으로 함.**
  - **시스템 전체에 영향을 주고 이후에 변경 가능성이 있는 것은 xml로 설정.**

어노테이션 기반 설정

- **어노테이션 문법**
  - @어노테이션"[어노테이션]"의 형태로 표현하며 클래스, 필드, 메소드의 선언부에 적용 할 수 있고 특정 기능이 부여된 표현법입니다

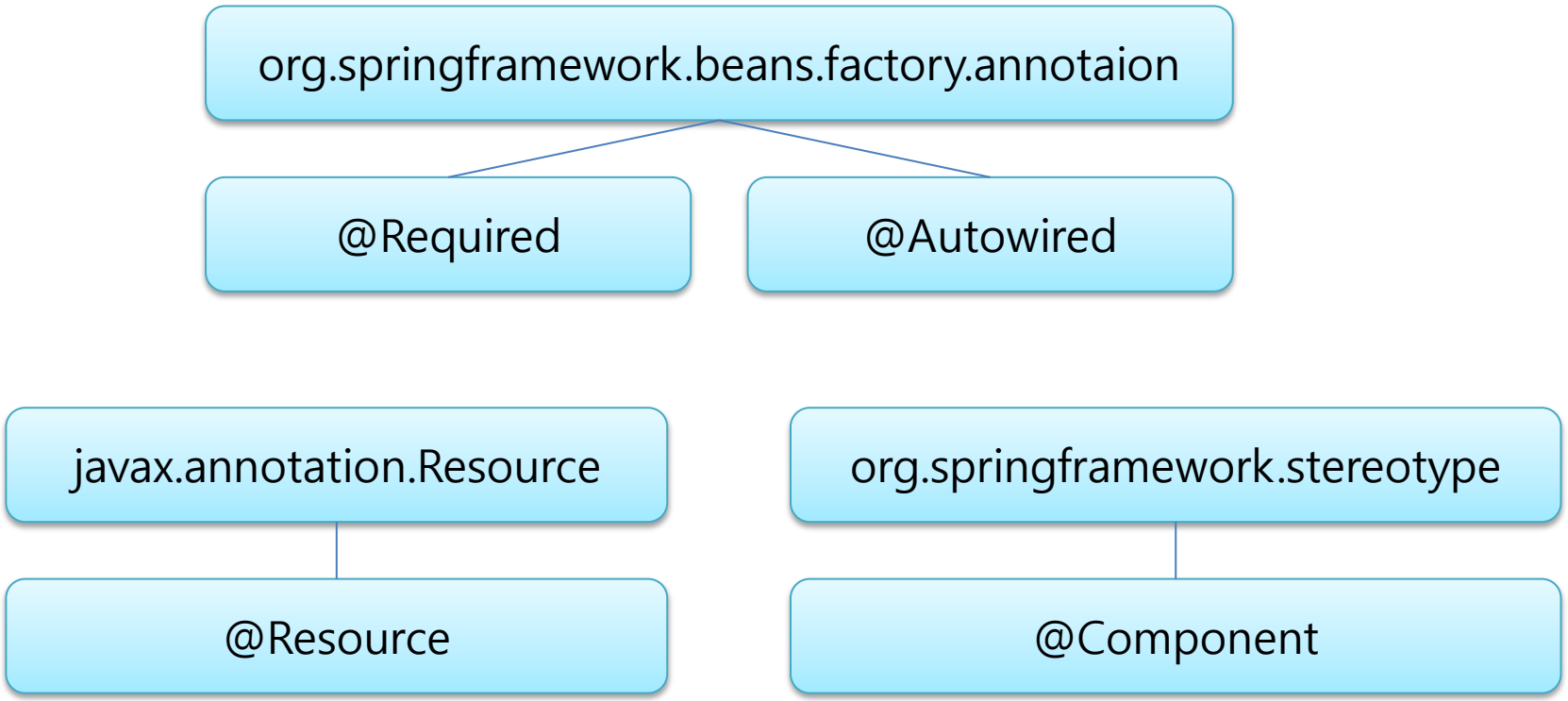
```
 9 @Component
10 public class PersonServiceImpl implements PersonService{
11     private String message;
12
13     @Resource(name="person1")
14     private Person person1;
15
16     @Resource(name="person2")
17     private Person person2;
```

어노테이션 기반 설정

- **Spring2.5 버전 부터는 어노테이션을 이용하여 빈과 관련된 정보를 설정할 수 있게 되었으며, 복잡한 XML 문서 생성과 관리에 따른 수고를 덜어주어 개발 속도를 향상 시킬 수 있음**

어노테이션	설 명
@Required	setter주입방식을 이용한 어노테이션, 필수 프로퍼티를 명시할 때 사용
@Autowired	타입을 기준으로(byType) 빈을 찾아 주입하는 어노테이션
@Resource	이름을 기준으로(byname) 빈을 찾아 주입하는 어노테이션
@Component	빈 스캐닝 기능을 이용한 어노테이션

어노테이션 기반 설정


org.springframework.beans.factory.annotaion
@Required
@Autowired
javax.annotation.Resource
@Resource
org.springframework.stereotype
@Component

@Autowired 어노테이션을 이용한 의존 자동 주입

- 자동 주입 대상에 @Autowire 어노테이션 사용
- XML 설정에 <context:annotation-config /> 설정 추가

```
public class MemberRegisterService {
        private MemberDao memberDao;

        @Autowired
        public MemberRegisterService(MemberDao memberDao) {
                this.memberDao = memberDao;
        }

        public MemberRegisterService() {
        }

        ....
```

@Autowired 어노테이션을 이용한 의존 자동 주입

```
<?xml version="1.0" encoding="UTF-8"?>

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

        <context:annotation-config />

        <bean id="memberDao" class="spring.MemberDao">
        </bean>

        <bean id="memberRegSvc" class="spring.MemberRegisterService">
        </bean>

        <bean id="memberPrinter" class="spring.MemberPrinter">
        </bean>

        <bean id="infoPrinter" class="spring.MemberInfoPrinter">
        </bean>

 </beans>
```

@Qualifier 어노테이션을 이용한 의존 객체 선택

- **@Qualifier 어노테이션은 사용할 의존 객체를 선택 할 수 있도록 해준다.**
- **@Qualifier 어노테이션을 사용하려면 아래 두 가지 설정이 필요**
  - 설정에서 빈의 한정자 설정
  - @Autowired 어노테이션이 적용된 주입 대상에 @Qualifier 어노테이션을 설정
    이때 @Qualifier 어노테이션의 값으로 앞서 설정한 한정자 사용

```
<bean id="printer1" class="spring.MemberPrinter">
        <qualifier value="sysout" />
</bean>

<bean id="printer2" class="spring.MemberPrinter" />

<bean id="infoPrinter" class="spring.MemberInfoPrinter">
</bean>
```

@Qualifier 어노테이션을 이용한 의존 객체 선택

```
<bean id="printer1" class="spring.MemberPrinter">
        <qualifier value="sysout" />
</bean>

<bean id="printer2" class="spring.MemberPrinter" />

<bean id="infoPrinter" class="spring.MemberInfoPrinter">
</bean>
```

```
@Autowired
@Qualifier("sysout")
public void setPrinter(MemberPrinter printer) {
        System.out.println("setPrinter: " + printer);
        this.printer = printer;
}
```

@Autowired 의 필수 여부 지정

- **@Autowired(required=false)**

```
public class MemberRegisterService {
        private MemberDao memberDao;

        @Autowired(required=false)
        public MemberRegisterService(MemberDao memberDao) {
                this.memberDao = memberDao;
        }

        public MemberRegisterService() {
        }

        ....
```

@Autowired 어노테이션의 적용 순서

1. 타입이 같은 빈 객체를 검색한다. 한 개면 그 객체를 사용한다.
   @Qualifier가 명시되어 있을 경우, @Qualifier와 같은 값을 갖는 빈 객체이어야 한다.
2. 타입이 같은 빈 객체가 두 개 이상 존재하면, @Qualifer로 지정한 빈 객체를 찾는다. 존재하면 그 객체를 사용한다.
3. 타입 같은 빈 객체가 두 개 이상 존재하고, @Qualifier가 없을 경우, 이름이 같은 빈 객체를 찾는다. 존재하면, 그 객체를 사용한다.

@Resource 어노테이션을 이용한 자동 의존 주입

- **@Autowired 어노테이션은 타입을 이용해서 주입할 객체를 검색**
- **@Resource 어노테이션은 빈의 이름을 이용해서 주입할 객체를 검색**
- **@Resource 어노테이션을 사용하려면 두 가지 설정이 필요**
  - 자동 주입 대상에 @Resource 어노테이션 사용
  - XML설정에 <context:annotation-config /> 설정 추가

```
public class MemberRegisterService {
        @Resource(name="memberDao")
        private MemberDao memberDao;

        //@Autowired
        public MemberRegisterService(MemberDao memberDao) {
                this.memberDao = memberDao;
        }

        public MemberRegisterService() {
        }
        ....
```

@Resource 어노테이션의 적용순서

1. name 속성에 지정한 빈 객체를 찾는다. 존재하면 해당 객체를 주입할 객체로 사용한다
2. name 속성이 없을 경우, 동일한 타입을 갖는 빈 객체를 찾는다. 존재하면 해당 객체를 주입할 객체로 사용한다.
3. name 속성이 없고, 동일한 타입을 갖는 빈 객체가 두 개 이상일 경우, 같은 이름을 가진 빈 객체를 찾는다. 존재하면 해당 객체를 주입할 객체로 사용한다.
4. name 속성이 없고, 동일한 타입을 갖는 빈 객체가 두 개 이상이고 같은 이름을 가진 빈 객체가 없을 경우, @Qualifer를 이용해서 주입할 빈 객체를 찾는다.

자동 주입과 명시적 의존 주입 간의 관계

- 자동 주입과 명시적 의존 주입 설정이 함께 사용되는 경우 명시적인 의존 주입 설정이 우선한다.

@Required

- Required는 필수 프로퍼티임을 명시하는 것으로 설정하지 않을 경우 빈 생성시 예외를 발생
  - public class TestBean{

```
@Required
private TestDao testDao;
public void setTestDao(TestDao testDao){
    this.testDao = testDao;
}
```

}

  - RequiredAnnotationBeanPostProcessor 클래스는 스프링 컨테이너에 등록된 bean 객체를 조사하여 @Required 어노테이션으로 설정되어 있는 프로퍼티의 값이 설정되어 있는지 검사.
  - <bean class="org.springframework.beans.factory.annotation.RequiredAnnotationBeanPostProcessor" /> 클래스를 빈으로 등록시켜줘야 하지만, <context:annotation-config> 태그를 사용

@Resource

- **@Autowired(by type), @Resource(by name)으로 연결**

  ***option**

  name : 연결될 빈 객체 이름 입력 @Resource(name="test")

@Scope

- 범위 옵션 : prototype, singleton, request, session, globalSession
- default 범위 "singleton"

```
@componet
@Scope(value="prototype")
public class User(){}
```

@PostConstruct, @PreDestory

- **@PostConstruct**

  해당 작업 메서드 앞에 설정. 의존하는 객체를 설정한 이후에 초기화 작업을 수행하기 위해 사용. 스프링에 의해 인스턴스가 생성된 후 어노테이션이 적용된 메서드 호출.

  <context:annotation-config> 태그로 설정

```
@PostConStruct
public void init(){
    Sysout("init...");
}
```

- **@PreDestory**

  해당 작업 메서드 앞에 설정, 컨테이너에서 객체를 제거하기 전에 해야 할 작업을 수행하기 위해 사용.

  <context:annotation-config> 태그로 설정

빈 객체 스캔

- XML 문서와 같이 한곳에 명시적으로 선언하지 않고 빈으로 사용될 클래스에 특별한 어노테이션을 부여해주면 특정 어노테이션이 붙은 클래스를 자동으로 찾아서 빈으로 등록해주는 방식
  - 빈 스캐닝 – 빈 클래스에 어노테이션을 선언해 둘 경우 어노테이션이 붙은 클래스들을 자동으로 찾아서 빈으로 등록해주는 방식의미
  - 빈 스캐너(bean scanner) – 스캐닝 작업을 담당하는 객체
- XML 설정 파일에 다양한 빈 정보를 추가하지 않고도 특정한 클래스를 빈으로 등록해 줌
- 빈 스캐너의 작동 원리
  - 지정된 클래스패스 아래에 있는 모든 패키지의 클래스를 대상으로 빈 스캐너가 자바 코드에 선언된 어노테이션 기능에 맞게 자동으로 선별해 내서 Spring 빈으로 설정

빈 객체 스캔 : @Component

- 클래스 패스에 위치한 클래스를 검색하여 @Component 어노테이션이 붙은 클래스를 자동으로 빈으로 등록하는 기능
- XML 설정 파일에 다양한 빈 정보를 추가하지 않고 특정한 클래스를 빈으로 등록 가능
- 검색된 클래스를 빈으로 등록할 경우 클래스의 이름을 빈의 이름으로 사용(첫 글자는 소문자로 변경)
- 기본 범위는 “singleton”, @Scope 어노테이션을 통해 범위 설정

  ex> @Scope(“prototype”)

```
@Component
public class SpringBean{
 …
}
```

```
…
SpringBean sb =
(SpringBean)context.getBean(“springBean”);
```

빈 객체 스캔 : @Component

- 설정 파일
  - <context:component-scan base-package=“package명"/>
    : 클래스를 검색할 패키지 지정
- 적용 위치 – 클래스 선언구 부분에 위치

빈 객체 스캔 대상 클래스 범위 지정

- **<context:include-filter>, <context:exclude-filter> 태그를 사용해서 자동 스캔 대상에 포함시킬 클래스와 포함시키지 않을 클래스를 구체적으로 명시할 수 있다.**

```
<context:component-scan base-package="spring" scoped-proxy="no">
<context:include-filter type="regex" expression="kame₩.spring₩.chap04₩.work₩.*"/>
<context:include-filter type="regex"
expression="kame₩.spring₩.chap04₩.homecontrol₩.*"/>
</context:component-scan>
```

@Inject, @Service, @Repository, @Contoroller

- **@Inject**

  Spring 3부터 지원하는 Annotation. 특정 framework에 종속되지 않은 어플리케이션을 구성하기 위해서 @Inject를 사용할 것을 권장. 의존 설정 (@autowored , @resource 와 비슷함)

- **@Service**

  비즈니스로직이 들어가는 Service 빈 등록

- **@Repository**

  일반적으로 DAO에 사용되며 DB Exception을 DataAcessException으로 변환한다.

- **@Contoroller**

  Spring MVC의 Controller 클래스 선언을 단순화. 스프링 컨트롤러

  @Controller로 등록된 클래스 파일에 대한 bean을 자동으로 생성해준다.(component-scan 이용)

@(어노테이션)

- **@RequestMapping**
  사용자 요청 URL 매핑
- **@PathVariable**
  URL 템플릿 변수 설정
- **@RequestParam**
  HTTP 파라미터와 매핑
- **@RequestHeader**
  HTTP 헤더 매핑
- **@CookieValue**
  HTTP 쿠키 매핑
- **@RequestBody**
  HTTP RequestBody에 접근할 때 사용. HttpMessage Converter를 이용해 RequestBody 데이터를 해당 타입으로 변환
- **@ResponseBody**
  @ResponseBody 어노테이션이 적용된 경우, 리턴 객체를 HTTP 응답으로 전송한다.
  HttpMessageConverter를 이용해서 객체를 HTTP 응답 스트림으로 변환한다.