Spring Framework
- 스프링 DI - 2

CONTENTS

- @Autowired 애노테이션을 이용한 의존 자동 주입
- @Resource 애노테이션을 이용한 자동 의존 주입
- 자동 주입과 명시적 의존 주입 간의 관계

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)

빈 객체 범위

```
Dao dao1 = ctx.getBean("memberDao", Dao.class);
Dao dao2 = ctx.getBean("memberDao", Dao.class);

System.out.println("dao1==dao2 => " + (dao1==dao2)); // true

// 4. MemberInforService 객체
MemberInfoService infoService1 = ctx.getBean("memberInfoService",
MemberInfoService.class);
MemberInfoService infoService2 = ctx.getBean("memberInfoService",
MemberInfoService.class);

System.out.println("infoService1==infoService2 =>
"+(infoService1==infoService2)); // true
```

```
<!-- MemberDao Bean(인스턴스)으로 등록 -->
<bean id="memberDao" class="member.dao.MemberDao" ></bean>

<!-- MemberInfoService Bean 등록 -->
<bean id="memberInfoService" class="member.service.MemberInfoService“
        scope="singleton"
>
```

빈 객체 범위

```
// 2. MemberRegService 객체가 필요
MemberRegService regService1 = ctx.getBean("memberRegService",
MemberRegService.class);
MemberRegService regService2 = ctx.getBean("memberRegService",
MemberRegService.class);

System.out.println("regService1==regService2 => "+
(regService1==regService2)); // false
```

```
<bean id="memberRegService"
        class="member.service.MemberRegService"
        scope="prototype"
 >
```

애노테이션 기반 설정

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

애노테이션 기반 설정

- **애노테이션 문법**
  - @애노테이션"[애노테이션]"의 형태로 표현하며 **클래스, 필드, 메소드의 선언부**에 적용할 수 있고 특정 기능이 부여된 표현법입니다

```
@Component
public class MemberInfoService3 implements MemberService {

        @Resource(name = "memberDao")
        Dao dao ;

        @Override
        public Object process() {
                System.out.println("MemberInfoService 인스턴스 실행");
                dao.select();
                return null;
        }

}
```

애노테이션 기반 설정

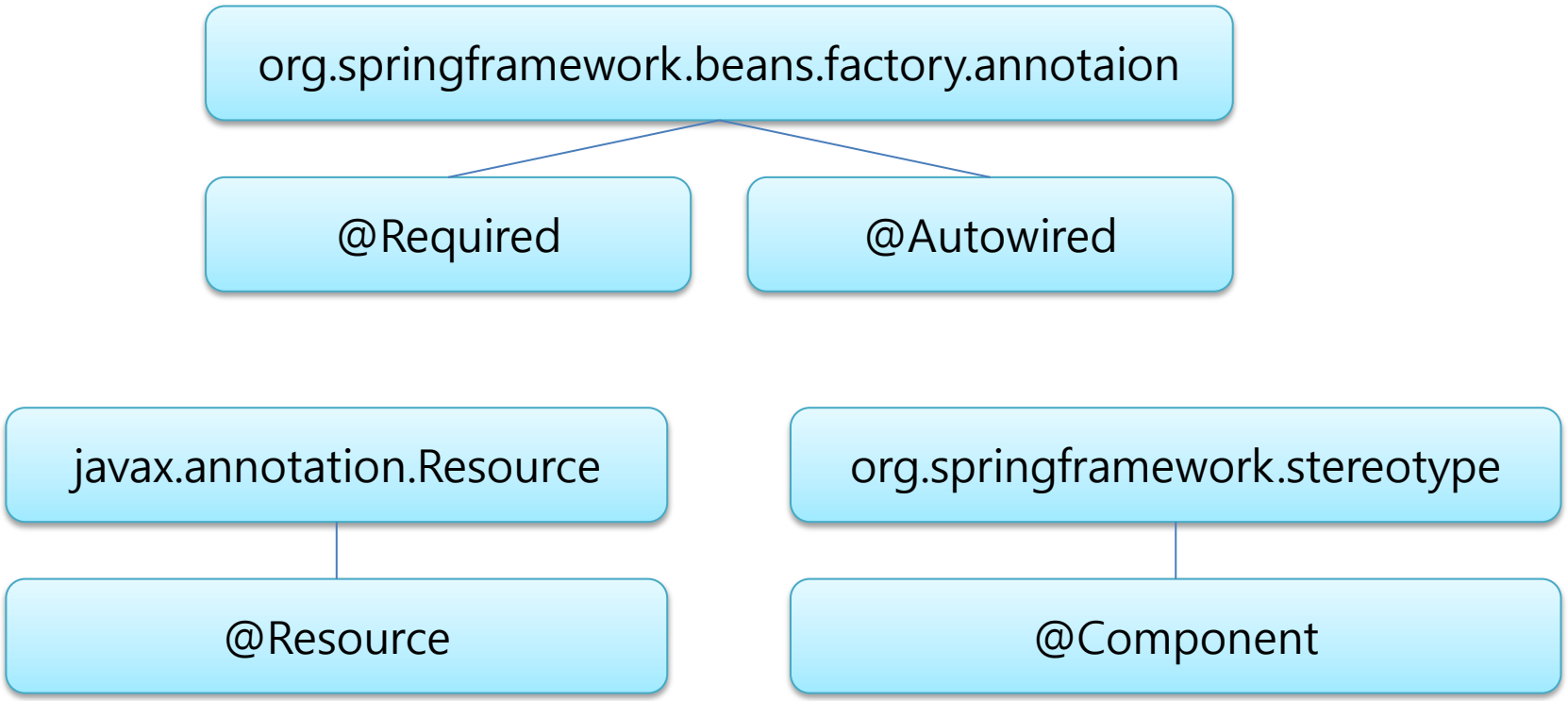
- **주석과 애노테이션의 차이점**
  - //과, @ 은 개발자가 보기에는 똑같이 실행되지 않는 코드 같지만 실제로 //은 컴파일러가 실행은 안 해도 @은 컴파일러가 실행되기 전에 애노테이션으로 설정한 내용대로 코드가 작성되었는지 확인하기 위해 실행을 함
  - "@Override" 애노테이션이 선언된 메소드를 Eclipse 툴에서 재정의 할 경우 재정의 문법에 위배된 코드로 개발이 되면 소스 화면에 문법 오류 발생이라는 표현을 해 주기 때문에 개발자의 실수도 쉽게 알 수 있으며, 빠른 시간에 문제점에 대한 해결을 하게 됨
  - 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

애노테이션 기반 설정

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

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

애노테이션 기반 설정


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

@Autowired 애노테이션을 이용한 의존 자동 주입

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

```
public class MemberRegService2 implements MemberService  {

        @Autowired
        private Dao dao; // 주입 받아야 하는 참조 변수

        @Override
        public Object process() {
                System.out.println("MemberRegService 실행");
                dao.insert();
                return null;
        }

}
```

@Autowired 애노테이션을 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/>

        <!-- MemberDao Bean(인스턴스)으로 등록 -->
        <bean id="memberDao" class="member.dao.MemberDao"/>

        <!-- MemberRegService Bean으로 등록 -->
        <bean id="memberRegService" class="member.service.MemberRegService2"/>

        <!-- MemberInfoService Bean 등록 -->
        <bean id="memberInfoService" class="member.service.MemberInfoService2"/>


</beans>
```

@Qualifier 애노테이션을 이용한 의존 객체 선택

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

```
<!-- MemberDao Bean(인스턴스)으로 등록 -->
<bean id="memberDao" class="member.dao.MemberDao">
        <qualifier value="member"/>
</bean>

<!-- GuestDao Bean 등록 -->
<bean id="guestDao" class="member.dao.GuestDao">
        <qualifier value="guest"/>
</bean>
```

@Qualifier 애노테이션을 이용한 의존 객체 선택

```xml
<!-- MemberDao Bean(인스턴스)으로 등록 -->
<bean id="memberDao" class="member.dao.MemberDao">
        <qualifier value="member"/>
</bean>

<!-- GuestDao Bean 등록 -->
<bean id="guestDao" class="member.dao.GuestDao">
        <qualifier value="guest"/>
</bean>
```

```java
public class MemberRegService2 implements MemberService  {

    @Autowired
    @Qualifier("member")
    private Dao dao; // 주입 받아야 하는 참조 변수

    @Override
    public Object process() {
        System.out.println("MemberRegService 실행");
        dao.insert();
        return null;
    }
}
```

@Autowired 의 필수 여부 지정

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

```
public class MemberRegService2 implements MemberService  {

    @Autowired(required = false)
    @Qualifier("member")
    private Dao dao; // 주입 받아야 하는 참조 변수

    @Override
    public Object process() {
        System.out.println("MemberRegService 실행");
        dao.insert();
        return null;
    }

}
```

@Autowired 애노테이션의 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 애노테이션을 이용한 자동 의존 주입

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

```
public class MemberRegService3 implements MemberService  {

    @Resource(name = "guestDao")
    private Dao dao; // 주입 받아야 하는 참조 변수

    @Override
    public Object process() {
        System.out.println("MemberRegService 실행");
        dao.insert();
        return null;
    }

}
```

@Resource 애노테이션의 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.