* **오토와이어링의 종류**

|  |  |
| --- | --- |
| byName | 해당 빈 프로퍼티와 일치하는 이름(또는 ID)을 가지는 빈 정의가 발견되는 경우 오토와이어링한다. 해당하는 빈이 없으면 아무 일도 하지 않는다. |
| byType | 해당 빈 프로퍼티와 일치하는 타입을 가지는 빈 정의가 발견되는 경우 오토와이어링한다. 해당하는 빈이 없으면 아무 일도 하지 않는다. |
| constructor | byType과 유사하지만, 해당 프로퍼티를 설정하기 위한 생성자가 존재하는 경우 오토와이어링한다. |
| autodetect | constructor 오토와이어링을 먼저 적용한다. 실패하면 byType을 적용한다. |

**byName**

|  |
| --- |
| <bean id=”kenny” class=”클래스 경로1”>  <property name=”song” value=”변수 값”/>  <property name=”instrument” ref=”saxophone”/>  </bean>  <bean id=”saxophone” class=”클래스 경로2”/> |

|  |
| --- |
| <bean id=”kenny” class=”클래스 경로1” autowire=”byName”>  <property name=”song” value=”변수 값”/>  </bean>  <bean id=”instrument” class=”클래스 경로2”/> 🡨 Kenny 빈의 property 이름과 동일하게 바꿈 |

autowire 애트리뷰트를 byName으로 지정하면 스프링은 지정된 빈의 모든 프로퍼티는 같은 이름을 가진 빈과 와이어링한다. 이때 byName을 위해 세터 주입 방식이 사용된다.(????) byName방식의 단점은 주입하려는 빈의 이름과 대상 프로퍼티의 이름이 같아야한다는 점이다. 이럴 경우 대상 클래스(클래스 경로1)의 빈이 여러 개인 경우 서로 다른 빈을 갖지 못하고 모두 프로퍼티 이름과 같은 빈 하나를 공유해야만 한다.

**byType**

byName 대신 byType 방식을 쓰게되면 빈 ID를 프로퍼티명과 동일하게 맞춰주지 않아도 되는 장점이 있지만 동일한 타입의 빈을 여러 개 정의할 수 있기 때문에 예외를 발생시킨다. 이 같은 모호함을 극복하기위해 스프링은 두가지 방법을 제공한다.

1. 오토와이어링을 위한 기본 후보를 식별하는 방식

|  |
| --- |
| <bean id=”instrument” class=”클래스 경로” primary=”true”/>  <bean id=”saxophone” class=”클래스 경로”  primary=”false”/> |

primary 애트리뷰트를 true로 설정하면 다른 빈들보다 우선적으로 와이어링될 수 있다. 대신 나머지 다른 후보들은 모두 false로 설정해주어야하는 단점이 있다.

1. 오토와이어링 후보에서 빈을 제거하는 방식

|  |
| --- |
| <bean id=”saxophone” class=”클래스 경로” autowire-candidate=”false”/> |

autowire-candidate 애트리뷰트로 와이어링 후보 빈에서 제거할 수 있다.

**constructor-arg**

|  |
| --- |
| <bean id=”빈 이름” class=”클래스 경로1”>  <constructor-arg>  <bean class=”클래스 경로2”  </constructor-arg>  </bean> |

|  |
| --- |
| <bean id=”빈 이름” class=”클래스 경로1” autowire=”constructor”/> |

생성자 주입 방식을 사용하는 빈 정의에서 대신 autowire 애트리뷰트에 constructor를 사용하여 정의하는 빈의 프로퍼티의 생성자 타입과 일치하는 빈을 찾아 주입해준다. byType 방식과 마찬가지로 해당 타입의 빈이 여러 개인 경우는 고려해주지 않는다. constructor 타입의 오토와이어링을 사용하는 경우에는 contructor-arg 애트리뷰트를 사용해서 혼용할 수 없다.

**autodetect**

autowire 애트리뷰트에 autodetect를 설정하면 스프링이 알아서 와이어링 방식을 결정해 준다. 먼저 constructor방식으로 와이어링을 시도하고 적합한 생성자를 찾지 못하면 다시 byType으로 시도해본다.

* **기본 오토와이어링**

|  |
| --- |
| <?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=”byType”>  </beans> |

애플리케이션 컨텍스트에 있는 모든 빈에 동일한 qutowire 애트리뷰트를 지정해야 한다면 루트 <beans> 엘리먼트에 default-autowire 애트리뷰트를 추가한다. 또한 빈별로 autowire 애트리뷰트를 설정해주면 우선적으로 적용가능하다.

* **명시적 와이어링과 혼용**

오토와이어링 설정을 해주었어도 해당 빈의 프로퍼티 값을 명시적으로 지정할 수 있다. 이는 byType 방식에서의 모호함을 보완해준다.

* **애너테이션을 이용한 와이어링**

|  |
| --- |
| <?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:context=”http://www.springframework.org/schema/context”  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 사용**

오토와이어링 되길 원하는 변수나 메소드, 생성자에 명시하여 byType 방식으로 오토와이어링한다. setter 메소드에만 사용할 필요가 없고 private 키워드로도 막을 수 없다. 그러나 와이어링 가능한 빈이 없거나 여러 개인 경우 예외가 발생한다. 아래의 세가지 방법으로 이러한 예외를 막을 수 있다.

1. 선택적 오토와이어링

@Autowired(required=false)

와이어링 가능한 빈이 하나도 없는 경우 required 애트리뷰트를 false로 설정하여 해당 프로퍼티에는 null 값을 부여해준다. 그러나 생성자와 함께 사용하는 경우에는 하나의 생성자만 required 값을 true로 설정해주고 나머지 @Autowired 애너테이트된 생성자들에는 false를 설정해주어야한다.

1. 모호한 종속객체 제한하기

@Autowired

@Qualifier(“빈 이름”)

private Instrument instrument;

와이어링될 빈이 여러 개인 경우 @Qualifier 애너테이션을 사용하여 빈 ID를 입력하여 지정할 수 있다. 이것은 오토와이어링할 빈 후보의 선택 범위를 좁히기위한 방법 중 하나이다.

1. 커스텀 한정자 생성

**@Inject**

- 표준기반의 오토와이어링 적용 애너테이션.

- 스프링의 @Autowired를 거의 완벽하게 대체하지만 required 애트리뷰트를 제공하지않아 오토와이어링에 실패하면 예외를 발생시킨다.

- 레퍼런스를 직접 주입하는 대신에 Provider에 주입할 수 있다.

- 모호한 빈을 정의해주기 위해 @Autowired의 @Qualifier 대신에 @Name 애너테이션을 사용하여 주입할 빈을 구체적으로 식별한다.

- @Inject의 @Qalifier는 자체적으로 사용되지는 않지만 스프링의 @Qalifier와 마찬가지로 커스텀 한정자 애너테이션을 만드는데 사용한다.

**@Value**

- int, Boolean, String 같은 원시 값을 와이어링한다.

- SpEL 표현식을 사용하여 동적인 값을 부여해 줄 수 있다.

* **오토디스커버리**

|  |
| --- |
| <?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:context=”http://www.springframework.org/schema/context”  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=”패키지\_경로”/>  </beans> |

<context:annotation-config>대신 <context:component-scan>를 사용해 스프링 애플리케이션에 있는 빈을 <bean>으로 선언해주지 않아도 자동으로 선언해줄 수 있다. 패키지 경로에는 스캔 시작 패키지 경로를 작성해주면된다.

* **오토디스커버리를 위한 빈 애너테이션**

|  |  |
| --- | --- |
| @Component | 범용 스테레오타입 애너테이션은 클래스가 스프링 컴포넌트임을 나타낸다. |
| @Controller | 클래스가 스프링 MVC 컨트롤러를 정의함을 나타낸다. |
| @Repository | 클래스가 데이터 저장소를 정의함을 나타낸다. |
| @Service | 클래스가 서비스를 정의함을 나타낸다. |
| @Component로 애너테이트된 커스텀 애너테이션 | |

- @Component 애너테이션은 XML 설정에서 명시적인 <bean> 선언을 제거할 수 있다. 스프링이 시작 패키지를 스캔할 때 @Component로 애너테이트된 클래스를 만나면 자동으로 스프링에 등록한다. 또한 @Component(“빈ID”)와 같은 방법으로 빈 ID를 지정해 줄 수 있다.

**component-scan 필터링**

|  |
| --- |
| <context:component-scan base-package=”패키지\_경로”>  <context:include-filter type=”필터타입” expression=”적용여부를\_결정할\_형식”/>  <context:exclude-filter type=”필터타입” expression=”적용여부를\_결정할\_형식”/>  </context:component-scan> |

필터링 기능의 서브 엘리먼트를 사용하여 빈 후보를 유연성있게 스캔할 수 있다. 아래 다섯 가지 종류의 필터를 사용하여 등록가능한 클래스와 등록하지말아야 할 클래스를 필터링하게 해준다.

|  |  |
| --- | --- |
| annotation | 클래스에 지정한 애너테이션이 적용됐는지의 여부.  expression 애트리뷰트에는 “org.example.SomeAnnotation”와 같은 애너테이션 이름을 입력한다. |
| assignable | 클래스에 지정한 타입으로 할당 가능한지의 여부.  expression 애트리뷰트에는 “org.exampleSomeClass”와 같은 타입 이름을 입력한다. |
| aspectj | 클래스 이름이 AspectJ의 표현식에 일치하는지의 여부.  expression 애트리뷰트에는 “org.example..\*Service+”와 같이 AspectJ의 표현식을 입력한다. |
| custom | org.springframework.core.type.TypeFilter의 커스텀 구현체를 사용.  expression 애트리뷰트에 지정한다. |
| regex | 클래스 이름이 정규 표현식에 일치하는지의여부.  expression 애트리뷰트에는 “org\.example\.Default.\*”와 같이 정규 표현식을 입력한다. |

* **스프링의 자바 기반 설정 사용**

스프링은 component-scan에서 정의된 패키지 경로에서 @Configuration으로 애너테이트 된 클래스를 찾는다. @Configuration이 적용된 클래스는 빈 선언 메소드를 추가해 빈을 선언할 수 있다. 빈 선언은 아래와 같이 메소드에 @Bean을 애너테이트 한다.

|  |
| --- |
| @Bean  public 빈의\_추상객체 빈ID(){  return new 빈();  } |