SPRING INJECTION WITH @RESOURCE, @AUTOWIRED AND @INJECT

**Overview**

I’ve been asked several times to explain the difference between injecting Spring beans with ‘@Resource’, ‘@Autowired’, and ‘@Inject’. While I received a few opinions from colleagues and read a couple of posts on this topic I didn’t feel like I had a complete picture.

**Annotations**

|  |  |  |
| --- | --- | --- |
| **ANNOTATION** | **PACKAGE** | **SOURCE** |
| @Resource | javax.annotation | Java |
| @Inject | javax.inject | Java |
| @Qualifier | javax.inject | Java |
| @Autowired | org.springframework.bean.factory | Spring |

In order to explore the behavior of each annotation I fired up [Spring Tool Suite](http://www.springsource.com/developer/sts) and started debugging the code. I used Spring 3.0.5.RELEASE in my research. The following is a summary of my findings.

**The Code**

I wanted to know how ‘@Resource’, ‘@Autowired’, and ‘@Inject’ resolved dependencies. I created an interface called ‘Party’ and created two implementations classes. This allowed me to inject beans without using the concrete type. This provided the flexibility I needed to determine how Spring resolves beans when there are multiple type matches.

|  |
| --- |
| **public** **interface** Party {    } |

‘Person’ is a component and it implements ‘Party’.

|  |
| --- |
| **package** com.sourceallies.person;  ...  @Component  **public** **class** Person **implements** Party {    } |

‘Organization’ is a component and it implements ‘Party’.

|  |
| --- |
| **package** com.sourceallies.organization;  ...  @Component  **public** **class** Organization **implements** Party {    } |

I setup a Spring context that scans both of these packages for beans marked with ‘@Component’.

|  |
| --- |
| **<context:component-scan** base-package="com.sourceallies.organization"**/>**  **<context:component-scan** base-package="com.sourceallies.person"**/>** |

**Tests**

**Test 1: Ambiguous Beans**

In this test I injected a ‘Party’ bean that has multiple implementations in the Spring context.

|  |
| --- |
| @Resource  **private** Party party; |
| @Autowired  **private** Party party; |

|  |
| --- |
| @Inject  **private** Party party; |

***In all three cases a ‘NoSuchBeanDefinitionException’ is thrown. While this exception’s name implies that no beans were found, the message explains that two beans were found. All of these annotations result in the same exception.***

|  |
| --- |
| org.springframework.beans.factory.NoSuchBeanDefinitionException:  No unique bean of type [com.sourceallies.Party] is defined:  expected single matching bean but found 2: [organization, person] |

**Test 2: Field Name**

In this test I named the Party field person. By default beans marked with ‘@Component’ will have the same name as the class. Therefore the name of the class ‘Person’ is person.

|  |
| --- |
| @Resource  **private** Party person; |
| @Autowired  **private** Party person; |

|  |
| --- |
| @Inject  **private** Party person; |

‘@Resource’ can also take an optional ‘name’ attribute. This is equivalent to the ‘@Resource’ code above. In this case the field variable name remains ‘party’. There is no equivalent syntax for ‘@Autowired’ or ‘@Inject’. Instead you would have to use a ‘@Qualifier’. This syntax will be covered later.

|  |
| --- |
| @Resource(name="person")  **private** Party party; |

***All four of these styles inject the ‘Person’ bean.***

**Test 3: Field Type**

In this test I changed the type to be a ‘Person’.

|  |
| --- |
| @Resource  **private** Person party; |
| @Autowired  **private** Person party; |

|  |
| --- |
| @Inject  **private** Person party; |

***All of these annotations inject the ‘Person’ bean.***

**Test 4: Default Name Qualifier**

In this test I use a ‘@Qualifier’ annotation to point to the default name of the ‘Person’ component.

|  |
| --- |
| @Resource  @Qualifier("person")  **private** Party party; |
| @Autowired  @Qualifier("person")  **private** Party party; |

|  |
| --- |
| @Inject  @Qualifier("person")  **private** Party party; |

***All of these annotations inject the ‘Person’ bean.***

**Test 5: Qualified Name**

I added a ‘@Qualifier’ annotation to the ‘Person’ class

|  |
| --- |
| **package** com.sourceallies.person;  ...  @Component  @Qualifier("personBean")  **public** **class** Person **implements** Party {    } |

In this test I use a ‘@Qualifier’ annotation to point to the qualified name of the ‘Person’ component.

|  |
| --- |
| @Resource  @Qualifier("personBean")  **private** Party party; |
| @Autowired  @Qualifier("personBean")  **private** Party party; |

|  |
| --- |
| @Inject  @Qualifier("personBean")  **private** Party party; |

***All of these annotations inject the ‘Person’ bean.***

**Test 6: List of Beans**

In this test I inject a list of beans.

|  |
| --- |
| @Resource  **private** List<Party> parties; |
| @Autowired  **private** List<Party> parties; |

|  |
| --- |
| @Inject  **private** List<Party> parties; |

***All of these annotations inject 2 beans into the list. This can also be accomplished with a ‘@Qualifier’. Each bean marked with a specific qualifier will be added to the list.***

**Test 7: Conflicting messages**

In this test I add a bad ‘@Qualifier’ and a matching field name.

|  |
| --- |
| @Resource  @Qualifier("bad")  **private** Party person; |
| @Autowired  @Qualifier("bad")  **private** Party person; |

|  |
| --- |
| @Inject  @Qualifier("bad")  **private** Party person; |

***In this case the field marked with ‘@Resource’ uses the field name and ignores the ‘@Qualifier’. As a result the ‘Person’ bean is injected.***

***However the ‘@Autowired’ and ‘@Inject’ field throw a ‘NoSuchBeanDefinitionException’ error because it can not find a bean that matches the ‘@Qualifier’.***

|  |
| --- |
| org.springframework.beans.factory.NoSuchBeanDefinitionException:  No matching bean of type [com.sourceallies.Party] found **for** dependency:  expected at least 1 bean which qualifies as autowire candidate **for** **this** dependency.  Dependency annotations: {@org.springframework.beans.factory.annotation.Autowired(required=**true**),  @org.springframework.beans.factory.annotation.Qualifier(value=bad)} |

**Conclusions**

With the exception of test 2 & 7 the configuration and outcomes were identical. When I looked under the hood I determined that the ‘@Autowired’ and ‘@Inject’ annotation behave identically. Both of these annotations use the [‘AutowiredAnnotationBeanPostProcessor’](http://static.springsource.org/spring/docs/3.0.x/javadoc-api/org/springframework/beans/factory/annotation/AutowiredAnnotationBeanPostProcessor.html) to inject dependencies. ‘@Autowired’ and ‘@Inject’ can be used interchangeable to inject Spring beans. However the ‘@Resource’ annotation uses the [‘CommonAnnotationBeanPostProcessor’](http://static.springsource.org/spring/docs/3.1.0.M2/javadoc-api/org/springframework/context/annotation/CommonAnnotationBeanPostProcessor.html) to inject dependencies. Even though they use different post processor classes they all behave nearly identically. Below is a summary of their execution paths.

**@Autowired and @Inject**

1. Matches by Type
2. Restricts by Qualifiers
3. Matches by Name

**@Resource**

1. Matches by Name
2. Matches by Type
3. Restricts by Qualifiers (ignored if match is found by name)

While it could be argued that ‘@Resource’ will perform faster by name than ‘@Autowired’ and ‘@Inject’ it would be negligible. This isn’t a sufficient reason to favor one syntax over the others. I do however favor the ‘@Resource’ annotation for it’s concise notation style.

|  |
| --- |
| @Resource(name="person") |
| @Autowired  @Qualifier("person") |

|  |
| --- |
| @Inject  @Qualifier("person") |

You may argue that they can be equal concise if you use the field name to identify the bean name.

|  |
| --- |
| @Resource  **private** Party person; |
| @Autowired  **private** Party person; |

|  |
| --- |
| @Inject  **private** Party person; |

True enough, but what happens if you want to [refactor](http://en.wikipedia.org/wiki/Code_refactoring) your code? By simply renaming the field name you’re no longer referring to the same bean. I recommend the following practices when wiring beans with annotations.

**Spring Annotation Style Best Practices**

1. Explicitly name your component [@Component(“beanName”)]
2. Use ‘@Resource’ with the ‘name’ attribute [@Resource(name=”beanName”)]
3. Avoid ‘@Qualifier’ annotations unless you want to create a list of similar beans. For example you may want to mark a set of rules with a specific ‘@Qualifier’ annotation. This approach makes it simple to inject a group of rule classes into a list that can be used for processing data.
4. Scan specific packages for components [context:component-scan base-package=”com.sourceallies.person”]. While this will result in more component-scan configurations it reduces the chance that you’ll add unnecessary components to your Spring context.

Following these guidelines will increase the readability and stability of your Spring annotation configurations.