The Spring @Qualifier Annotation

Last modified: October 3, 2019

by baeldung (https://www.baeldung.com/author/baeldung/)

1. Overview

In this article, we'll explore **what the *@Qualifier* annotation can help us with**, which problems it solves, and how to use it.

We'll also explain how it's different from the *@Primary* annotation and from autowiring by name.

2. Autowire Need for Disambiguation

The *@Autowired* (https://www.baeldung.com/spring-autowire) annotation is a great way of making the need to inject a dependency in Spring explicit. And although it's useful, there are use cases for which this annotation alone isn't enough for Spring to understand which bean to inject.

By default, Spring resolves autowired entries by type.

If more than one bean of the same type is available in the container, the framework will throw *NoUniqueBeanDefinitionException*, indicating that more than one bean is available for autowiring.

Let's imagine a situation in which two possible candidates exist for Spring to inject as bean collaborators in a given instance:

```
@Component("fooFormatter")
public class FooFormatter implements Formatter {

    public String format() {
        return "foo";
    }
}

@Component("barFormatter")
public class BarFormatter implements Formatter {

    public String format() {
        return "bar";
    }
}

@Component
public class FooService {

    @Autowired
    private Formatter formatter;
}
```

If we try to load *FooService* into our context, the Spring framework will throw a *NoUniqueBeanDefinitionException*. This is because **Spring doesn't know which bean to inject**. To avoid this problem, there are several solutions. The *@Qualifier* annotation is one of them.

3. *@Qualifier* Annotation

By using the *@Qualifier* annotation, we can **eliminate the issue of which bean needs to be injected**.

Let's revisit our previous example and see how we solve the problem by including the *@Qualifier* annotation to indicate which bean we want to use:

```
public class FooService {

    @Autowired
    @Qualifier("fooFormatter")
    private Formatter formatter;
}
```

By including the *@Qualifier* annotation together with the name of the specific implementation we want to use – in this example, *Foo* – we can avoid ambiguity when Spring finds multiple beans of the same type.

We need to take into consideration that the qualifier name to be used is the one declared in the *@Component* annotation.

Note that we could've also used the *@Qualifier* annotation on the *Formatter* implementing classes, instead of specifying the names in their *@Component* annotations, to obtain the same effect:

```
@Component
@Qualifier("fooFormatter")
public class FooFormatter implements Formatter {
    //...
}

@Component
@Qualifier("barFormatter")
public class BarFormatter implements Formatter {
    //...
}
```

4. *@Qualifier* vs *@Primary*

There's another annotation called *@Primary* (https://www.baeldung.com/spring-primary) that we can use to decide which bean to inject when ambiguity is present regarding dependency injection.

This annotation **defines a preference when multiple beans of the same type are present**. The bean associated with the *@Primary* annotation will be used unless otherwise indicated.

Let's see an example:

```java
@Configuration
public class Config {

    @Bean
    public Employee johnEmployee() {
        return new Employee("John");
    }

    @Bean
    @Primary
    public Employee tonyEmployee() {
        return new Employee("Tony");
    }
}
```

In this example, both methods return the same *Employee* type. The bean that Spring will inject is the one returned by the method *tonyEmployee*. This is because it contains the *@Primary* annotation. This annotation is useful when we want to **specify which bean of a certain type should be injected by default**.

And in case we require the other bean at some injection point, we would need to specifically indicate it. We can do that via the *@Qualifier* annotation. For instance, we could specify that we want to use the bean returned by the *johnEmployee* method by using the *@Qualifier* annotation.

It's worth noting that **if both the *@Qualifier* and *@Primary* annotations are present, then the *@Qualifier* annotation will have precedence.** Basically, *@Primary* defines a default, while *@Qualifier* is very specific.

Let's see another way of using the *@Primary* annotation, this time using the initial example:

```java
@Component
@Primary
public class FooFormatter implements Formatter {
    //...
}

@Component
public class BarFormatter implements Formatter {
    //...
}
```

In this case, the *@Primary* annotation is placed in one of the implementing classes and will disambiguate the scenario.

5. *@Qualifier* vs Autowiring by Name

Another way to decide between multiple beans when autowiring is by using the name of the field to inject. **This is the default in case there are no other hints for Spring**. Let's see some code based on our initial example:

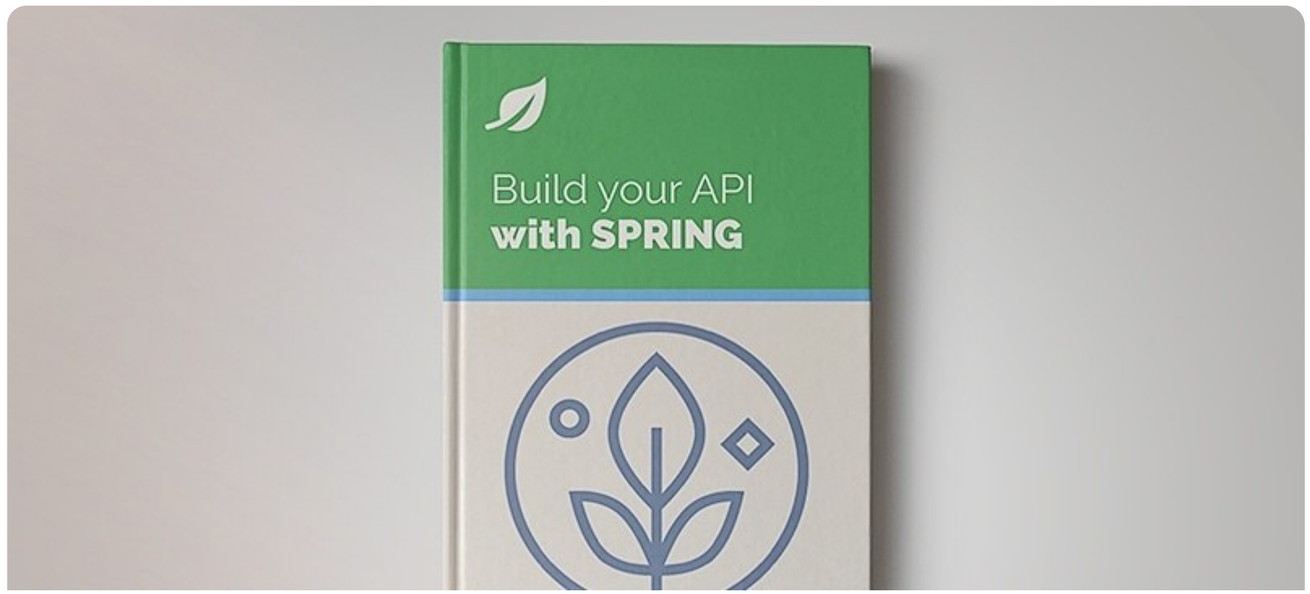
```
public class FooService {

    @Autowired
    private Formatter fooFormatter;
}
```

In this case, Spring will determine that the bean to inject is the *FooFormatter* one since the field name is matched to the value that we used in the *@Component* annotation for that bean.

6. Conclusion

We've described the scenarios in which we need to disambiguate which beans to inject. In particular, we described the *@Qualifier* annotation and compared it with other similar ways of determining which beans need to be used.

As usual, the complete code for this article is available over on GitHub (https://github.com/eugenp/tutorials/tree/master/spring-di).



Learning to build your API
with Spring?

Enter your email address

>> Get the eBook

Comments are closed on this article!

CATEGORIES

SPRING (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SPRING/)
REST (HTTPS://WWW.BAELDUNG.COM/CATEGORY/REST/)
JAVA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JAVA/)
SECURITY (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)
PERSISTENCE (HTTPS://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)
JACKSON (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JSON/JACKSON/)
HTTP CLIENT-SIDE (HTTPS://WWW.BAELDUNG.COM/CATEGORY/HTTP/)
KOTLIN (HTTPS://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

SERIES

JAVA "BACK TO BASICS" TUTORIAL (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
HTTPCLIENT 4 TUTORIAL (/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES)
SECURITY WITH SPRING (/SECURITY-SPRING)

ABOUT

ABOUT BAELDUNG (/ABOUT)
THE COURSES (HTTPS://COURSES.BAELDUNG.COM)
JOBS (/TAG/ACTIVE-JOB/)
META BAELDUNG (HTTP://META.BAELDUNG.COM/)
THE FULL ARCHIVE (/FULL_ARCHIVE)
WRITE FOR BAELDUNG (/CONTRIBUTION-GUIDELINES)
EDITORS (/EDITORS)
OUR PARTNERS (/PARTNERS)
ADVERTISE ON BAELDUNG (/ADVERTISE)

TERMS OF SERVICE (/TERMS-OF-SERVICE)
PRIVACY POLICY (/PRIVACY-POLICY)
COMPANY INFO (/BAELDUNG-COMPANY-INFO)
CONTACT (/CONTACT)