(/)

La anotación Spring @Qualifier

Última modificación: 15 de agosto de 2019

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

Primavera (https://www.baeldung.com/category/spring/) +

Anotaciones de primavera (https://www.baeldung.com/tag/spring-annotations/)

Spring Core Basics (https://www.baeldung.com/tag/spring-core-basics/)

Si tiene algunos años de experiencia en **Linux** y está interesado en compartirlo con la comunidad (y recibir un pago por su trabajo, por supuesto), eche un vistazo a la página "Escribir para nosotros" (/contribution-guidelines) . Aclamaciones. Eugen

1. Información general

En este artículo, exploraremos en **qué *nos* puede ayudar la anotación *@Qualifier*** , qué problemas resuelve y cómo usarla.

También explicaremos cómo es diferente de la anotación *@Primary* y del cableado automático por nombre.

2. Autowire necesidad de desambiguación

La anotación *@Autowired* (https://www.baeldung.com/spring-autowire) es una excelente manera de hacer explícita la necesidad de inyectar una dependencia en Spring. Y aunque es útil, hay casos de uso en los que esta anotación por sí sola no es suficiente para que Spring entienda qué bean inyectar.

De manera predeterminada, Spring resuelve las entradas automáticas por tipo.

Si hay más de un bean del mismo tipo disponible en el contenedor, el marco arrojará *NoUniqueBeanDefinitionException* , lo que indica que hay más de un bean disponible para el cableado automático.

Imaginemos una situación en la que existen dos posibles candidatos para que Spring los inyecte como colaboradores de bean en una instancia dada:

```
@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;
}
```

Si intentamos cargar *FooService* en nuestro contexto, el marco de Spring arrojará una *NoUniqueBeanDefinitionException* . Esto se debe a que **Spring no sabe qué bean inyectar** . Para evitar este problema, hay varias soluciones. La anotación *@Qualifier* es una de ellas.

3. Anotación *@Qualifier*

Al usar la anotación *@Qualifier* , podemos **eliminar el problema de qué bean necesita ser inyectado** .

Volvamos a nuestro ejemplo anterior y veamos cómo resolvemos el problema al incluir la anotación *@Qualifier* para indicar qué bean queremos usar:

```
public class FooService {

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

Al incluir la anotación *@Qualifier* junto con el nombre de la implementación específica que queremos usar, en este ejemplo, *Foo* , podemos evitar la ambigüedad cuando Spring encuentra múltiples beans del mismo tipo.

Debemos tener en cuenta que el nombre del calificador que se utilizará es el declarado en la anotación *@Component* .

Tenga en cuenta que también podríamos haber usado la anotación *@Qualifier* en las clases de implementación del *formateador* , en lugar de especificar los nombres en sus anotaciones *@Component* , para obtener el mismo efecto:

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

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

4. *@Qualifier* vs *@Primary*

Hay otra anotación llamada *@Primary* (https://www.baeldung.com/spring-primary) que podemos usar para decidir qué bean inyectar cuando hay ambigüedad con respecto a la inyección de dependencia.

Esta anotación **define una preferencia cuando hay múltiples beans del mismo tipo** . Se *utilizará* el bean asociado con la anotación *@Primary* a menos que se indique lo contrario.

Veamos un ejemplo:

```
@Configuration
public class Config {

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

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

En este ejemplo, ambos métodos devuelven el mismo tipo de *empleado* . El bean que inyectará Spring es el que devuelve el método *tonyEmployee* . Esto se debe a que contiene la anotación *@Primary* . Esta anotación es útil cuando queremos **especificar qué bean de cierto tipo debe inyectarse de forma predeterminada** .

Y en caso de que necesitemos el otro frijol en algún punto de inyección, deberíamos indicarlo específicamente. Podemos hacerlo a través de la anotación *@Qualifier* . Por ejemplo, podríamos especificar que queremos usar el bean devuelto por el método *johnEmployee* usando la anotación *@Qualifier* .

Vale la pena señalar que **si las anotaciones *@Qualifier* y *@Primary* están presentes, entonces la anotación *@Qualifier* tendrá prioridad.** Básicamente, *@Primary* define un valor predeterminado, mientras que *@Qualifier* es muy específico.

Veamos otra forma de usar la anotación *@Primary* , esta vez usando el ejemplo inicial:

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

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

En este caso, la anotación *@Primary* se coloca en una de las clases de implementación y desambigua el escenario.

5. *@Qualifier* vs Autowiring by Name

Otra forma de decidir entre múltiples beans cuando el cableado automático es mediante el nombre del campo para inyectar. **Este es el valor predeterminado en caso de que no haya otras sugerencias para Spring** . Veamos un código basado en nuestro ejemplo inicial:

```
public class FooService {

    @Autowired
    private Formatter fooFormatter;
}
```

En este caso, Spring determinará que el bean a inyectar es el *FooFormatter* ya que el nombre del campo coincide con el valor que usamos en la anotación *@Component* para ese bean.

6. Conclusión

Hemos descrito los escenarios en los que necesitamos desambiguar qué beans inyectar. En particular, describimos la anotación *@Qualifier* y la comparamos con otras formas similares de determinar qué beans deben usarse.

Como de costumbre, el código completo de este artículo está disponible en GitHub (https://github.com/eugenp/tutorials/tree/master/spring-di) .

¡Los comentarios están cerrados en este artículo!

CATEGORÍAS

PRIMAVERA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SPRING/)
DESCANSO (HTTPS://WWW.BAELDUNG.COM/CATEGORY/REST/)
JAVA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JAVA/)
SEGURIDAD (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)
PERSISTENCIA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)
JACKSON (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JSON/JACKSON/)
HTTP DEL LADO DEL CLIENTE (HTTPS://WWW.BAELDUNG.COM/CATEGORY/HTTP/)
KOTLIN (HTTPS://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

SERIE

TUTORIAL DE JAVA "VOLVER A LO BÁSICO" (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
HTTPCLIENT 4 TUTORIAL (/HTTPCLIENT-GUIDE)
RESTO CON SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
TUTORIAL SPRING PERSISTENCE (/PERSISTENCE-WITH-SPRING-SERIES)
SEGURIDAD CON PRIMAVERA (/SECURITY-SPRING)

ACERCA DE

SOBRE BAELDUNG (/ABOUT)
LOS CURSOS (HTTPS://COURSES.BAELDUNG.COM)
TRABAJO DE CONSULTORÍA (/CONSULTING)
META BAELDUNG (HTTP://META.BAELDUNG.COM/)
EL ARCHIVO COMPLETO (/FULL_ARCHIVE)
ESCRIBIR PARA BAELDUNG (/CONTRIBUTION-GUIDELINES)
EDITORES (/EDITORS)
NUESTROS COMPAÑEROS (/PARTNERS)
ANUNCIE EN BAELDUNG (/ADVERTISE)

TÉRMINOS DE SERVICIO (/TERMS-OF-SERVICE)
POLÍTICA DE PRIVACIDAD (/PRIVACY-POLICY)
INFORMACIÓN DE LA COMPAÑÍA (/BAELDUNG-COMPANY-INFO)
CONTACTO (/CONTACT)