**依赖注入及注解@Autowired和@Primary和@Qualifier的使用——SpringBoot**

文章目录

一，演示依赖注入

二，注解@Autowired

三，消除歧义性——@Primary和@Qualifier

一，演示依赖注入

Bean之间的依赖，在SpringIOC容器中，我们称为依赖注入。

列如：人类有时候会利用一些动物去完成一些事情，比如狗是看门，猫抓老鼠。于是做一些事情就依赖于那些可爱的动物。

为了更好展现此过程，我们做一个演示

第一步：定义人和动物接口，并写分别写两个实现类

接口：Person

package com.atstudying.demo.POJO;

public interface Person {

//使用动物服务

public void service();

//设置动物

public void setAnimal(Animal animal);

}

接口：Animal

package com.atstudying.demo.POJO;

public interface Animal {

public void use();

}

实现类：BussinessPerson.class

package com.atstudying.demo.POJO;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

@Component

public class BussinessPerson implements Person{

@Autowired

private Animal animal=null;

public BussinessPerson() {

super();

}

@Override

public void service(){

this.animal.use();

}

@Override

public void setAnimal(Animal animal){

this.animal=animal;

}

}

实现类：Dog.class

package com.atstudying.demo.POJO;

import org.springframework.stereotype.Component;

@Component

public class Dog implements Animal {

@Override

public void use(){

System.out.print("狗"+"["+Dog.class.getSimpleName()+"]"+"看门");

}

}

这里应注意注解 @Autowired ，这也是我们在Spring中最常用的注解之一，十分重要， 它会根据属性的类型(by type)找到对应的Bean进行注入。这里的Dog类是动物的一种，所以SpringIoC容器会把Dog的实例注入BussinessPerson中。这样通过Spring IoC容器获取BusinessPerson实例的时候就能够使用Dog实例来提供服务了

第二步：测试

DemoApplication.class

package com.atstudying.demo.Demo;

import com.atstudying.demo.POJO.BussinessPerson;

import com.atstudying.demo.POJO.Person;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

@SpringBootApplication

public class DemoApplication {

public static void main(String[] args) {

ApplicationContext applicationContext=new AnnotationConfigApplicationContext();

Person person=applicationContext.getBean(BussinessPerson.class);

person.service();

}

}

在上面我们都基于一个默认的情况，那就是不带参数的构造方法实现依赖注入。但事实上有些类只有带参数的构造方法，于是上述方法不可以在使用。

为了满足这个功能：我们使用 @Autowired 注解对构造方法的参数进行注入

例如，修改类BussinessPerson.class来满足此功能

package com.atstudying.demo.POJO;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

@Component

public class BussinessPerson implements Person{

@Autowired

private Animal animal=null;

public BussinessPerson(@Autowired @Qualifier("dog") Animal animal) {

super();

this.animal=animal;

}

@Override

public void service(){

this.animal.use();

}

@Override

public void setAnimal(Animal animal){

this.animal=animal;

}

}

解读：代码中取消了 @Autowired 对属性和方法的标注。注意构造方法的代码，在参数上加入了 @Autowired 和 @Qualifier注解，使得它能够注入进来。这里使用 @Qulifer 是为了避免歧义性（如果环境中只有猫或狗，则可以不使用 @Qulifer，而单单使用 @Autowired 即可）

二，注解@Autowired

问题一：如果上面的例子，我们又创建了一个猫类（猫可以抓老鼠），

cat.class

package com.atstudying.demo.POJO;

import org.springframework.stereotype.Component;

@Component

public class Dog implements Animal {

@Override

public void use(){

System.out.print("猫"+"["+Dog.class.getSimpleName()+"]"+"吃老鼠");

}

}

那么因为这个类只是定义了一个动物属性（Animal），而我们却有两个动物，猫和狗类，那么SpringIOC容器会如何注入呢 ？如果再继续进行测试会抛出如下异常。

Caused by: org . springfr amework. beans. factory . NoUni queBeanDe finitionException: No

qualifying bean of type ' com. springboot . chapter3.pojo. definition.Animal' available:expected

\*\*single matching bean but found 2: cat, dog\*\*

at org. springframework. beans. factory.config. DependencyDescriptor. resolveNotUnique(DependencyDescriptor. java:173)

at org. springframework. beans . factory.support. DefaultListableBeanFactory .doResolveDependency (DefaultListableBeanFactory.java:1116)

at org. springframework . beans. factory.support.DefaultListableBeanFactory.resolveDependency (Defaul tListableBeanFactory.java:1066)

at org. springframework . beans . factory. annotation .AutowiredAnnotationBeprocessorSAutowiredFieldE lement . inject (Autowiredannotatineaostrocessor java:585)anPost

onBe14 more

从加\* 的single matching bean but found 2: cat, dog的日志可以看出，Spring IoC容器并不能知道你需要注入什么动物(是狗?是猫?) 给BussinessPerson类对象，从而引起错误。那 @Autowired 能处理这个问题吗?

答案是肯定的。假设我们目前需要的是狗提供服务，那么可以把属性名称转化为dog，也就是原来的

@Autowired

private Animal animal=null;

修改为

@Autowired

private Animal Dog=null;

这里，我们只是将属性的名称从animal修改为了dog, 那么我们再测试的时候，你可以看到是采用狗来提供服务的。

那是因为@Autowired提供这样的规则，首先它会根据类型找到对应的Bean.如果对应类型的Bean不是唯一的， 那么它会根据其属性名称和Bean的名称进行匹配。如果匹配得上，就会使用该Bean;如果还无法匹配，就会抛出异常。

注意：@Autowired 是一个默认必须找到对应Bean的注解，如果不能确定其标注属性一定会存在并且允许这个被标注的属性为null,那么你可以配置@Autowired属性required为false.例如，像下面一样:

@Autowired(required = false)

同样，它除了可以标注属性外，还可以标注方法，如seAnimal方法，如下所示:

@Override

@Autowired

public void setAnimal (Animal animal){

this.animal = animal;

}

我们可以使用该方法从IOC容器中找到对应的动物进行注入，我们还可以使用在方法参数上，后面博客会在谈到

三，消除歧义性——@Primary和@Qualifier

当我们我们发现有猫有狗的时候，为了使@Autowired能够继续使用，我们做了一个决定，将BussinessPerson的属性名称从animal修改为dog。显然这是一个憋屈的做法，好好的一个动物，却被我们定义为了狗。

产生注入失败的问题根本是按类型(by type)查找，正如动物可以有多种类型，这样会造成Spring IoC容器注入的困扰，我们把这样的一一个问题称为歧义性。知道这个原因后，那么这两个注解是从哪个角度去解决这些问题的呢?

方法：

首先是一个注解 @Primary 它是个修改优先权的注解， 当我们有猫有狗的时候，假设这次需要使用猫，那么只需要在猫类的定义上加入\*\*@Primary就可以了\*\*，类似下面这样:

@Component

@Primary

public class Cat implements Animal ()

......

这里的 @Primary 的含义告诉Spring IoC容器，当发现有多个同样类型的Beam时，请优先使用被标注为了 @Primary.于是再进行测试时会发现，系统将用猫为你提供服务。因为当Spring进行注入的时候虽然它发现存在多个动物，但因为Cat被标注为了@Primary,所以优先采用Cat的实例进行了注入 ，

这样就通过优先级的变换使得IoC容器知道注入哪个具体的实例来满足依赖注入。

问题一：有时候 @Primary 也可以使用在多个类上，也许无论是猫还是狗都可能带上 @Primary 注解，其结果是IoC容器还是无法区分采用哪个Bean的实例进行注入，又或者说我们需要更加灵话的机制来实现注入

解决办法：@Qualifier 可以满足你的这个愿望。它的配置项value需要一个字符串去定义，它将与 @Autowired 组合在一起，通过类型和名称一起找到 Bean.我们知道Bean名称在Spring loc容器中是唯一的标识，通过这个就可以消除歧义性了。

下面假设猫已经标注 @Primary 而我们因此需要修改BussinessPerson属性animal的标注以适合我们的需要，如下所示:

@Autowired

@Qualifier ("dog")

private Animal animal

一旦这样声明，SpringIoC将会以类型和名称去寻找对应的Bean.显然也只能找到狗为我们服务了。

————————————————

版权声明：本文为CSDN博主「像一条恶狗」的原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/qq\_44891295/article/details/100864321