文档修订历史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版本** | **作者** | **工作描述** | **修订历史** | **修改日期** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

目录

[文档修订历史 1](#_Toc522896849)

[Spring5：@Autowired注解、@Resource注解和@Service注解 3](#_Toc522896850)

[1 什么是注解 3](#_Toc522896851)

[2 不使用注解 3](#_Toc522896852)

[3 @Autowired 5](#_Toc522896853)

[4 @Autowired接口注入 8](#_Toc522896854)

[5 @Resource 9](#_Toc522896855)

[6 @Service 10](#_Toc522896856)

[7 补充细节 11](#_Toc522896857)

Spring5：@Autowired注解、@Resource注解和@Service注解

# 什么是注解

传统的Spring做法是使用.xml文件来对bean进行注入或者是配置aop、事物，这么做有两个缺点：

1、如果所有的内容都配置在.xml文件中，那么.xml文件将会十分庞大；如果按需求分开.xml文件，那么.xml文件又会非常多。总之这将导致配置文件的可读性与可维护性变得很低

2、在开发中在.java文件和.xml文件之间不断切换，是一件麻烦的事，同时这种思维上的不连贯也会降低开发的效率

为了解决这两个问题，Spring引入了注解，通过"@XXX"的方式，让注解与Java Bean紧密结合，既大大减少了配置文件的体积，又增加了Java Bean的可读性与内聚性。

本篇文章，讲讲最重要的三个Spring注解，也就是@Autowired、@Resource和@Service，希望能通过有限的篇幅说清楚这三个注解的用法。

# 不使用注解

先看一个不使用注解的Spring示例，在这个示例的基础上，改成注解版本的，这样也能看出使用与不使用注解之间的区别，先定义一个老虎：

|  |
| --- |
| public class Tiger  {  private String tigerName = "TigerKing";    public String toString()  {  return "TigerName:" + tigerName;  }  } |

再定义一个猴子：

|  |
| --- |
| public class Monkey  {  private String monkeyName = "MonkeyKing";    public String toString()  {  return "MonkeyName:" + monkeyName;  }  } |

定义一个动物园：

|  |
| --- |
| public class Zoo  {  private Tiger tiger;  private Monkey monkey;    public void setTiger(Tiger tiger)  {  this.tiger = tiger;  }    public void setMonkey(Monkey monkey)  {  this.monkey = monkey;  }    public Tiger getTiger()  {  return tiger;  }    public Monkey getMonkey()  {  return monkey;  }    public String toString()  {  return tiger + "\n" + monkey;  }  } |

spring的配置文件这么写：

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns="http://www.springframework.org/schema/beans"  xmlns:context="http://www.springframework.org/schema/context"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans-4.2.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context-4.2.xsd"  default-autowire="byType">    <bean id="zoo" class="com.xrq.bean.Zoo" >  <property name="tiger" ref="tiger" />  <property name="monkey" ref="monkey" />  </bean>    <bean id="tiger" class="com.xrq.domain.Tiger" />  <bean id="monkey" class="com.xrq.domain.Monkey" />    </beans> |

都很熟悉，权当复习一遍了。

# @Autowired

@Autowired顾名思义，就是自动装配，其作用是为了消除代码Java代码里面的getter/setter与bean属性中的property。当然，getter看个人需求，如果私有属性需要对外提供的话，应当予以保留。

因此，引入@Autowired注解，先看一下spring配置文件怎么写：

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns="http://www.springframework.org/schema/beans"  xmlns:context="http://www.springframework.org/schema/context"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans-4.2.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context-4.2.xsd">    <context:component-scan base-package="com.xrq" />    <bean id="zoo" class="com.xrq.bean.Zoo" />  <bean id="tiger" class="com.xrq.domain.Tiger" />  <bean id="monkey" class="com.xrq.domain.Monkey" />    </beans> |

注意第10行，使用必须告诉spring一下我要使用注解了，告诉的方式有很多，<context:component-scan base-package="xxx" />是一种最简单的，spring会自动扫描xxx路径下的注解。

看到第12行，原来zoo里面应当注入两个属性tiger、monkey，现在不需要注入了。再看下，Zoo.java也很方便，把getter/setter都可以去掉：

|  |
| --- |
| public class Zoo  {  @Autowired  private Tiger tiger;    @Autowired  private Monkey monkey;    public String toString()  {  return tiger + "\n" + monkey;  }  } |

这里@Autowired注解的意思就是，当Spring发现@Autowired注解时，将自动在代码上下文中找到和其匹配（默认是类型匹配）的Bean，并自动注入到相应的地方去。

有一个细节性的问题是，假如bean里面有两个property，Zoo.java里面又去掉了属性的getter/setter并使用@Autowired注解标注这两个属性那会怎么样？答案是Spring会按照xml优先的原则去Zoo.java中寻找这两个属性的getter/setter，导致的结果就是初始化bean报错。

OK，假设此时我把.xml文件的13行、14行两行给去掉，再运行，会抛出异常：

|  |
| --- |
| Exception in thread "main" org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'Zoo': Injection of autowired dependencies failed; nested exception is org.springframework.beans.factory.BeanCreationException: Could not autowire field: private com.xrq.domain.Tiger com.xrq.bean.Zoo.ttiger; nested exception is org.springframework.beans.factory.NoSuchBeanDefinitionException: No qualifying bean of type [com.xrq.domain.Tiger] 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)}  at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor.postProcessPropertyValues(AutowiredAnnotationBeanPostProcessor.java:334)  at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.populateBean(AbstractAutowireCapableBeanFactory.java:1214)  at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(AbstractAutowireCapableBeanFactory.java:543)  at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean(AbstractAutowireCapableBeanFactory.java:482)  at org.springframework.beans.factory.support.AbstractBeanFactory$1.getObject(AbstractBeanFactory.java:305)  at org.springframework.beans.factory.support.DefaultSingletonBeanRegistry.getSingleton(DefaultSingletonBeanRegistry.java:230)  at org.springframework.beans.factory.support.AbstractBeanFactory.doGetBean(AbstractBeanFactory.java:301)  at org.springframework.beans.factory.support.AbstractBeanFactory.getBean(AbstractBeanFactory.java:196)  at org.springframework.beans.factory.support.DefaultListableBeanFactory.preInstantiateSingletons(DefaultListableBeanFactory.java:772)  at org.springframework.context.support.AbstractApplicationContext.finishBeanFactoryInitialization(AbstractApplicationContext.java:835)  at org.springframework.context.support.AbstractApplicationContext.refresh(AbstractApplicationContext.java:537)  at org.springframework.context.support.ClassPathXmlApplicationContext.<init>(ClassPathXmlApplicationContext.java:139)  at org.springframework.context.support.ClassPathXmlApplicationContext.<init>(ClassPathXmlApplicationContext.java:83)  at com.xrq.test.MyTest.main(MyTest.java:13)  Caused by: org.springframework.beans.factory.BeanCreationException: Could not autowire field: private com.xrq.domain.Tiger com.xrq.bean.Zoo.ttiger; nested exception is org.springframework.beans.factory.NoSuchBeanDefinitionException: No qualifying bean of type [com.xrq.domain.Tiger] found for dependency: expected at least 1 bean which qualifies as autowire candidate for this dependency. Dependency annotations: {@org.springframework.beans.factory.annotation.Autowired(required=true)}  at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor$AutowiredFieldElement.inject(AutowiredAnnotationBeanPostProcessor.java:571)  at org.springframework.beans.factory.annotation.InjectionMetadata.inject(InjectionMetadata.java:88)  at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor.postProcessPropertyValues(AutowiredAnnotationBeanPostProcessor.java:331)  ... 13 more  Caused by: org.springframework.beans.factory.NoSuchBeanDefinitionException: No qualifying bean of type [com.xrq.domain.Tiger] 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)}  at org.springframework.beans.factory.support.DefaultListableBeanFactory.raiseNoSuchBeanDefinitionException(DefaultListableBeanFactory.java:1373)  at org.springframework.beans.factory.support.DefaultListableBeanFactory.doResolveDependency(DefaultListableBeanFactory.java:1119)  at org.springframework.beans.factory.support.DefaultListableBeanFactory.resolveDependency(DefaultListableBeanFactory.java:1014)  at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor$AutowiredFieldElement.inject(AutowiredAnnotationBeanPostProcessor.java:543)  ... 15 more |

因为，@Autowired注解要去寻找的是一个Bean，Tiger和 Monkey的Bean定义都给去掉了，自然就不是一个Bean了，Spring容器找不到也很好理解。那么，如果属性找不到我不想让Spring容器抛 出异常，而就是显示null，可以吗？可以的，其实异常信息里面也给出了提示了，就是将@Autowired注解的required属性设置为false 即可：

此时，找不到tiger、monkey两个属性，Spring容器不再抛出异而是认为这两个属性为null。

# @Autowired接口注入

上面的比较简单，我们只是简单注入一个Java类，那么如果有一个接口，有多个实现，Bean里引用的是接口名，又该怎么做呢？比如有一个Car接口：

|  |
| --- |
| public interface Car  {  public String carName();  } |

两个实现类BMW和Benz：

|  |
| --- |
| @Service  public class BMW implements Car  {  public String carName()  {  return "BMW car";  }  } |

|  |
| --- |
| @Service  public class Benz implements Car  {  public String carName()  {  return "Benz car";  }  } |

写一个CarFactory，引用Car：

|  |
| --- |
| @Service  public class CarFactory  {  @Autowired  private Car car;    public String toString()  {  return car.carName();  }  } |

注意@Qualifier注解括号里面的应当是Car接口实现类的类名，我之前试的时候一直以为是bean的名字，所以写了"bMW"，结果一直报错。

# @Resource

把@Resource注解放在@Autowired下面说，是因为它们作用非常相似，这个就简单说了，例子过后点明一下@Resource和@Autowired的区别。先看一下@Resource，直接写Zoo.java了：

|  |
| --- |
| @Service  public class Zoo  {  @Resource(name = "tiger")  private Tiger tiger;    @Resource(type = Monkey.class)  private Monkey monkey;    public String toString()  {  return tiger + "\n" + monkey;  }  } |

这是详细一些的用法，说一下@Resource的装配顺序：

1、@Resource后面没有任何内容，默认通过name属性去匹配bean，找不到再按type去匹配

2、指定了name或者type则根据指定的类型去匹配bean

3、指定了name和type则根据指定的name和type去匹配bean，任何一个不匹配都将报错

**然后，区分一下@Autowired和@Resource两个注解的区别：**

1、@Autowired默认按照byType方式进行bean匹配，@Resource默认按照byName方式进行bean匹配

2、@Autowired是Spring的注解，@Resource是J2EE的注解，这个看一下导入注解的时候这两个注解的包名就一清二楚了

Spring属于第三方的，J2EE是Java自己的东西，因此，建议使用@Resource注解，以减少代码和Spring之间的耦合。

# @Service

上面这个例子，还可以继续简化，因为spring的配置文件里面还有12行~14行三个bean，下一步的简化是把这三个bean也给去掉，使得spring配置文件里面只有一个自动扫描的标签，增强Java代码的内聚性并进一步减少配置文件。

要继续简化，可以使用@Service。先看一下配置文件，当然是全部删除了：

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns="http://www.springframework.org/schema/beans"  xmlns:context="http://www.springframework.org/schema/context"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans-4.2.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context-4.2.xsd">    <context:component-scan base-package="com.xrq" />    </beans> |

是不是感觉很爽？起码我觉得是的。OK，下面以Zoo.java为例，其余的Monkey.java和Tiger.java都一样：

|  |
| --- |
| @Service  public class Zoo  {  @Autowired  private Tiger ttiger;    @Autowired  private Monkey mmonkey;    public String toString()  {  return ttiger + "\n" + mmonkey;  }  } |

这样，Zoo.java在Spring容器中存在的形式就是"zoo"，即可以通过ApplicationContext的getBean("zoo")方法来得到Zoo.java。@Service注解，其实做了两件事情：

1、声明Zoo.java是一个bean，这点很重要，因为Zoo.java是一个bean，其他的类才可以使用@Autowired将Zoo作为一个成员变量自动注入

2、Zoo.java在bean中的id是"zoo"，即类名且首字母小写

如果，我不想用这种形式怎么办，就想让Zoo.java在Spring容器中的名字叫做"Zoo"，可以的：

|  |
| --- |
| @Service  @Scope("prototype")  public class Zoo  {  @Autowired  private Monkey monkey;  @Autowired  private Tiger tiger;    public String toString()  {  return "MonkeyName:" + monkey + "\nTigerName:" + tiger;  }  } |

这样，就可以通过ApplicationContext的getBean("zoo")方法来得到Zoo.java了。

这里我还多加了一个@Scope注解，应该 很好理解。因为Spring默认产生的bean是单例的，假如我不想使用单例怎么办，xml文件里面可以在bean里面配置scope属性。注解也是一 样，配置@Scope即可，默认是"singleton"即单例，"prototype"表示原型即每次都会new一个新的出来。

# 补充细节

最后再补充一个我发现的细节。假如animal包下有Tiger、domain包下也有Tiger，它们二者都加了@Service注解，那么在Zoo.java中即使明确表示我要引用的是domain包下的Tiger，程序运行的时候依然会报错。

细想，其实这很好理解，两个Tiger都使 用@Service注解标注，意味着两个Bean的名字都是"tiger"，那么我在Zoo.java中自动装配的是哪个Tiger呢？不明确，因 此，Spring容器会抛出BeanDefinitionStoreException异常，Caused by：

|  |
| --- |
| Caused by: org.springframework.context.annotation.ConflictingBeanDefinitionException: Annotation-specified bean name 'monkey' for bean class [com.xrq.domain.Monkey] conflicts with existing, non-compatible bean definition of same name and class [com.xrq.animal.Monkey] |