[SegmentFault](https://segmentfault.com/)

* **[首页](https://segmentfault.com/)**
* **[问答](https://segmentfault.com/questions)**
* **[专栏](https://segmentfault.com/blogs)**
* **[课程](https://segmentfault.com/lives)**
* **[活动](https://segmentfault.com/events)**
* **[发现](https://segmentfault.com/a/1190000018077152" \l "##)**

窗体顶端

窗体底端

* [立即登录](https://segmentfault.com/user/login)
* [免费注册](https://segmentfault.com/user/register)

IMG_256

[](https://segmentfault.com/u/nongmayisheng)**[农码一生](https://segmentfault.com/u/nongmayisheng)**

* **16**

**2**

# **[Spring源码分析：@Autowired注解原理分析](https://segmentfault.com/a/1190000018077152)**

[IMG_258 java](https://segmentfault.com/t/java)

 阅读约 8 分钟

# **前言**

****关于@Autowired这个注解，我们再熟悉不过了，经常跟@Resource来做对比，这篇文章我们不讨论两者有何异同，仅分析@Autowired的原理（基于Spring5）。****

# **问题**

假如一个接口(IUserService)有两个实现类，分别是(UserServiceImpl01)和(UserServiceImpl02)，在我们给类注入的时候，这样写(@Autowired private IUserService userService)会发生什么情况？答案肯定是报错，那么原理呢？文字描述：因为首先@Autowired是按照类型注入的，也就是.class，但UserServiceImpl01和UserServiceImpl02都是IUserService类型的，于是Spring就会按照后面的名字(userService)在容器中查找，但发现根本没有这个名字，因为两个实现类在不指定名字情况下，就是首字母小写的类名，然后抛出异常：expected single matching bean but found 2。。。

# **如何解决这类问题**

1. 如果有两个实现类，还要使用@Autowired注解，可以将userService改成我们指定的实现类名称，比如UserServiceImpl01，或者不想改userService，可以加@Qualifier(value = "userServiceImpl01")，指定需要注入的实现类。
2. 使用@Resource注解，手动指定实现类名称。

****还有很多种方法，但基本思想都一样，无非就是如何区分两个同祖宗的儿子，既然根儿相同，那就只有指定名字了。****

# **@Autowired原理**

提到@Autowired我们一般都知道叫依赖注入

1. 什么是依赖注入？
2. 什么是注入，注到哪里？
3. 什么时候注入的？

## **什么是依赖注入？**

****依赖注入：Dependency Injection，简称DI，说白了就是利用反射机制为类的属性赋值的操作。****

## **什么是注入，注入到哪里？**

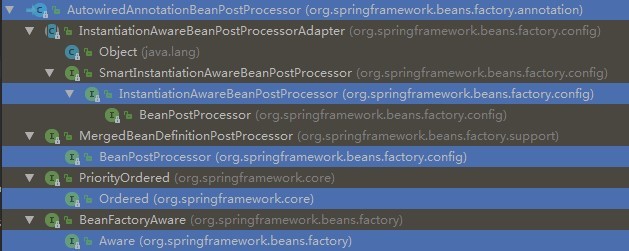
****注入就是为某个对象的外部资源赋值，注入某个对象所需要的外部资源（包括对象、资源、常量数据等）。IOC容器注入应用程序某个对象，应用程序所依赖的对象。****

## **什么时候注入的？**

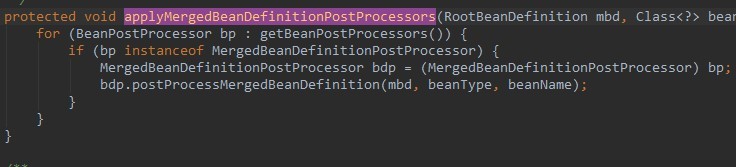
****在完成对象的创建，为对象变量进行赋值的时候进行注入（populate）。****

# **源码分析**

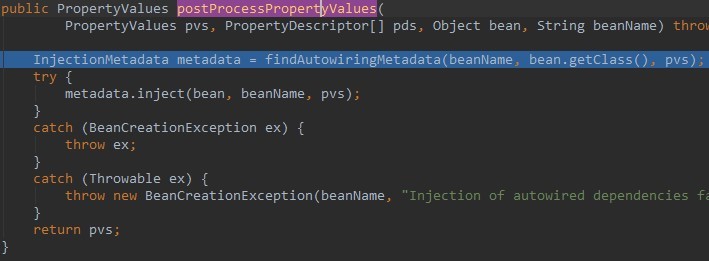
1. 首先点开@Autowired，注释上写Please consult the javadoc for the AutowiredAnnotationBeanPostProcessor，让我们去查阅这个类，看一下这个类的继承关系树，如下：

  
****可见它间接实现InstantiationAwareBeanPostProcessor，就具备了实例化前后(而不是初始化前后)管理对象的能力，实现了BeanPostProcessor，具有初始化前后管理对象的能力，实现BeanFactoryAware，具备随时拿到BeanFactory的能力，也就是说，这个AutowiredAnnotationBeanPostProcessor具备一切后置处理器的能力。****

1. 容器在初始化的时候，后置处理器的初始化要优先于剩下自定义Bean(比如我们自定义的Service，Controller等等)的初始化的，我们自定义的Bean初始化是在finishBeanFactoryInitialization(beanFactory)这里完成的，来到AbstractApplicationContext的refresh()方法。
2. finishBeanFactoryInitialization(beanFactory)-->beanFactory.preInstantiateSingletons()-->getBean(beanName)-->doGetBean(beanName)-->来到AbstractBeanFactory第317行createBean(beanName, mbd, args)，来创建bean实例-->来到AbstractAutowireCapableBeanFactory第503行doCreateBean(beanName, mbdToUse, args)-->紧接着来到AbstractAutowireCapableBeanFactory的第543行，instanceWrapper = createBeanInstance(beanName, mbd, args)就已经把Bean实例创建出来了，只不过instanceWrapper是一个被包装过了的bean，它里面的属性还未赋实际值-->然后来到第555行applyMergedBeanDefinitionPostProcessors(mbd, beanType, beanName)，这一步的作用就是将所有的后置处理器拿出来，并且把名字叫beanName的类中的变量都封装到InjectionMetadata的injectedElements集合里面，目的是以后从中获取，挨个创建实例，通过反射注入到相应类中。



1. 紧接着来到AbstractAutowireCapableBeanFactory第588行populateBean(beanName, mbd, instanceWrapper)-->点进去，来到AbstractAutowireCapableBeanFactory的第1347行，来循环遍历所有的后置处理器for (BeanPostProcessor bp : getBeanPostProcessors())，从方法名字postProcessPropertyValues也能看出来，就是给属性赋值，当bp是AutowiredAnnotationBeanPostProcessor的时候，进入postProcessPropertyValues方法，来到AutowiredAnnotationBeanPostProcessor的postProcessPropertyValues方法，如下：

  
首先找到需要注入的哪些元数据，然后metadata.inject（注入），注入方法点进去，来到InjectionMetadata的inject方法，在一个for循环里面依次执行element.inject(target, beanName, pvs)，来对属性进行注入。

1. 进入element.inject(target, beanName, pvs)，注意，这里必须要debug才可以进入真正的方法。来到AutowiredAnnotationBeanPostProcessor的inject方法，

第584行，value = beanFactory.resolveDependency(desc, beanName, autowiredBeanNames, typeConverter)，由工厂解析这个依赖，进入，来到DefaultListableBeanFactory第1065行，result = doResolveDependency(descriptor, requestingBeanName, autowiredBeanNames, typeConverter)再次解析依赖，点击进入，来到DefaultListableBeanFactory的doResolveDependency()方法，前面是一堆判断，比较，查看属性类型，这种类型的有几个(matchingBeans)，如果只有一个匹配，那么来到第1138行，instanceCandidate = descriptor.resolveCandidate(autowiredBeanName, type, this)，进入这个方法，可以看到就是前面说的根据工厂来创建实例的过程了：beanFactory.getBean(beanName)，其中这个beanName就是属性的名称，当经过一系列操作完成属性的实例化后，便来到AutowiredAnnotationBeanPostProcessor的第611行，利用反射为此对象赋值。这样，对象的创建以及赋值就完成了。

# **总结**

****在容器启动，为对象赋值的时候，遇到@Autowired注解，会用后置处理器机制，来创建属性的实例，然后再利用反射机制，将实例化好的属性，赋值给对象上，这就是Autowired的原理。****

阅读 5k发布于 2019-01-31

 赞 2

 收藏 2 赞赏

 分享

本作品系 原创 ， [采用《署名-非商业性使用-禁止演绎 4.0 国际》许可协议](https://creativecommons.org/licenses/by-nc-nd/4.0/" \t "/Users/liu123/Documents\\x/_blank)

[](https://segmentfault.com/u/nongmayisheng)

##### **[农码一生](https://segmentfault.com/u/nongmayisheng)**

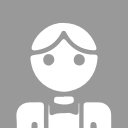
* **16**

关注作者

IMG_263

**1 条评论**

[得票时间](javascript:;)



窗体顶端

提交评论

窗体底端



**[王晓安](https://segmentfault.com/u/wangxiaoan" \t "/Users/liu123/Documents\\x/_blank)** ：

~(≧▽≦)/~赞

[回复](javascript:;) 2019-07-25

##### **推荐阅读**

##### **[Spring Core Container 源码分析七：注册 Bean Definitions](http://segmentfault.com/a/1190000013276001" \t "/Users/liu123/Documents\\x/_blank)**

[原本以为，Spring通过解析bean的配置，生成并注册beandefintions的过程不太复杂，比较简单，不用单独开辟一篇博文来讲述；但是当在分析前面两个章节有关@Autowired、@Component、@Service注解的注入机制的时候，发现，如果没有对有关beandefintions的解析和注册机制彻底弄明白，则很难弄清楚annotation在Spring容器中的底层运行机制；所以，本](http://segmentfault.com/a/1190000013276001" \t "/Users/liu123/Documents\\x/_blank)

[伤神 阅读 243](http://segmentfault.com/a/1190000013276001" \t "/Users/liu123/Documents\\x/_blank)

##### **[Spring Core Container 源码分析三：Spring Beans 初始化流程分析](http://segmentfault.com/a/1190000013212015" \t "/Users/liu123/Documents\\x/_blank)**

[本文是笔者所著的SpringCoreContainer源码分析系列之一；本篇文章主要试图梳理出SpringBeans的初始化主流程和相关核心代码逻辑；本文转载自本人的私人博客，伤神的博客：http:www.shangyang.me2017...本文为作者的原创作品，转载需注明出处；参考SpringCoreContainer源码分析二：环境准备依然使用这个官网上的用例，来进行调试；Person.ja](http://segmentfault.com/a/1190000013212015" \t "/Users/liu123/Documents\\x/_blank)

[伤神 阅读 9](http://segmentfault.com/a/1190000013212015" \t "/Users/liu123/Documents\\x/_blank)

##### **[Spring BeanUtils源码分析](http://segmentfault.com/a/1190000014833730" \t "/Users/liu123/Documents\\x/_blank)**

[在我们着手一个JavaWeb项目的时候，经常会遇到DO、VO、DTO对象之间的属性拷贝，若采用get、set的方法来进行赋值的话，代码会相当冗长丑陋，一般我们会采用Spring的BeanUtils类来进行属性拷贝，其基本原理就是通过Java的反射机制，下面我们来看一下源码的具体实现。在分析源码前，我们先温习一下以下的知识点。在Java中万物皆对象，而且我们在代码中写的每一个类也都是对象，是java](http://segmentfault.com/a/1190000014833730" \t "/Users/liu123/Documents\\x/_blank)

[扑火的蛾 阅读 29](http://segmentfault.com/a/1190000014833730" \t "/Users/liu123/Documents\\x/_blank)

##### **[Retrofit源码分析三 源码分析](http://segmentfault.com/a/1190000014911412" \t "/Users/liu123/Documents\\x/_blank)**

[我们先来看一下Retrofit的常见使用方法：上面是Retrofit的最基本使用方法，当然现在使用最多的还是RxJava2+Retrofit搭配使用，关于RxJava2，大家可以看我的另一篇RxJava2源码分析，当然RxJava2与Retrofit搭配使用的解析我会在稍后分析，这里我们先关注最基本的使用方法。这一步的目的就是封装我们网络请求相关的一些参数，没什么好多说的。Retrofit实例对象](http://segmentfault.com/a/1190000014911412" \t "/Users/liu123/Documents\\x/_blank)

[BlackFlagBin 阅读 9](http://segmentfault.com/a/1190000014911412" \t "/Users/liu123/Documents\\x/_blank)

##### **[spring boot源码分析之springmvc(上)](http://segmentfault.com/a/1190000011085572" \t "/Users/liu123/Documents\\x/_blank)**

[相对于springboot我们使用springmvc更加频繁和熟悉。sprintboot具有springmvc的功能，两者之间到底存在着什么样的关系，通过下面的篇幅我们一起来分析。我们再回顾下springmvc的配置，首先看下web.xml:可以看到该文件中配置了context-param、listener以及servlet信息(执行顺序：context-param->listener-&g](http://segmentfault.com/a/1190000011085572" \t "/Users/liu123/Documents\\x/_blank)

[帅帅的波 阅读 17](http://segmentfault.com/a/1190000011085572" \t "/Users/liu123/Documents\\x/_blank)

##### **[Spring专题之IOC源码分析](http://segmentfault.com/a/1190000016261917" \t "/Users/liu123/Documents\\x/_blank)**

[以下源码基于Spring5.0.2版本解析。容器，顾名思义可以用来容纳一切事物。我们平常所说的SpringIOC容器就是一个可以容纳对象的东西。IOC全名InversionofControl，即控制反转，什么是控制反转？平时我们代码里需要创建一个对象是需要通过new操作或者反射等方式创建，也就是说现在是我们人为地创建对象，控制对象，那么控制反转的意思就显而易见了，就是将原来属于我们的控制权交由Sp](http://segmentfault.com/a/1190000016261917" \t "/Users/liu123/Documents\\x/_blank)

[DearBelinda 阅读 772](http://segmentfault.com/a/1190000016261917" \t "/Users/liu123/Documents\\x/_blank)

##### **[窥探React - 源码分析](http://segmentfault.com/a/1190000013267175" \t "/Users/liu123/Documents\\x/_blank)**

[所谓知其然还要知其所以然.本文将分析React15-stable的部分源码,包括组件初始渲染的过程和组件更新的过程.在这之前,假设读者已经:代码架构预览首先,我们找到React在Github上的地址,把15-stable版本的源码copy下来,观察它的整体架构,这里首先阅读关于源码介绍的官方文档,再接着看.我们要分析的源码在src目录下:分析方法1、首先看一些网上分析的文章,对重点部分的源码有个印](http://segmentfault.com/a/1190000013267175" \t "/Users/liu123/Documents\\x/_blank)

[莫凡 阅读 16](http://segmentfault.com/a/1190000013267175" \t "/Users/liu123/Documents\\x/_blank)

##### **[Guava 源码分析（Cache 原理）](http://segmentfault.com/a/1190000015272229" \t "/Users/liu123/Documents\\x/_blank)**

[Google出的Guava是Java核心增强的库，应用非常广泛。我平时用的也挺频繁，这次就借助日常使用的Cache组件来看看Google大牛们是如何设计的。缓存在日常开发中举足轻重，如果你的应用对某类数据有着较高的读取频次，并且改动较小时那就非常适合利用缓存来提高性能。缓存之所以可以提高性能是因为它的读取效率很高，就像是CPU的L1、L2、L3缓存一样，级别越高相应的读取速度也会越快。但也不是什么](http://segmentfault.com/a/1190000015272229" \t "/Users/liu123/Documents\\x/_blank)

[crossoverJie 阅读 112](http://segmentfault.com/a/1190000015272229" \t "/Users/liu123/Documents\\x/_blank)

##### **[Spring源码窥探](https://segmentfault.com/blog/springsource_explore)**

**用户专栏**

这可能是除了书籍外总结最详细的Spring源码分析了。

****3**** 人关注

****4**** 篇文章

关注专栏[专栏主页](https://segmentfault.com/blog/springsource_explore)

IMG_266



[广告位促销，月曝光三千万，10 元/天](https://segmentfault.com/sponsor)

**[迎接Vue 3.0](https://sponsor.segmentfault.com/ck.php?oaparams=2__bannerid=217__zoneid=7__cb=965453dc90__oadest=https://segmentfault.com/ls/1650000021531837" \t "/Users/liu123/Documents\\x/_blank)** [Vue 3.0的新变化,我们应该怎么准备](https://sponsor.segmentfault.com/ck.php?oaparams=2__bannerid=217__zoneid=7__cb=965453dc90__oadest=https://segmentfault.com/ls/1650000021531837" \t "/Users/liu123/Documents\\x/_blank)

IMG_268

**[网络安全之 Kali 渗透入门实战](https://sponsor.segmentfault.com/ck.php?oaparams=2__bannerid=218__zoneid=9__cb=093e2747c9__oadest=https://segmentfault.com/ls/1650000021447494" \t "/Users/liu123/Documents\\x/_blank)** [kali一个包含主流安全工具的一个集合系统](https://sponsor.segmentfault.com/ck.php?oaparams=2__bannerid=218__zoneid=9__cb=093e2747c9__oadest=https://segmentfault.com/ls/1650000021447494" \t "/Users/liu123/Documents\\x/_blank)

IMG_269

IMG_270

IMG_271

IMG_272

###### **目录**

[前言](https://segmentfault.com/a/1190000018077152" \l "item-1)[问题](https://segmentfault.com/a/1190000018077152" \l "item-2)[如何解决这类问题](https://segmentfault.com/a/1190000018077152" \l "item-3)[@Autowired原理](https://segmentfault.com/a/1190000018077152" \l "item-4)[什么是依赖注入？](https://segmentfault.com/a/1190000018077152" \l "item-4-1)[什么是注入，注入到哪里？](https://segmentfault.com/a/1190000018077152" \l "item-4-2)[什么时候注入的？](https://segmentfault.com/a/1190000018077152" \l "item-4-3)**[源码分析](https://segmentfault.com/a/1190000018077152" \l "item-5)**[总结](https://segmentfault.com/a/1190000018077152" \l "item-6)

IMG_273

###### **产品**

[热门问答](https://segmentfault.com/questions/hottest?utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

[热门专栏](https://segmentfault.com/blogs/hottest?utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

[热门课程](https://segmentfault.com/lives?utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

[最新活动](https://segmentfault.com/events?utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

[技术圈](https://segmentfault.com/groups?utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

[酷工作](https://segmentfault.com/groups?tab=jobs&utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

[移动客户端](https://segmentfault.com/app?utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

###### **课程**

[Java 开发课程](https://segmentfault.com/lives/edu?tag=java&sort=hottest&utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

[PHP 开发课程](https://segmentfault.com/lives/edu?tag=php&sort=hottest&utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

[Python 开发课程](https://segmentfault.com/lives/edu?tag=python&sort=hottest&utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

[前端开发课程](https://segmentfault.com/lives/edu?category=1&sort=hottest&utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

[移动开发课程](https://segmentfault.com/lives/study?category=3&sort=hottest&utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

###### **资源**

[每周精选](https://segmentfault.com/weekly?utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

[用户排行榜](https://segmentfault.com/users?utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

[徽章](https://segmentfault.com/badges?utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

[帮助中心](https://segmentfault.com/faq?utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

[声望与权限](https://segmentfault.com/repu?utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

[社区服务中心](https://segmentfault.com/community?utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

###### **合作**

[关于我们](https://about.segmentfault.com/?utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

[广告投放](https://business.segmentfault.com/ads?utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

[职位发布](https://segmentfault.com/groups?tab=jobs&utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

[讲师招募](https://jinshuju.net/f/HK5r9K?utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

[联系我们](https://about.segmentfault.com/contact.html?utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

[合作伙伴](https://segmentfault.com/link?utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

###### **关注**

[产品技术日志](https://segmentfault.com/blog/segmentfault?utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

[社区运营日志](https://segmentfault.com/blog/community_admin?utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

[市场运营日志](https://segmentfault.com/blog/segmentfault_news?utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

[团队日志](https://segmentfault.com/blog/segmentfault_team?utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

[社区访谈](https://segmentfault.com/blog/interview?utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

###### **条款**

[服务条款](https://segmentfault.com/tos?utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

[隐私政策](https://segmentfault.com/privacy?utm_source=sf-footer" \t "/Users/liu123/Documents\\x/_blank)

[  
下载 App](https://segmentfault.com/app?utm_source=sf-footer)

Copyright © 2011-2020 SegmentFault. 当前呈现版本 19.02.27

[浙ICP备 15005796号-2](http://beian.miit.gov.cn/" \t "/Users/liu123/Documents\\x/_blank) [浙公网安备 33010602002000号](http://www.beian.gov.cn/portal/registerSystemInfo?recordcode=33010602002000" \t "/Users/liu123/Documents\\x/_blank) 杭州堆栈科技有限公司版权所有

CDN 存储服务由 [又拍云](https://www.upyun.com/?utm_source=segmentfault&utm_medium=link&utm_campaign=upyun&md=segmentfault" \t "/Users/liu123/Documents\\x/_blank) 赞助提供