<https://mp.weixin.qq.com/s/9_lUOU2tgVUf5VMZImfWJA>

# 面试被问了几百遍的 IoC 和 AOP ，还在傻傻搞不清楚？

这篇文章会从下面从以下几个问题展开对 IoC & AOP 的解释

* 什么是 IoC？
* IoC 解决了什么问题？
* IoC 和 DI 的区别？
* 什么是 AOP？
* AOP 解决了什么问题？
* AOP 为什么叫做切面编程？

*首先声明：IoC & AOP 不是 Spring 提出来的，它们在 Spring 之前其实已经存在了，只不过当时更加偏向于理论。Spring 在技术层次将这两个思想进行了很好的实现。*

### **什么是 IoC**

IoC （Inversion of control ）控制反转/反转控制。它是一种思想不是一个技术实现。描述的是：Java 开发领域对象的创建以及管理的问题。

例如：现有类 A 依赖于类 B

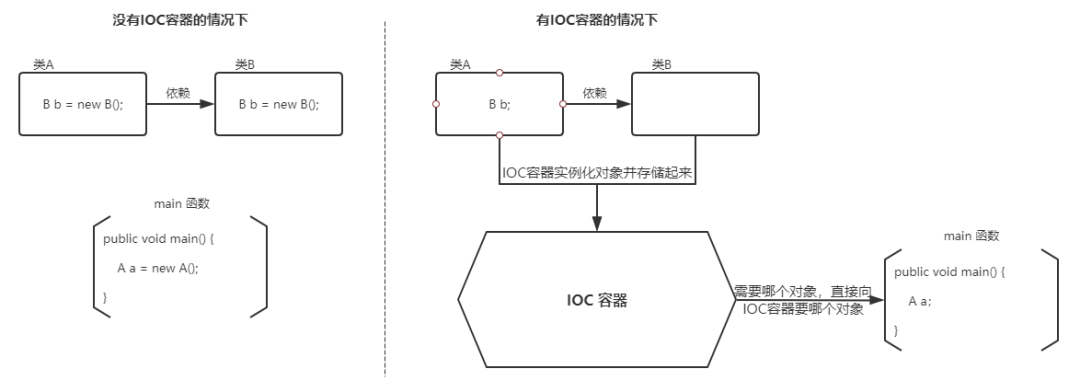
* **传统的开发方式** ：往往是在类 A 中手动通过 new 关键字来 new 一个 B 的对象出来
* **使用 IoC 思想的开发方式** ：不通过 new 关键字来创建对象，而是通过 IoC 容器(Spring 框架) 来帮助我们实例化对象。我们需要哪个对象，直接从 IoC 容器里面过去即可。

从以上两种开发方式的对比来看：我们 “丧失了一个权力” (创建、管理对象的权力)，从而也得到了一个好处（不用再考虑对象的创建、管理等一系列的事情）。

#### **为什么叫控制反转**

**控制** ：指的是对象创建（实例化、管理）的权力

**反转** ：控制权交给外部环境（Spring 框架、IoC 容器）



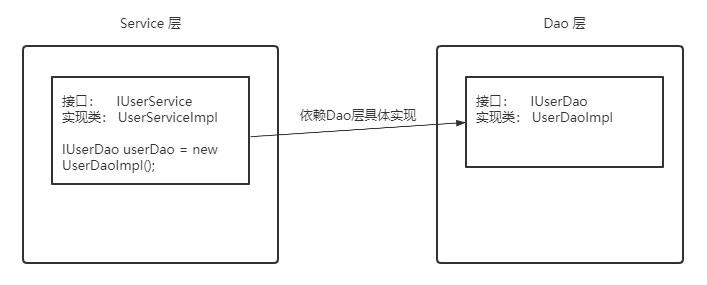
### **IoC 解决了什么问题**

IoC 的思想就是两方之间不互相依赖，由第三方容器来管理相关资源。这样有什么好处呢？

1. 对象之间的耦合度或者说依赖程度降低；
2. 资源变的容易管理；比如你用 Spring 容器提供的话很容易就可以实现一个单例。

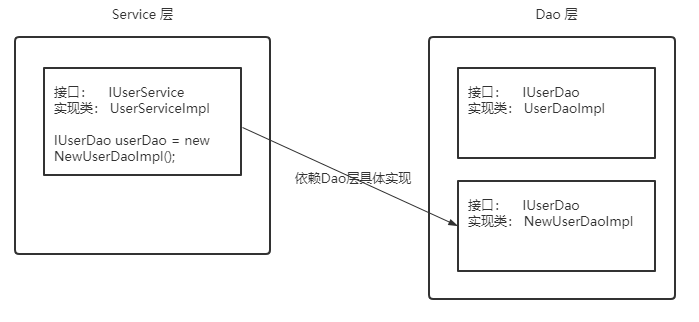
例如：现有一个针对 User 的操作，利用 Service 和 Dao 两层结构进行开发

在没有使用 IoC 思想的情况下，Service 层想要使用 Dao 层的具体实现的话，需要通过 new 关键字在UserServiceImpl 中手动 new 出 IUserDao 的具体实现类 UserDaoImpl（不能直接 new 接口类）。

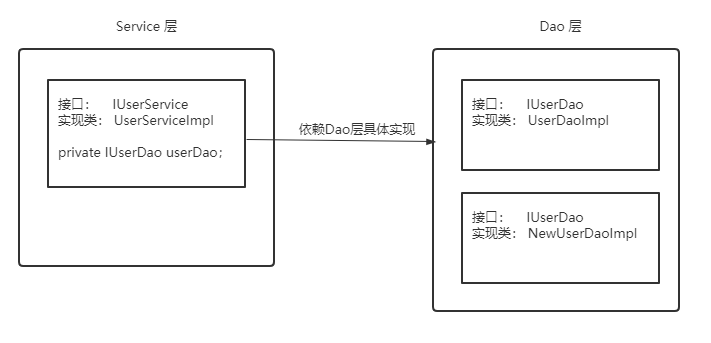


很完美，这种方式也是可以实现的，但是我们想象一下如下场景：

开发过程中突然接到一个新的需求，针对对IUserDao 接口开发出另一个具体实现类。因为 Server 层依赖了IUserDao的具体实现，所以我们需要修改UserServiceImpl中 new 的对象。如果只有一个类引用了IUserDao的具体实现，可能觉得还好，修改起来也不是很费力气，但是如果有许许多多的地方都引用了IUserDao的具体实现的话，一旦需要更换IUserDao 的实现方式，那修改起来将会非常的头疼。



使用 IoC 的思想，我们将对象的控制权（创建、管理）交有 IoC 容器去管理，我们在使用的时候直接向 IoC 容器 “要” 就可以了。

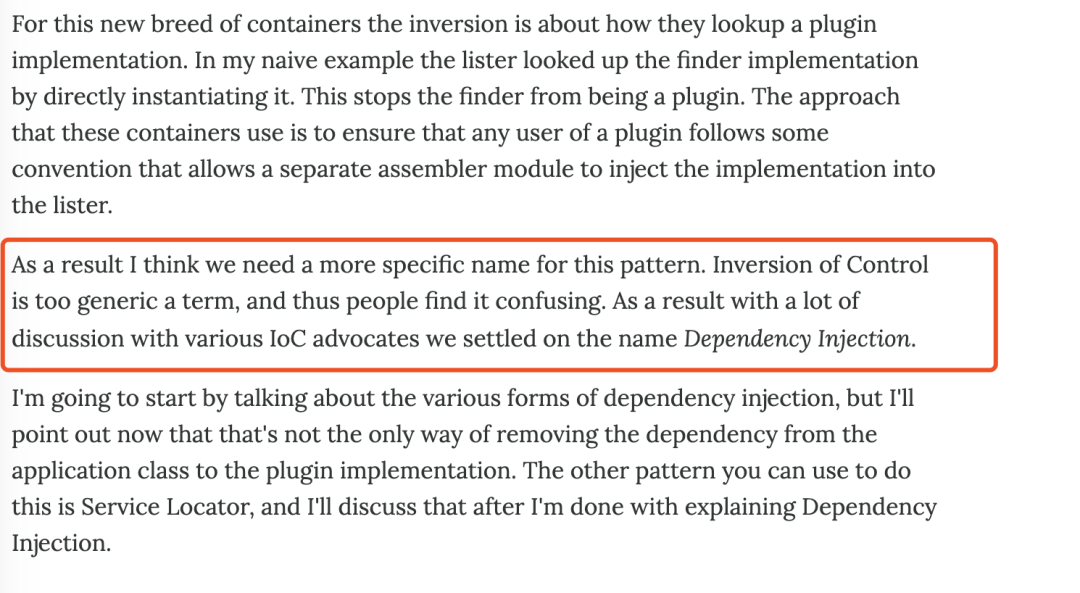


### **IoC 和 DI 别再傻傻分不清楚**

IoC（Inverse of Control:控制反转）是一种**设计思想** 或者说是某种模式。这个设计思想就是 **将原本在程序中手动创建对象的控制权，交由 Spring 框架来管理。** IoC 在其他语言中也有应用，并非 Spring 特有。**IoC 容器是 Spring 用来实现 IoC 的载体， IoC 容器实际上就是个 Map（key，value）,Map 中存放的是各种对象。**

IoC 最常见以及最合理的实现方式叫做依赖注入（Dependency Injection，简称 DI）。

并且，老马（Martin Fowler）在一篇文章中提到将 IoC 改名为 DI，原文如下，原文地址：https://martinfowler.com/articles/injection.html 。



老马的大概意思是 IoC 太普遍并且不表意，很多人会因此而迷惑，所以，使用 DI 来精确指名这个模式比较好。

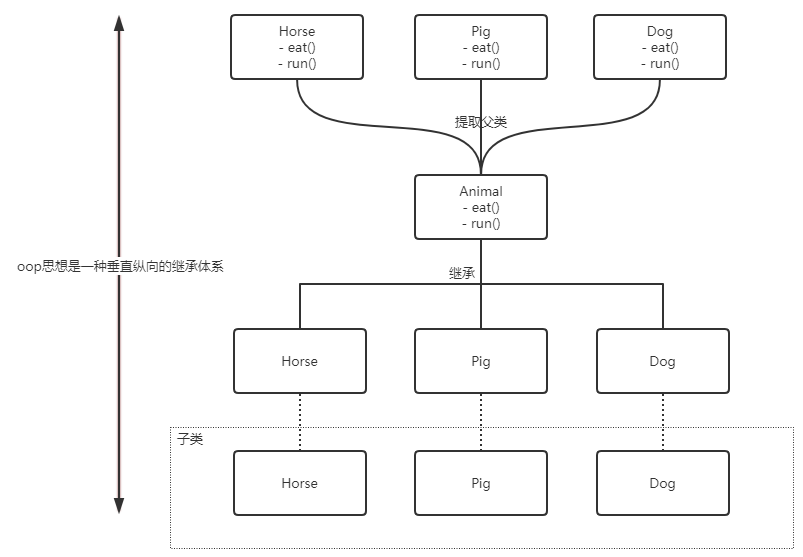
### **什么是 AOP**

AOP：Aspect oriented programming 面向切面编程，AOP 是 OOP（面向对象编程）的一种延续。

下面我们先看一个 OOP 的例子。

例如：现有三个类，Horse、Pig、Dog，这三个类中都有 eat 和 run 两个方法。

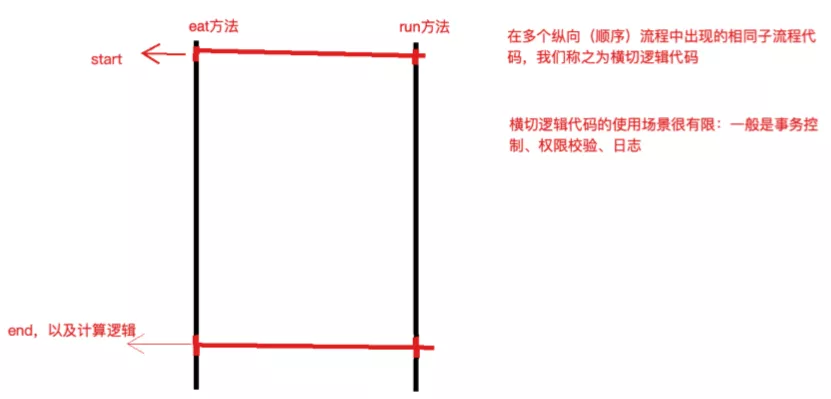
通过 OOP 思想中的继承，我们可以提取出一个 Animal 的父类，然后将 eat 和 run 方法放入父类中，Horse、Pig、Dog通过继承Animal类即可自动获得 eat() 和 run() 方法。这样将会少些很多重复的代码。



OOP 编程思想可以解决大部分的代码重复问题。但是有一些问题是处理不了的。比如在父类 Animal 中的多个方法的相同位置出现了重复的代码，OOP 就解决不了。



这部分重复的代码，一般统称为 **横切逻辑代码**。

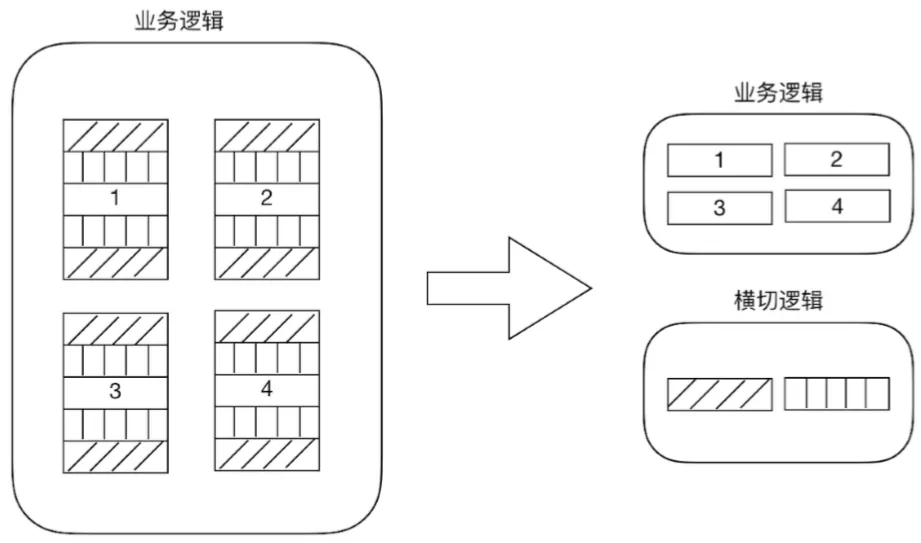


横切逻辑代码存在的问题：

* 代码重复问题
* 横切逻辑代码和业务代码混杂在一起，代码臃肿，不变维护

**AOP 就是用来解决这些问题的**

AOP 另辟蹊径，提出横向抽取机制，将横切逻辑代码和业务逻辑代码分离



代码拆分比较容易，难的是如何在不改变原有业务逻辑的情况下，悄无声息的将横向逻辑代码应用到原有的业务逻辑中，达到和原来一样的效果。

### **AOP 解决了什么问题**

通过上面的分析可以发现，AOP 主要用来解决：在不改变原有业务逻辑的情况下，增强横切逻辑代码，根本上解耦合，避免横切逻辑代码重复。

### **AOP 为什么叫面向切面编程**

**切** ：指的是横切逻辑，原有业务逻辑代码不动，只能操作横切逻辑代码，所以面向横切逻辑

**面** ：横切逻辑代码往往要影响的是很多个方法，每个方法如同一个点，多个点构成一个面。这里有一个面的概念