


Spring 两大核心：IOC 和 AOP

IOC (控制反转)：对象创建和对象之间的调用交给 Spring 管理
通过 XML 可创建和调用对象

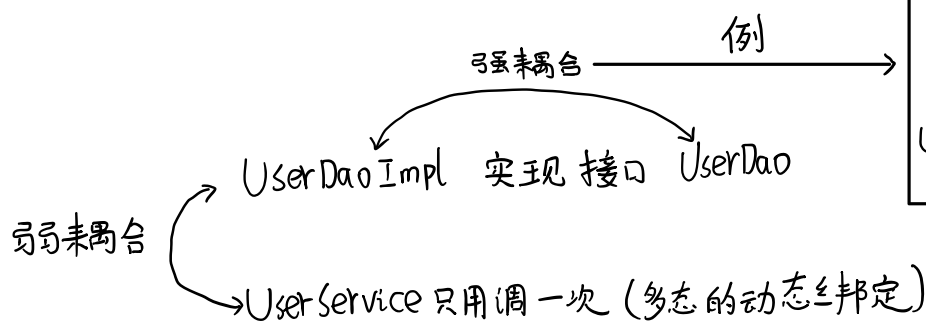
目的：解耦 (降低类和方法之间的依赖)

1) 传统直接调用对象

需要 UserDao1, UserDao2 类

UserService 类需多次调用  强耦合

2) 接口解耦 (面向接口编程)

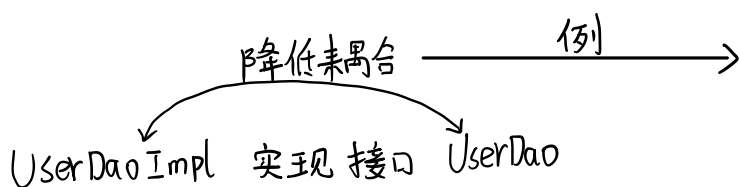


```
UserDao UserDao = new UserDaoImpl()  
UserDao UserDao = new UserDaoImpl2()
```

需要改

3) 工厂模式解耦

中间人



```
UserDao UserDao = new UserDaoImpl()  
BeanFactory.getBean(UserDao1())  
  
UserDao UserDao = new UserDaoImpl2()  
BeanFactory.getBean(UserDao2())
```

但是 接口和工厂类产生了耦合
UserDao BeanFactory

4) xml配置 + 反射 + 工厂解耦 (IoC底层的实现)

IoC 不是什么技术, 而是一种设计思想

在Java开发中 IoC 意味着将你设计好的对象交给容器, 而不是传统的在你的对象内部直接控制

控制: IoC容器来控制对象的创建

控制了外部资源获取 (不只是对象也可是文件等)

反转: 因为由容器帮我们查找及注入依赖对象, 对象只是被动的

接受依赖对象, 所以是反转 (PS: 正转: 传统应用程序由我们自己在对象中主动控制去直接获取依赖对象)

依赖对象的获取被反转了

其实IoC对编程带来的最大改变不是从代码上, 而是从思想上, 发生了“主从换位”的变化。应用程序原本是老大, 要获取什么资源都是主动出击, 但是在IoC/DI思想中, 应用程序就变成被动的了, 被动的等待IoC容器来创建并注入它所需要的资源了。

IoC很好的体现了面向对象设计法则之一——好莱坞法则: “别找我们, 我们找你”; 即由IoC容器帮对象找相应的依赖对象并注入, 而不是由对象主动去找

DI: 由容器动态的将某个依赖关系注入到组件之中

IoC vs DI 是同一个概念的不同角度描述
思想 实现方式

总之依赖注入的意思是你需要的东西不是由你创建的, 而是第三方, 或者说容器提供给你的。这样的设计符合正交性, 即所谓的松耦合。

为了更好的理解，我找了一个例子：

一个人(Java实例，调用者)需要一把斧子(Java实例，被调用者)

在原始社会里，几乎没有社会分工；需要斧子的人(调用者)只能自己去磨一把斧子(被调用者)；对应情形为：Java程序里的调用者自己创建被调用者，通常采用new关键字调用构造器创建一个被调用者

进入工业社会，工厂出现了，斧子不再由普通人完成，而在工厂里被生产出来，此时需要斧子的人(调用者)找到工厂，购买斧子，无须关心斧子的制造过程；对应简单工厂设计模式，调用者只需定位工厂，无须管理被调用者的具体实现

进入“共产主义”社会，需要斧子的人甚至无须定位工厂，“坐等”社会提供即可；调用者无须关心被调用者的实现，无须理会工厂，等待Spring依赖注入