关于 Python 面向对象设计原则

在 Python 编码过程中，面向对象思想会提供更便捷、更贴合人类思想的代码逻辑思考，根据祁老师的讲解及资料考究，面向对象设计原则可以分为三个特征六个方向：

三个特征：

封装，继承，多态

六个方向：

开闭原则、单一原则、依赖倒置原则、组合复用原则、里氏替换、迪米特法则

三个特征和六个方向本身互相包含互相牵扯，所以为了方便阐述，本次论点以六个方向为主，其中会穿插三个特征

• 关于开闭原则【Open Closed Principle】：

开闭原则是面向对象方法的主要方向。
“开”指的是允许一个类甚至往大了说允许一个系统随时可以对自己的功能进行扩展。
“闭”指的是不允许在扩展和修改功能的时候触及到已经写好的底层代码（比如父类）。

举一个比较浅显的例子，可以理解为电脑与硬盘以及 U 盘的关系。

面向过程类型的编写会把所有关键代码写在一起，就好比在给一个已经装好的主机箱添加硬盘，那首先需要先拆开主机箱，然后将装机时为了美观扎好的数据线进行拆解，选择数据线插在硬盘上，再把剩下的线重新扎好，重新封装好主机箱，费时费力。

而如果想给已经装好的主机箱加一个 U 盘，只需要将准备好的 U 盘对准 USB 接口接入，就可以了，在整个过程中主机箱早就装好的内部构造并没有任何变化；而如果想对 U 盘进行扩容或者修改，那么只需要操作 U 盘甚至是替换 U 盘，过程中主机箱是不会发生任何改变的。相比之下，效率就更高了。

那么把这个原理套用在代码之中，尽可能将后期会变化的因素放在外部,而将确定不会产生改变的固定因素作为底层,这样在后期代码扩展和修改中效率就会比较理想。同时，这种将代码互相拆分，来避免在修改过程中牵一发而动全身的特点，称之为 封装（三个特征）。

开闭原则是代码编写的终极目标，而面向编程的其他五个设计方向全部都是以开闭原则为基本目标的实例思想。

• 关于单一原则【Single Responsibility Principle】：

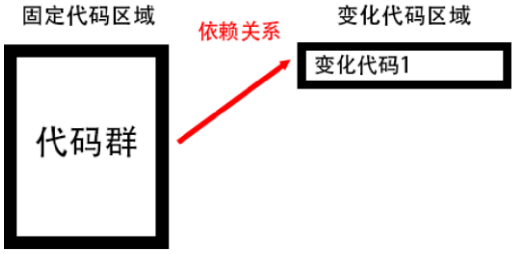
单一原则泛指[类的单一原则]，很多人认为这个原则本意上是要求每一个类应当只有一个功能。其实不然，类的单一原则并不是体现在实现上的，而是体现在修改上的，因为一切的原则都是希望代码的结构方便修改，如果只是为了区分功能而去区分类，那么虽然也会降低耦合性但是太激进了，有时候一个类放少数几个功能在书写上也比较便捷。

那么单一原则的单一体现在哪里？

我个人认为所谓的单一指的应该是一个类能被修改的原因只能有一个，单一原则直接针对的问题是代码的耦合性，而所谓的耦合性指的就是在面临修改的时候被波及的范围，那么假设一个类里存在两个或者三个功能，但是每次整个类只会因为同一件事情去发生修改，发生其他问题都不会触及到此类，那么耦合性就已然降至理想状态了，除此之外的分类细化纵然没有影响代码后期的工作难度，但也是在消耗生成代码时的时间精力。

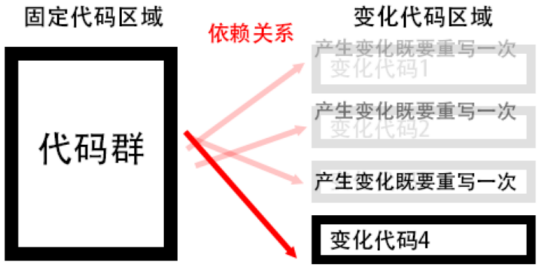
- 关于依赖倒置【**D**ependence **I**nversion **P**rinciple】：

说到依赖倒置要先阐述一下依赖关系，在面向对象思想过程中首先要做的是分，将需要改变的代码分离出去，将不需要改变的代码整合到一起，至此，我们实现了最基本的两个封装，之后，整合的代码往往需要调取会产生改变的代码，这种关系可以理解为最直接的依赖关系



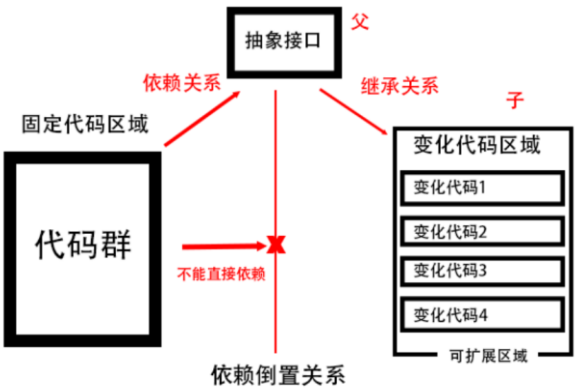
直接依赖关系

而当产生变化时，这种基础依赖关系就会随之产生变化，变化的越多，依赖关系修改的越多。



依赖关系发生改变

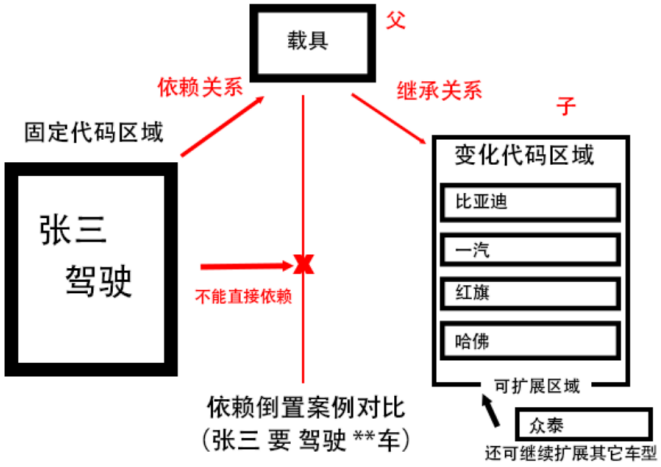
为了避免后期代码的变化导致工程量的增加,首先要做的是将会变化的代码变成一个大的框架,给它们一个统一的类型,也就是统一继承（三个特征）给一个父类，父类本身只是作为一个中转站用来衔接固定代码区域和变化代码区域，换句话说，此时固定代码区域如果想要调用变化代码区域，必须要经过父类，通过把传统的依赖具象代码关系转变成依赖抽象代码方式，就称之为依赖倒置。



依赖倒置关系

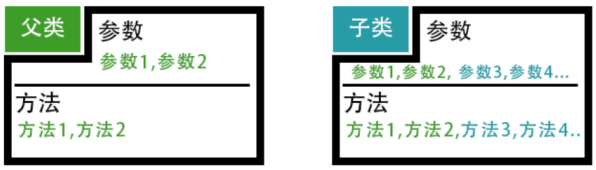
以下是对这个代码结构进行举例演示，比如“张三驾驶汽车”这件事，“张三”这个人以及“驾驶”这件事在当前的架构里是不会发生改变的，所以统一封装在固定代码区域，而驾驶的“汽车”作为可能发生变化的代码封装在变化代码区域，随后所有汽车类再统一继承给一个叫做“载具”的父类，这个类因为没有具体的特征所以是抽象的，而张三每次驾驶不同载具都要先指向“载具”，再由“载具”按需求去挑选继承了自己的每一个具体的汽车子类，如果有新的需求，还可以再继续扩展子类。

如果一开始“张三”作为人也是可以替换的，或者“驾驶”这件事也是可以替换的，那么同样将“张三”封装到一个变化区域，继承给一个叫做“人”的父类；“驾驶”这件事也可以继承给类似于“行为”这样的抽象父类，既满足依赖倒置原则。



- 关于组合复用【**C**omposite **R**euse **P**rinciple】：

组合复用原则需要对之前已经讲过的继承先进行一个详细介绍：

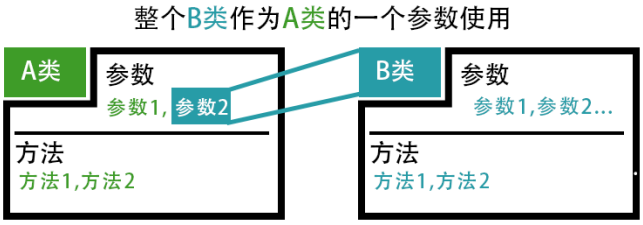


继承(泛化)关系

关于继承关系，声明一个父类，里面可以内置若干参数，若干方法，而之后声明的子类，只要是继承自这个父类，那么就可以直接获取关于父类的参数和方法，除此之外，子类还可以拥有自己专属的额外参数和方法，这些新的参数和方法父类是不能享用的。

原本继承关系就已经可以满足初步的封装和降低耦合，但是程序员不满足于继承关系里的“血缘关系”，因为毕竟子类在继承父类的时候，已经把所有参数和方法都复制过来了，那么假设父类发生了变化，所有子类都会受到影响，这样耦合度的情况又会显得非常尴尬，于是，一种比继承耦合度还要低的关系就诞生了，而低于继承耦合度的关系有两个，一个被称为关联，另一个被称为依赖，与之对应，继承也有了一个新名字：泛化。

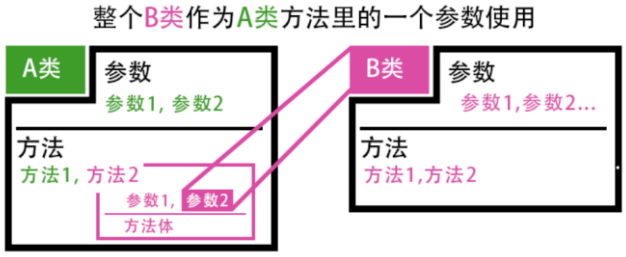
所谓的关联关系，也被称为聚合关系或者组合关系，指的是两个类之间不做直接继承，而是将一个类作为另一个类其中的一个参数进行连接。



关联(聚合 / 组合)关系

通过参数连接就避免了继承必须继承所有参数和方法的弊端，在耦合度上相对于继承又下降了许多。

最后是依赖关系，也被称为合作关系，在关联关系的基础上进一步降低了耦合度，其中一个类的整体将只能作为另一个类的方法中的一个参数来进行调用。



依赖(合作)关系

这是三种关系中耦合度最低的了，毕竟参数可能不可或缺，方法调不调却是随意的了。

所以对这三种关系进行耦合度排序的话：
依赖关系耦合度最低　　关联关系耦合度适中　继承关系耦合度相对最高

那么结合以上结论来阐述组合复用原则就显得简单明了多了。原意指的是在组合关系和继承关系都能满足业务要求时优先使用组合关系。

不过根据整体思想来说，我个人的观点是应该将组合复用理解为既是指在三种关系都能满足要求的情况下，尽量使用其中耦合度最低的关系。

所以尽管相比之下继承关系耦合度并不比其他两种关系好，但是不代表继承关系一文不值，在很多情况下，继承的实用性依然不错。

- 关于里氏替换【**L**iskov　**S**ubstitution **P**rinciple】：

里氏替换针对的依然是继承关系，上文提到过，继承关系指的是子类在继承父类的时候会继承父类内部的所有参数和方法，那么就意味着，只要是父类能参与的任何构造，子类完全可以胜任，所以里氏替换的主要思想就是只要父类可以被调用，子类就一定要代替父类被调用。这种情况并不是说父类没有意义，相反的，里氏替换进一步要求父类尽可能不要存在太具体的功能，能抽象就尽量抽象，任何的修改都完全依靠子类来补充和修改，从而进一步实现开闭原则（父类对修改关闭，子类对修改开放）

同时补充一下，因为每一个子类在父类的构造上都可以额外拓展，于是，就算每一个子类内部的方法名字跟父类完全一样，但是因为每个子类都对这同一个方法产生了新的拓展，所以在调用这个方法的时候，方法名始终没有任何改变，但是方法本身却是随着子类的不同而千变万化的。这种现象就称之为多态（三个特征）。

这种同样的方法名不同的方法内容就称之为多态

所以里氏替换原则也是在为父类的抽象化和子类的多态化进行阐述。

• 关于迪米特法则【Law Of Demeter】：

迪米特法则也被戏称为“最少知道法则”，其实说白了就是针对面向对象里的低耦合。

它的本意指的是类与类之间尽可能不要有太多的关联，当一个类需要产生变化时，其他的类尽量做到不产生改动。

具体的体现其实与上面阐述的单一原则是一致的，既每一个类最好能做到只会被同一件事情影响和改变，其他类尽可能不受其影响。

• 小结

其实不论是三个特征还是六个方向，说白了它们的本意都是在代码的结构上尽可能让后期的维护工作变得轻松。面向对象思想是许多前辈在工作中总结出的一个让工作相对轻松的思想，是需要花时间去琢磨和感悟的，它太过于抽象，以至于不好形容不好表达，甚至有可能会让人陷入误区，所以，面向对象是一个值得去慢慢体会的东西，但是不能为了刻意的封装而封装，只要代码的修改工程量尽可能压到自己能接受的合理限度，代码的味道就是好的。

懒是人类进步的阶梯，也是面向思想对象的催化剂，当什么时候意识到自己改代码改到累了，希望下次再改代码能省事的时候，面向对象思想就在前方招手了。

共勉。