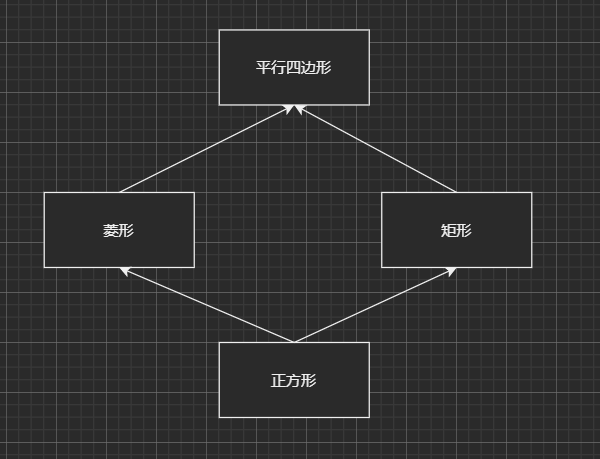
Atitit oop的缺点面向对象

OOP，**面向对象的编程**  
一直我都把Java看的比其他语言重要，今天突发奇想，想看看Java的缺点，然鹅，Java的知识点太多了，一两句话就说完没意义，写几万字也不行，所以先找了OOP的**麻烦**。

## “罪过1”

熟悉的人都知道，继承使得复用以前的代码非常容易，能够大大缩短开发周期，降低开发费用。  
那么，**面向对象语言的问题在于，它们依赖于特定的环境。你想要个香蕉，但拿到的却是拿着香蕉的猩猩，乃至最后你拥有了整片丛林。**  
就是它的问题了。简而言之，就是你不仅要拿来真正需要的类，还可能附带把类的父类，父类的父类等等，也拿来了。怎么解决我不说，O(∩\_∩)O

## “罪过2”

菱形继承，就类似这样，正方形同时属于菱形和矩形，而矩形和菱形都属于平行四边形。当正方形继承某个菱形和矩形都有的方法时，继承哪一个呢？  
visual studio code画图  


## “罪过3”

封装是用来保护对象状态变量的，但是，遇到**引用**时，就可能出现问题了。

如果一个对象的引用被传递给另一个对象的构造函数，构造函数就能将这个对象引用放到私有变量中，用封装保护起来。  
但这个传递的对象不是安全的！  
为什么不是？因为其他代码也可能拥有指向该对象的指针，比如调用构造函数的那段代码。它必须有指向对象的引用，否则没办法传递给构造函数。

所有主流面向对象语言都有这个问题

1. 过度封装

使用OOP时，会把一些复杂的问题分拆抽象成较简单的独立对象，通过对象的互相调用去实现方案。但是，由于对象包含自己封装的数据，一个问题的数据集会被分散在不同的内存区域。互相调用时很可能会出现数据的cache miss的情况。

2. 多态

在C++的一般的多态实现中，会使用到虚函数表。虚函数表是通过加入一次间接层来实现动态派送。但在调用的时候需要读取虚函数表，增加cache miss的可能性。基本上要支持动态派送，无论用虚函数表、函数指针都会形成这个问题，但如果类的数目极多，把函数指针如果和数据放在一起有时候可放缓问题。

3. 数据布局

虽然OOP本身并无限制数据的布局方式，但基本上绝大部分OOP语言都是把成员变量连续包裹在一段内存中。甚至使用C去编程的时候，也通常会使用到OOP或Object-based的思考方式，把一些相关的数据放置于一个struct之内：

大部分问题是面向对象无法解决的。能够采用面向对象简化模型得到代码重用性的时候，就用；不能的时候，就放弃，不要到处都用。  
有的场合下，面向对象会增加代码复杂度，增加维护的困难。  
除了面向对象，我们还有很多办法来实现我们要做的功能。不谈更加高端大气的函数式编程，其实用一串switch case或者else if就可以实现，代码还简单。  
所有需要的东西都放在一起，同一个文件里面同一段代码，阅读起来并不困难，很整齐。某个case下面太长的时候，也可以独立出来包装成一个函数调用；你爱放在同一个文件里面还是另开一个文件都可以。  
可是面向对象是一个很重量级的方法。你得设计类的继承关系。得写类的声明，得实现不同的虚方法；实现时要注意是否要调用父类的方法。调试的时候经常搞不清楚到底执行哪个类的虚方法了。而用一大堆switch case，语句执行顺序一目了然。  
可能还是觉得代码层级过多？有很多办法可以整理得更干净。用了面向对象，省了switch case，却要写更多的virtual function的声明和实现。  
业务逻辑本身是复杂的，无法避免。没有银弹。

作者：白起  
链接：https://www.zhihu.com/question/20275578/answer/26543719  
来源：知乎  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。