跳出面向对象思想(一) 继承 (http://casatwy.com/tiao-chu-mian-xiang-dui-xiang-si-xiang-yi-ji-cheng.html)

Date Mon 01 December 2014 Tags Object Oriented Programming (http://casatwy.com/tag/object-oriented-programming.html) / Experience (http://casatwy.com/tag/experience.html) / jooo (http://casatwy.com/tag/jooo.html)

简述

我会在~~这篇~~这一系列文章中谈谈面向对象思想的几个部分，并且给出对应的解决方案，这些解决方案有些是用面向过程的思路解决的，有些也还是停留在面向对象中。到最后我会给大家一个比较，然后给出结论。

上下文规范

在进一步地讨论这些概念之前，我需要跟大家达成一个表达上的共识，我会采用下面的语法来表达对象相关的信息：

Social

RSS (http://casatwy.com/feeds/all.atom.xml)

github (http://github.com/casatwy)

facebook (https://www.facebook.com/taloyum)

google+ (https://plus.google.com/u/0/108264119649922067163)

weibo (http://weibo.com/casatwy)

Tags

(http://casatwy.com/)

Links

casatwy (http://casatwy.com/)

刘坤的技术博客 (http://blog.cnbluebox.com)

齐道长的博客 (http://qitaos.github.io)

所有的大写字母都是类或对象，小写字母表示属性或方法。

`FOO:{ isLoading, _data, render(), _switch() }`	这表示一个FOO对象，isL
`A -> B`	这表示从A派生出了B，A是
`A -> B:{ [a, b, c(), d()], e, f() }`	[]里面是父类的东西，e、
`B:{ [ A ], e, f() }`	省略了对父类的描述，用
`B:{ [ A ], e, f(), @c() }`	省略了对父类的描述，函
`B:{ [ A,D ], e, f() }`	多继承，B继承了A和D
`B<protocol>`	符合某个protocol接口的
`<protocol>:{foo(), bar}`	protocol这个接口中包含
`foo(A, int)`	foo这个函数，接收A类和

来，我们谈谈对象

面向对象思想三大支柱：继承、封装、多态。这篇文章说的是继承。当然面向对象和面向过程都会有好有坏，但是做决定的时候，更多地还是去权衡值得不值得放弃。关于这样的立场问题，我都会给出非常明确的倾向，不会跟你们打太极。
如果说这个也好那个也好，那还发表毛个观点，那叫没有观点。

继承

继承从代码复用的角度来说，特别好用，也特别容易被滥用和被错用。不恰当地使用继承导致的最大的一个~~缺陷~~特征就是**高耦合**。
在这里我要补充一点，耦合是一个**特征**，虽然大部分情况是缺陷的特征，但是**当耦合成为需求的时候，耦合就不是缺陷**了。耦合成为需求的例子在后面会提到。
我们来看下面这个场景：

有一天，产品经理Yuki说:

> 我们不光首页要有一个搜索框，在进去的这个页面，也要有一个搜索框，只不过这个搜索框要多一些功能，它是可以即时给用户搜索提示的。

Casa接到这个任务，他研究了一下代码，说:OK，没问题~
Casa知道代码里已经有了一个现成的搜索框，Casa立刻从 `HOME_SEARCH_BAR` 派生出 `PAGE_SEARCH_BAR`
嗯，目前事情进展到这里还不错：

```
HOME_SEARCH_BAR:{textField, search(), init()}
PAGE_SEARCH_BAR:{ [ HOME_SEARCH_BAR ], overlay, prompt() }
```

过了几天，产品经理Yuki要求:

> 用户收藏的东西太多了，我们的app需要有一个本地搜索的功能。

Casa轻松通过方法覆盖摆平了这事儿:

```
HOME_SEARCH_BAR:{textField, search()}
PAGE_SEARCH_BAR:{ [ HOME_SEARCH_BAR ], overlay, prompt() }
LOCAL_SEARCH_BAR:{ [ HOME_SEARCH_BAR ], @search() }
```

---

app上线一段时间之后，UED不知哪根筋搭错了，决定要修改搜索框的UI，UED跟Casa说:

> 把HOME_SEARCH_BAR的样式改成这样吧，里面PAGE_SEARCH_BAR还是老样子就OK。

Casa表示这个看似简单的修改其实很蛋碎， `HOME_SEARCH_BAR` 的样式一改， `PAGE_SEARCH_BAR` 和 `LOCAL_SEARCH_BAR` 都会改变，怎么办呢？ 与其每个手工修一遍，Casa不得已只能给 `HOME_SEARCH_BAR` 添加了一个函数： `initWithStyle()`

```
    HOME_SEARCH_BAR:{ textField, search(), init(), initWithStyle()
    PAGE_SEARCH_BAR:{ [ HOME_SEARCH_BAR ], overlay, prompt() }
    LOCAL_SEARCH_BAR:{ [ HOME_SEARCH_BAR ], @search() }
```

于是代码里面就出现了各种init()和initWithStyle()混用的情况。

> 无所谓了，先把需求应付过去再说。

Casa这么想。

有一天，另外一个team的leader来对Casa抱怨:

> 搞什么玩意儿？为毛我要把LOCAL_SEARCH_BAR独立出来还特么连带着把那么多文件都弄出来？我就只是想要个本地搜索的功能而已！！

这是因为 LOCAL_SEARCH_BAR 依赖于它的父类 HOME_SEARCH_BAR ,然而 HOME_SEARCH_BAR 本身也带着API相关的对象，同时还有数据解析的对象。 也就是说，要想把 LOCAL_SEARCH_BAR 移植给另外一个TEAM，拔出萝卜带出泥，差不多整个Networking框架都要移植过去。 嗯，Casa又要为了解耦开始一个不眠之夜了～

以上是典型的错误使用继承的案例，虽然继承是代码复用的一种方案，但是使用继承仍然是需要好好甄别代码复用的方式的，不是所有场景的代码复用都适用于继承。

> 继承是紧耦合的一种模式，主要的体现就在于牵一发动全身。

- 第一种类型的问题是改了一处，到处都要改，但解决方案还算方便，多添加一个特定的函数(initWithStyle())就好了。只是代码里面难看一点。
- 第二种类型的问题是代码复用的时候，要跟着把父类以及父类所有的相关依赖也复制过去，高耦合在复用的时候造成了冗余。

对于这样的问题，业界其实早就给出了解决方案：**用组合替代继承**。将Textfield和search模块拆开，然后通过定义好的接口进行交互，一般来说可以选择Delegate模式来交互。

解决方案:

```
<search_protocol>:{search()}

SEARCH_LOGIC<search_protocol>

SEARCH_BAR:{textField, SEARCH_LOGIC<search_protocol>}

HOME_SEARCH_BAR:{SearchBar1, SearchLogic1}
PAGE_SEARCH_BAR:{SearchBar2, SearchLogic1}
LOCAL_SEARCH_BAR:{SearchBar2, SearchLogic2}
```

这样一来，搜索框和搜索逻辑分别形成了两个不同的组件，分别在 HOME_SEARCH_BAR，PAGE_SEARCH_BAR，LOCAL_SEARCH_BAR 中以不同的形态组合而成。 textField 和 SEARCH_LOGIC<search_protocol> 之间通过delegate的模式进行数据交互。 这样就解决了上面提到的两种类型的问题。 **大部分我们通过代码复用来选择继承的情况，其实都是变成组合比较好。** 因此我在团队中一直在推动使用组合来代替继承的方案。 那么什么时候继承才有用呢?

纠结了一下，貌似实在是没什么地方非要用继承不可的。但事实上使用继承，我们得要分清楚层次，使用继承其实是如何给一类对象划分层次的问题。在正确的继承方式中，父类应当扮演的是底层的角色，子类是上层的业务。举两个例子:

```
Object -> Model
Object -> View
Object -> Controller

ApiManager -> DetailManager
ApiManager -> ListManager
ApiManager -> CityManager
```

这里是有非常明确的层次关系的，我在这里也顺便提一下使用继承的3大要点:

> 父类只是给子类提供服务，并不涉及子类的业务逻辑

```
Object并不影响Model, View, Controller的执行逻辑和业务
Object为子类提供基础服务，例如内存计数等

ApiManager并不影响其他的Manager
ApiManager只是给派生的Manager提供服务而已,ApiManager做的只会是份内的是
```

> 层级关系明显，功能划分清晰，父类和子类各做各的。

> Object并不参与MVC的管理中，那些都只是各自派生类自己要处理的事情
>
> DetailManager, ListManager, CityManager都只是处理各自业务的对象
> ApiManager并不应该涉足对应的业务。

> 父类的所有变化，都需要在子类中体现，也就是说此时耦合已经成为需求

> Object对类的描述，对内存引用的计数方式等，都是普遍影响派生类的。
> ApiManager中对于网络请求的发起，网络状态的判断，是所有派生类都需要的。
> 此时，牵一发动全身就已经成为了需求，是适用继承的

此时我们回过头来看为什么
`HOME_SEARCH_BAR`，`PAGE_SEARCH_BAR`，`LOCAL_SEARCH_BAR` 采用继承的方案是不恰当的：

- 他们的父类是 `HOME_SEARCH_BAR`，父类不只提供了服务，也在一定程度上影响了子类的业务逻辑。派生出的子类也是为了要做搜索，虽然搜索的逻辑不同，但是互相涉及到搜索这一块业务了。

- 子类做搜索，父类也做搜索，虽然处理逻辑不同，但是这是同一个业务，与父类在业务上的联系密切。在层级关系上，`HOME_SEARCH_BAR` 和其派生出的 `LOCAL_SEARCH_BAR`，`PAGE_SEARCH_BAR` 其实是并列关系，并不是上下层级关系。

- 由于这里所谓的父类和子类其实是并列关系而不是父子关系，且并没有需要耦合的需求，相反，每个派生子类其实都不希望跟父类有耦合，此时耦合不是需求，是缺陷。

总结

可见，代码复用也是分类别的，如果当初只是出于代码复用的目的而不区分类别和场景，就采用继承是不恰当的。我们应当考虑以上3点要素看是否符合，才能决定是否使用继承。就目前大多数的开发任务来看，继承出现的场景不多，主要还是代码复用的场景比较多，然而通过组合去进行代码复用显得要比继承麻烦一些，因为组合要求你有更强的抽象能力，继承则比较符合直觉。然而从未来可能产生的需求变化和维护成本来看，使用组合其实是很值得的。另外，当你发现你的继承超过2层的时候，你就要好好考虑是否这个继承的方案了，第三层继承正是滥用的开端。确定有必要之后，再进行更多层次的继承。

所以我的态度是： 万不得已不要用继承，优先考虑组合

评论系统我用的是Disqus，不定期被墙。所以如果你看到文章下面没有加载出评论列表，翻个墙就有了。

我的博客拒绝挂任何广告，如果您觉得文章有价值，可以通过支付宝扫描下面的二维码捐助我。

Comments

80 条评论 Casa Taloyum 1 登录

Recommend 分享 按从新到旧排序

Join the discussion…

eric · 4天前

还有个问题想咨询下：
“把HOME_SEARCH_BAR的样式改成这样吧，里面PAGE_SEARCH_BAR还是老样子就OK”
博主在HOME_SEARCH_BAR这个基类里增加一个initWithStyle进行解决。
（1）是不是就紧紧是HOME_SEARCH_BAR在创建实例的时候由init变成initWithStyle，然后在initWithStyle里面根据style类型更改bar的样式。PAGE_SEARCH_BAR和LOCAL_SEARCH_BAR在创建实例的时候还是用init吗?

（2）这样做了以后貌似就不符合Initializer Patterns了
https://blog.twitter.com/2014/...

· 回复 · 分享 ›

CasaTaloyum 管理员 eric · 4天前

你特么有没有搞明白我举的这个initWithStyle是反例？能不要断章取义吗？更何况initWithStyle哪里违背Initializer Patterns了？你发的那篇文章你理解的也是断章取义啊！

· 回复 · 分享 ›

eric CasaTaloyum · 4天前

博主息怒，楼主的文章我可以说是一个字一个字去精读的，所以不会存在只看一句话而忽略整体环境。

我知道博主前面举的是一个反例，其实我本来只是想针博主说的那个模拟情景（就是如果一开始前人已经继承埋了这个坑了，然后因为项目紧，也没时间去重构成下面这种组合的方式，而采取的一种临时解决方案）下，咨询下initwithstyle（）以及init实现的细节的问题的。不过刚才提问的有点匆忙，尤其是Initializer Patterns，没有看里面的实现，就根据自己的设想做假设确实是不妥的，现在自己也想明白了。

其实包括其他前面提的一些问题，有的时候真的是博主误解我提问的初衷了。不过，我觉得主要原因还在于我表达的不够清楚，后面我会改进的，谢谢

↑ Back to top