跳出面向对象思想(三) 封装 (http://casatwy.com/tiao-chu-mian-xiang-dui-xiang-si-xiang-san-feng-zhuang.html)

Date Mon 12 January 2015 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

我认为"封装"的概念在面向对象思想中是最基础的概念，它实质上是通过将相关的一堆函数和一堆对象放在一起，对外有函数作为操作通道，对内则以变量作为操作原料。只留给外部程序员操作方式，而不暴露具体执行细节。大部分书举的典型例子就是汽车和灯泡的例子：你不需要知道不同车子的发动机原理，只要踩油门就可以跑；你不需要知道你的灯泡是那种灯泡，打开开关就会亮。我们都会很直觉地认为这种做法非常棒，是吧?

但是有的时候还是会觉得有哪些地方不对劲，使用面向对象语言的时候，我隐约觉得封装也许并没有我们直觉中认为的那么好，也就是说，面向对象其实并没有我们直觉中的那么好，虽然它已经流行了很多很多年。

1. 将数据结构和函数放在一起是否真的合理?

函数就是做事情的，它们有输入，有执行逻辑，有输出。 数据结构就是用来表达数据的，要么作为输入，要么作为输出。

两者本质上是属于完全不同的东西，面向对象思想将他们放到一起，使得函数的作用被限制在某一个区域里，这样做虽然能够很好地将操作归类，但是这种归类方法是根据"

作用领域"来归类的，在现实世界中可以，但在程序的世界中，有些不妥。

不妥的理由有如下几个：

> 在并行计算时，由于执行部分和数据部分被绑定在一起，这就使得这种方案制约了并行程度。在为了更好地实现并行的时候，业界的工程师们发现了一个新的思路：函数式编程。将函数作为数据来使用，这样就能保证执行的功能在时序上的正确性了。但你不觉得，只要把数据表达和执行部分分开，形成流水线，这不就能够非常方便地将并行数提高了么？

我来举个例子： 在数据和函数没有分开时，程序的执行流程是这样：

```
A.function1() -> A.function2() -> A.function3()     最后得到经过处理的
```

当处于并发环境时，假设有这么多任务同时到达

```
A.f1() -> A.f2() -> A.f3()     最后得到经过处理的A
B.f1() -> B.f2() -> B.f3()     最后得到经过处理的B
C.f1() -> C.f2() -> C.f3()     最后得到经过处理的C
D.f1() -> D.f2() -> D.f3()     最后得到经过处理的D
E.f1() -> E.f2() -> E.f3()     最后得到经过处理的E
F.f1() -> F.f2() -> F.f3()     最后得到经过处理的F
...
```

假设并发数是3，那么完成上面类似的很多个任务，时序就是这样

```
| time | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   | 10
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----
| A    | A.1 | A.2 | A.3 |     |     |     |     |     |     |
| B    | B.1 | B.2 | B.3 |     |     |     |     |     |     |
| C    | C.1 | C.2 | C.3 |     |     |     |     |     |     |
| D    |     |     |     | D.1 | D.2 | D.3 |     |     |     |
| E    |     |     |     | E.1 | E.2 | E.3 |     |     |     |
| F    |     |     |     | F.1 | F.2 | F.3 |     |     |     |
| G    |     |     |     |     |     |     | G.1 | G.2 | G.3 |
| H    |     |     |     |     |     |     | H.1 | H.2 | H.3 |
| I    |     |     |     |     |     |     | I.2 | I.2 | I.3 |
| J    |     |     |     |     |     |     |     |     |     | J.1
| K    |     |     |     |     |     |     |     |     |     | K.1
| L    |     |     |     |     |     |     |     |     |     | L.1
```

当数据和函数分开时，并发数同样是3，就能形成流水线了，有没有发现吞吐量一下子上来了？

```
| time | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10| 11| 12|
|------|---|---|---|---|---|---|---|---|---|---|---|---|
| f1() | A | B | C | D | E | F | G | H | I | J | K | L |
| f2() | Z | A | B | C | D | E | F | G | H | I | J | K |
| f3() | Y | Z | A | B | C | D | E | F | G | H | I | J |
```

你要是粗看一下，诶？怎么到了第13个周期K才刚刚结束？上面一种方案在第12个周期的时候就结束了？不能这么看的哦，其实在12个周期里面，Y、Z也已经交付了。因为流水线吞吐量的提升是有过程的，我截取的片段应该是机器在持续运算过程中的一个片段。

我们不能单纯地去看ABCD，要看交付的任务数量。在12个周期里面，大家都能够完成12个任务，在11个周期里面，流水线完成了11个任务，前面一种只完成了9个任务，流水线的优势在这里就体现出来了：每个时间段都能稳定地交付任务，吞吐量很大。而且并发数越多，跟第一种方案比起来的优势就越大，具体的大家也可以通过画图来验证。

> 数据部分就是数据部分，执行部分就是执行部分，不同类的东西放在一起是不合适的

函数就是一个执行黑盒，只要满足函数调用的充要条件（给够参数），就是能够确定输出结果的。面向对象思想将函数和数据绑在一起，这样的封装扩大了代码重用时的粒度。如果将函数和数据拆开，代码重用的基本元素就由对象变为了函数，这样才能更灵活更方便地进行代码重用。

嗯，谁都经历过重用对象时，要把这个对象所依赖的所有东西都要移过来，哪怕你想用的只是这个对象里的一个方法，然而很有可能你的这些依赖是跟你所需要的方法无关的。

但如果是函数的话，由于函数自身已经是天然完美封装的了，所以如果你要用到这个函数，那么这个函数所有的依赖你都需要，这才是合理的。

2. 是否所有的东西都需要对象化?

面向对象语言一直以自己做到"一切皆对象"为荣，但事实是：是否所有的东西都需要对象化?

在iOS开发中，有一个类叫做NSNumber，它封装了所有数值：double，float，unsigned int, int...等等类型，在使用的时候它弱化了数值的类型，使得非常方便。但问题也来了，计算的时候是不能直接对这个对象做运算的，你得把它们拆成数值，然后进行运算，然后再把结果变成NSNumber对象，然后返回。这是第一点不合理。第二点不合理的地方在于，运算的时候你不知道原始数据的类型是什么，拆箱装箱过程中难免会导致内存的浪费（比如原来uint8_t的数据变成unsigned int），这也十分没有必要。

还有就是我们的file descriptor，它本身是一个资源的标识号，如果将资源抽象成对象，那么不可避免的就会使得这个对象变得非常庞大，资源有非常多的用法，你需要将这些函数都放到对象里去。在真正传递资源的时候，其实我们也只是关心资源标识而已，其它的真的无需关心。

我们已经有函数作为黑盒了，拿着数据塞到黑盒里就够了。

3. 类型爆炸

由于数据和函数绑定到了一起，在逻辑上有派生关系的两种对象往往可以当作一种，以派生链最上端的那个对象为准。单纯地看这个现象直觉上会觉得非常棒，父亲有的儿子都有。但在实际工程中，派生是非常不好控制的，它导致同一类类型在工程中泛滥：ViewController、AViewController、BViewController、ThisViewController、ThatViewController...

你有没有发现，一旦把执行和数据拆解开，就不需要这么多ViewController了，派生只是给对象添加属性和方法。但事实上是这样：

```
struct A {              Class A extends B
    struct B b;         {
    int number;             int number;
}                       {
```

前者和后者的相同点是：在内存中，它们的数值部分的布局是一模一样的。不同点是：前者更强烈地表达了组合，后者更强烈地表达的是继承。然而我们都知道一个常识：组合要比继承更加合适，这在我这一系列的第一篇文章 (http://casatwy.com/tiao-chu-mian-xiang-dui-xiang-si-xiang-yi-ji-cheng.html)中有提到。

上两者的表达在内存中没有任何不同，但在实际开发阶段中，后者会更容易把项目引入一个坏方向。

总结

为什么面向对象会如此流行？我想了一下业界关于这个谈论的最多的是以下几点：

1. 它能够非常好地进行代码复用
2. 它能够非常方便地应对复杂代码
3. 在进行程序设计时，面向对象更加符合程序员的直觉

第一点在理论上确实成立，但实际上大家都懂，在面向对象的大背景下，写一段便于复用的代码比面向过程背景下难多了。关于第二点，你不觉得正是面向对象，才把工程变复杂的么？如果层次清晰，调用规范，无论面向对象还是面向过程，处理复杂业务都是一样好，等真的到了非常复杂的时候，对象间错综复杂的关系只会让你处理起来更加头疼，不如面向过程来得简洁。关于第三点，这其实是一个障眼法，因为无论面向什么的设计，最终落实下来，还是要面向过程的，面向对象只是在处理调用关系时符合直觉，在架构设计时，理清需求是第一步，理清调用关系是第二步，理清实现过程是第三步。面向对象让你在第二步时就产生了设计完成的错觉，只有再往下落地到实现过程的时候，你才会发现第二步中都有哪些错误。

所以综上所述，我的观点是：面向对象是在架构设计时非常好的思想，但如果只是简单映射到程序实现上来，引入的缺点会让我们得不偿失。

后记

距离上一次博文更新已经快要一个月了，不是我偷懒，实在是太忙，现在终于有时间可以把"跳出面向对象"系列完成了。针对面向对象的3个支柱概念我写了三篇文章来挑它的刺，看上去有一种全盘否定的感觉，而我倒不至于希望大家回去下一个项目就开始面向过程的开发，我希望大家能够针对这一系列文章提出的面向对象的弊端，严格规范代码的行为，知道哪些可行哪些不可行。过去的工作中我深受其苦，往往没有时间去详细解释为什么这么直觉的东西实际上不可行，要想解释这些东西就得需要各种长篇大论。最痛苦的是，即便长篇大论说完了，最后对方还无法理解，照样写出垃圾代码出来害人。

现在好了，长篇大论落在纸上了，说的时候听不懂，回去总可以翻文章慢慢理解了吧。

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

21 条评论 Casa Taloyum 1 登录

Recommend 2 分享 按从新到旧排序

Join the discussion...

LunWang · 2个月前

问一下撸主， 不同模块的Target/Action的引用问题。 模块A要用到模块B的Action 模块A要把B作为dependency还是模块A引用Actions文件。

· 回复 · 分享 ›

CasaTaloyum 管理员 LunWang · 2个月前

规范做法是模块B有一个umbrella header，模块A引用模块B的umbrella，然后umbrella里面暴露所有可被外部调用的action。

· 回复 · 分享 ›

LunWang CasaTaloyum · 2个月前

按我个人猜测你们现在用的CocoaPods管理不同模块 如果是按照你上面所说 这个umbrella Header引用应该还是会存在dependency的问题，
我现在的想法是每个模块要么单独做一个接口pod 要么在已有模块做一个单独的subpod（不依赖模块内任何业务逻辑）提供接口给外面用

另外 umbrella Header是我理解的是iOS8做成Framework会自动产生。 不知道对不对
如果 umbrella Header不通过pod的方式 那还是会存在文件导出拷贝，同步的问题

· 回复 · 分享 ›

CasaTaloyum 管理员 LunWang · 2个月前

这个场景中的依赖问题是解决不了的。

即使是你的做法，也还是没有解决依赖问题。真正不依赖的场景是我在组件化那篇文章里提出的方法，但也并不是所有场景都适合使用mediator。

另外，你这个问题其实适合放在组件化那篇文章里面去讨论，而不是这里。

· 回复 · 分享 ›

LunWang CasaTaloyum · 2个月前

不好意思 看的是那篇文章 回的时候就回到这里来了 😔
你们实际项目中肯定有这种情况的依赖。那你们项目中是如何解决的

· 回复 · 分享 ›

CasaTaloyum 管理员 LunWang · 2个月前

就是分场景使用啊。

↑ Back to top