Atitit 提升效率 非严格的面向对象方法和oo缺点

时，计算机科学（CS）专业的学生学习 OOP 的方法非常严格和教条。不仅鼓励实践者以对象和类的形式来构建应用程序，甚至还要求他们从对象和类的角度来考虑问题空间，这种实践方法被称为“面向对象的分析和设计”。

然而，随着时间的流逝，人们开始意识到，严格的面向对象方法会带来很多问题。这些问题往往会使代码更复杂、更难以理解且更难以测试。

事实证明，OOP 更适合某些特定的问题领域。例如，OOP 仍然是构建用户界面（窗口和按钮）最自然的方式 。但是，

## 试图将面向对象技术应用于关系数据库一直是一场灾难。

面我提到了，以面向对象的方式处理关系数据库是有问题的。对象关系映射（ORM）被一位评论家称为“[越南的计算机科学](http://blogs.tedneward.com/post/the-vietnam-of-computer-science/?source=post_page---------------------------" \t "https://www.infoq.cn/article/_blank)”。（注意：那篇文章很长，深奥难懂，而且有一定的倾向性。）

## “鸭嘴兽”效应

现实世界并不总是能被整齐地划分成具有明确属性定义的类别。例如，假设我们创建了一个代表动物王国的类层次结构。该类层次结构中既包含爬行动物（冷血、有鳞片、产卵等等），又包含哺乳动物 （恒温、有毛、生育等等），还包含鸟类、两栖动物、无脊椎动物等等。

然而，对于鸭嘴兽，它似乎不属于我们上述定义的任何类别。我们要做什么呢？我们是创建一个全新的类别，还是重新考虑整个分类方案呢？就工作量和程序复杂性而言，这两种方法都会产生显著的成本

## 九层的类继承好多。

然而，情况还会变得更糟糕。

、因此，举例来说，如果一个按钮具有颜色，我们将通过向常规的“按钮”对象中添加一个“颜色”属性或子对象的形式来实现， 而不是再创建一个新的“颜色按钮”类。

由于这一强制要求，新工具包的类层次结构相对较浅，如果我没记错的话，只有两三层。而且它更容易理解和使用，也更强大了

## 在这些高层类中，有许多都被只与少数子类相关的方法和属性“污染”了。

例如，“Control”类有[90 多个方法](https://google.github.io/closure-library/api/goog.ui.Control.html?source=post_page---------------------------" \t "https://www.infoq.cn/article/_blank)。它具有设置状态的方法，即使特定的子类是无状态的；它有添加和删除子元素的方法，即使控件包含子元素也没有任何意义。

## 组合了数据和操作混杂

最近，我采取了不同的方法。在我最新的编译器中，所有这些内部数据结构都是“哑”的，这意味着它们所做的只是保存数据，而不再是其他的了。所有用于操作和转换对象的代码都在这些对象的外部实现。

这对代码组织结构有很大的好处。每个算法都集中在一个地方，而不是分散在一堆源文件中。当我想测试某个给定的编译器操作时，我可以轻松地创建一些示例对象并将其提供给该操作。因此，我可以更容易得编写测试了，这意味着我可以编写更多的测试来提升测试覆盖率。

# 我更喜欢使用“混合”的方式，在某些地方使用 FP 技术，而在其他地方使用 OOP 技术。

对于某些类型的问题来说，使用 FP 会事半功倍，而另一些问题，使用 OOP 则是明智的选择。

我知道很多 FP 爱好者都是“纯”函数式语言的狂热拥护者，在纯式函数语言中，所有对

面向对象编程的兴衰.html