# 代码的坏味道

|  |  |
| --- | --- |
| **坏味道** | **常用重构** |
| Alternative Classes with Different Interfaces  (异曲同工类) | Rename Method ,  Move Method |
| Comments  (过多注释类) | Extract Method ,  Introduce Assertion |
| Data Class  (纯稚的数据类) | Move Method ,  Encapsulate Field ,  Encapsulate Collection |
| Data Clumps  (数据泥团) | Extract Class ,  Introduce Parameter ,  Preserve Whole Object |
| Drivergent Change  (发散式变化) | Extract Class |
| Duplicated Code  (重复代码) | Extract Method ,  Extract Class ,  Pull Up Method,  From Template Method |
| Feature Envy  (依恋情节) | Move Method ,  Move Field ,  Change Bidirectional ,  Association to Unidirectional ,  Replace Inheritance with Delegation ,  Hide Delegate |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

备注：

Alternative n. 二中择一；供替代的选择

adj. 供选择的；选择性的；交替的

Assertion n. 断言，声明；主张，要求；坚持

Encapsulate n. 压缩

Preserve n. 保护区；禁猎地；加工成的食品

vt. 保存；保护；维持；腌；禁猎

Whole n. 整体；全部

adj. 完整的；纯粹的

Bidirectional adj. 双向的；双向作用的

3.1 **重复代码（Duplicated Code）**

坏味道行列中首当其冲的就是Duplicated Code，如果你在一个以上的地点看到相同的程序结构，那么肯定：设法将他们合而为一，程序会变得更好。

最单纯的Duplicated Code就是“同一个类的两个函数含有相同的表达式”。这时候你需要做的就是采用Extract Method 提炼出重复的代码，然后让这两个地点都调用被提炼出来的那一段代码。

另一种常见的情况就是“两个互为兄弟的子类内含有相同表达式”。避免这种情况，只需要对两个类都使用 Extract Method ， 然后再对被提炼出来的代码使用 Pull Up Method ，将它推入超类内。如果代码之间只是类似，并非完全相同，那么就运用Extract Method 将相似的部分和差异部分割开，构成单独的一个函数。然后你可能发现运用 From Template Method 获得一个 Template Method 设计模式。如果有些函数以不同的算法做相同的事情，你可以选择其中较清晰的一个，并使用 Substitute Algorithm 将其他函数的算法替换掉。

如果两个毫不相关的类出现 Duplicated Code,你应该考虑对其中一个使用 Extract Class，将重复代码提炼到一个独立的类中，然后在另一个类内使用这个新类。但是，重复代码所在的函数也可能的确只应该属于某个类，另一个类只能调用它，抑或这个函数可能属于第三个类，而另两个类应该引用这第三个类。你必须决定这个函数放在哪儿合适，并确保它被安置后就不会再在其他任何地方出现。

3.2 **过长的函数（Long Method）**

拥有短函数的对象会活的比较好，比较长，不熟悉面向对象的技术的人，常常觉得对象程序中只有无穷无尽的委托，根本没有进行任何计算。和此类程序共同生活数年之后，你才会知道，这些小小的函数有多大的价值，“间接层”所能带来的全部利益——解释能力、共享能力、选择能力——都是由小型函数支持的（间接层和重构）

很久以前程序员就已经认识到：程序越长越难理解。早期的编程语言中，子程序调用需要额外的开销，这使得人们不太乐意使用小函数。现代OO语言几乎已经完全免除了进程内的函数调用开销，不过代码阅读者还是得多费力气，因为他必须经常转换上下文去看看子程序做了什么。某些开发环境允许用户同时看到两个函数，这可以帮助你省去很多麻烦，但是让小函数容易理解的真正关键在于一个好名字。如果你能给函数取一个好名字，读者就可以通过名字了解函数的作用，根本不必去看其中写了些什么。

最终的效果是：你应该更积极的分解函数。我们遵循这样一条原则：每当感觉需要以注释来说明点什么的时候，我们就可把需要说明的东西写进一个独立的函数中，并以其用途（而非现实手法）命名。我们可以对一组甚至短短一行代码做这件事，哪怕替换后的函数调用动作比函数自身还长，只要函数名称能够解释其用途，我们也该毫不犹豫地这么做。关键不在于函数的长度，而在于函数“做什么”和“如何做”之间的语义距离。

百分之九十的场合里，要把函数变小，只需要使用Extract Method。找到函数中适合集中在一起的部分，将他们提炼出来形成一个新函数。

如果函数内有大量的参数和临时变量，他们会对你的函数提炼形成障碍。如果你尝试运用 Extract Method, 最终就会把许多参数和临时变量当做参数，传递给被提炼出来的新函数，导致可读性几乎没有任何提升。此时，你可以运用 replace temp with query 来消除这些临时变量。Introduce Parameter Object 和 Preserve Whole Object 则可以将过长的参数列变得更简洁一些。

如果你已经这么做了，仍然有太多临时变量和参数，那就应该使用我们的杀手锏： Replace Method with Method Object。

如果确定该提炼哪一段代码呢？一个很好的技巧是：寻找注解。他们通常能指出代码用途和现实手法之间的语义距离。如果代码前方有一行注释，就是在提示你：可以将这段代码替换成一个函数，而且可以在注释的基础上给这个函数命名。就算只有一行代码，如果它需要以注释来说明，那也值得将它提炼到独立的函数中去。

条件表达式和循环常常也是提炼信号。你可以使用 Decompase Condition 处理条件表达式。至于循环，你应该将循环和期内的代码提炼到一个独立的函数中。