11个小技巧让你写的代码可读性更好！

2016-09-05 实验楼

有很多理由都能说明为什么我们应该写出清晰、可读性好的程序。最重要的一点，程序你只写一次，但以后会无数次的阅读。

当你第二天回头来看你的代码时，你就要开始阅读它了。当你把代码拿给其他人看时，他必须阅读你的代码。

因此，在编写时多花一点时间，你会在阅读它时节省大量的时间。

让我们看一些基本的编程技巧：

1. 尽量保持方法简短
2. 永远永远不要把同一个变量用于多个不同的目的
3. 使用自描述的变量名和方法名
4. 尽可能的把变量定义在靠近使用它的地方
5. 拒绝神秘数字
6. 友好的对待你的语言
7. 不要逆常规而行
8. 警惕过早优化
9. 积极重构测试过的程序
10. 不要过度沉迷于技巧
11. 通过习例学习新知

现在，让我们把每个小点展开来详细讲一下。

1.尽量保持方法简短

尽管很多人都遵循这个规则，但它仍然非常的重要。你写的方法要始终能在一个屏幕里放得下。如果你需要去滚动屏幕，这会分散你的注意力，而且你看不到整个的上下文。最佳长度是5-20行，这根据你的情况而定。

当然，getters/setters 通常是一行代码的方法，但与其说它们是真正的方法，不如说它们只是存取工具。

2. 永远永远不要把同一个变量用于多个不同的目的

一个变量应该始终只为一个目的服务。

通过使变量常量化(C++里的const， Java里的final)，使得编译器能够优化编译，而且使你的代码醒目表达这个变量是不能改变的，你的程序的可读性会变得更好。

3. 使用自描述的变量名和方法名

你的代码应该，对于任何人来说，只要看一眼就能知道是干嘛的。尽量不要用简写方式，除非有特殊的习惯，就像下面的：

```
src - source
pos - position
prev - previous
```

如果你认为描述性的名称并不是那么有价值，请对比一下n, ns, nsisd 和numTeamMembers, seatCount, numSeatsInStadium。

4. 尽可能的把变量定义在靠近使用它的地方

盖房子时，你可不希望把锤子放到别人的院子里。你希望把它们放的离手头越近越好。定义变量也是同样的道理。

```
int foo = 3;
int bar = 5;
// 一大段使用“bar”的代码，
// 但没用到“foo”
// ...

baz(foo);
```

这段代码可以简单的重构成

```
int bar = 5;
// 一大段使用“bar”的代码，
// 但没用到“foo”
// ...

int foo = 3;
baz(foo);
```

当你把变量的声明和第一次用到它的地方间隔太远时(距离超过一个屏幕)，这确实会成为一个问题。记住上下文关系会变得困难，你需要滚动屏幕去找哪来的这个变量。

5. 拒绝神秘数字

当你要把什么东西跟一个常量值做比较时，记得把这个值定义成常量。没有什么会比去猜测你的同事写的这样的代码更让人头疼的事了：

```
il < 4384
```

换个形式感觉如何?

```
inputLength < MAX_INPUT_LENGTH
```

6. 友好的对待你的语言

学习新语言是一种很有乐趣的事情，你能学到一种新的完成任务的途径。当一个对一种语言已经很专业的人去学习另一种语言时，会出现一种很大的负面效应。

比如说你是一个Java开发者，试图去学习Ruby。你应该学会用Ruby的方式解决问题，而不是沿用Java的解决问题的思想。

当你需要重复5遍”Hello world!“时，在Java里，你可能会这样做：

```
for (int i = 0; i < 5; i++) {
    System.out.println("Hello world!");
}
```

在Ruby里，你也许会禁不住这样写：

```
for i in (0..5)
  puts "Hello world!"
```

```
end
```

这样看起来没问题，但有一个更好的方式：

```
5.times { puts "Hello world!" }
```

7. 不要逆常规而行

每种语言都有自己不同的习俗约定。一般来说，人们听的最多的是Java的编码规范。让我们看看其中的一些习俗规范：

- 方法名应该小写字母开头，其后用字母大写的单词连接(veryLongVariableName)
- 类名应该都使用首字母大写的单词连接而成
- 常量名应该全部大写，用下划线连接(MY_CONSTANT)
- 左大括号应该跟 if 语句在同一行

只有在有必要的理由时才去打破这些常规，不要轻易的因为你不高兴就违反它。

如果你只是在团队里改变一些这样的习惯，那也没问题，但当把你代码拿出来和其他的没有这些思想准备的程序员共享时，问题就会来了。

8. 警惕过早优化

过早优化是所有问题的根源，至少电视上是这么说的 ...

你第一应该关心的事情是写出易于理解的代码。起初写的程序不要求快。除非你的程序很慢，否则谈优化都是为时太早。

如果你想优化什么东西，你首先需要知道问题出在哪。这就是我们需要profilers这个工具的原因。

在没有知道问题在哪的情况下试图对程序进行优化，其结果必然是把程序能坏，至少你的代码会丧失可读性。

如果你觉得有些地方很慢，不要盲目的重写代码，你应先找到慢的证据。

不要傻乎乎的去解决根本不存在的问题。

9. 积极重构测试过的程序

没有任何东西会是完美的。即使你感觉你真正写出了一段完美的代码，几个月后回头再看看，你可能会惊讶道”怎么会这样傻？“

改进程序的一个好方法就是重构，但要等程序测试通过之后。你首先要确保程序是好的可运行的，你可以通过自动化测试或手工测试完成这个工作。

之初，你需要的是程序可用。不要期望在第一次就写出完美的程序，你只需要把它写出来，可用。然后重构它，使之完美。对于你们当中知道测试驱动开发(TDD)的人来说，对这个会很熟悉。

这里的关键就在于你要习惯于重构这种事情。如果你使用的是像IntelliJ IDEA这样强大的集成开发工具的话，重构的工作会变得简单的多。

重构之后，你也许会弄出一些Bug，导致某些功能出问题。这就是为什么说写自动化测试的原因。不论何时重构后，只要运行一下所有的测试用例，你就能准确的知道什么地方出了问题。

10. 不要过度沉迷于技巧

当我第一次读到有关设计模式的知识时，我觉得我找到了圣杯。

这些精心设计的思想作用显著，它能使你的设计易于理解，因为你可以简单的说”我使用的是‘观察器模式’“，而不用从头到尾的解释一遍。那么，有问题吗？一切看起来都这么自然、简单，你开始不论在哪都使用设计模式。为什么不把这个类做成singleton呢？干嘛不去再创建一些工厂类呢？

于是一个80行就能写完的脚本，你最终使用了10个类，15个接口，外加一大堆范式和标记符。97%的代码不做任何事情。

设计模式是一种十分有用的用来简化你的设计的工具，但这不意味着你该在所有能用到的地方都用它。你应该用它们，但不能滥用。

11. 通过习例学习新知

编程是一种学习新知的过程。当你学到了新的程序库或新语言，你可能会迫不及待的丢掉旧的代码，用你新学到的东西重新写一遍。有很多的理由都能说明你不该这么做。

往现有的应用里增加新的类库或框架同属于这种情况。就说你写了一个Javascript的web应用，期间，你发现了jQuery。现在你突然急切的想丢到你的Javascript程序，重新用jQuery写，尽管你还从来没用过它。

最好的方式是你先用jQuery写一些简单的例子，通过这种方式把你在应用里将要用到的知识都学会。需要AJAX？在你的项目之外做一些小例子，当完全弄懂了后，丢掉例子，应用到你的产品里。

如果你非常关注编程技术，我强烈的推荐你阅读Steve McConnell写的《代码大全》 一书。它会永远的改变你对编程的认识。:)

转载自：程序师

文章地址：http://www.techug.com/11-tips-for-better-code

英文原文：http://progfu.com/best-practices/11-tips-for-better-code/