[翻译]



**能源与环境学院**

**热能与动力工程**

**03013432**

**李建杰**

**2016.5.13**

将Guttag, John的《Introduction to Computation and Programming Using Python》中的：

6.2.3 When the Going Gets Tough

6.2.4 And When You Have Found “The” Bug

翻译为中文。

要求：

1. 翻译重在“意达”，翻译中能体现自己理解的“意”为佳；
2. 翻译以后根据自己编程体验，写体会。
3. 翻译文档内容：英文原文、翻译和编程体验

**目录**

**[一、翻译 3](#_Toc8615)**

**[6.2.3 When the Going Gets Tough 3](#_Toc9674)**

**[6.2.4 And When You Have Found “The” Bug 7](#_Toc17343)**

**[二、 感受 10](#_Toc15283)**

**一、翻译**

**6.2.3 When the Going Gets Tough**

Joseph P. Kennedy, father of President Kennedy, reputedly instructed his

children, “When the going gets tough, the tough get going.” But he never

debugged a piece of software. This subsection contains a few pragmatic hints about what do when the debugging gets tough.

肯尼迪的父亲在教导他们的孩子时说：“当过程变得艰难时，只有坚强者才能继续前行。”但他从来没有调试过程序，以下这篇描述将在你调试程序困难时找出问题的关键性提示。

• *Look for the usual suspects*. E.g., have you

找出常见的错误，比如你有没有

o Passed arguments to a function in the wrong order,

以错误的顺序给给函数传递参数

o Misspelled a name, e.g., typed a lowercase letter when you should

have typed an uppercase one,

拼错了代码单词,例如应该大写的单词你将它小写了

o Failed to reinitialize a variable,

初始化变量失败

o Tested that two floating point values are equal (==) instead of

nearly equal (remember that floating point arithmetic is not the

same as the arithmetic you learned in school),

检验两个浮点数是否相等而不是近似相等（注意浮点的计算和学校里学习到的是不一样的）

o Tested for value equality (e.g., compared two lists by writing the

expression L1 == L2) when you meant object equality (e.g.,

id(L1) == id(L2)),

比较两个对象是否相等（例如你想比较L1与L2对象是否相等，你在代码中应写作id（L1）==id（L2））

o Forgotten that some built-in function has a side effect,

忘记了一些内置函数对外界的影响

o Forgotten the () that turns a reference to an object of type

function into a function invocation,

忘记了（）会将函数类型对象的引用转换成函数的调用

o Created an unintentional alias, or

创建了一个没有定义的对象

o Made any other mistake that is typical for you.

犯了一些你经常犯的错误

• *Stop asking yourself why the program isn’t doing what you want it to.*

*Instead, ask yourself why it is doing what it is.* That should be an easier

question to answer, and will probably be a good first step in figuring out

how to fix the program.

当程序没有按照你预想的方向运行时，不要去问他人。相反你应该 问问自己它为什么会这样工作。这个问题回答起来应该很简单，并 且是你找出完善程序方法的第一步。

• *Keep in mind that the bug is probably not where you think it is.* If it were,you would probably have found it long ago. One practical way to go

about deciding where to look is asking where the bug cannot be. As

Sherlock Holmes said, “Eliminate all other factors, and the one which

remains must be the truth.”

记住问题很可能不是出在你认为的地方，如果是，那你很早以前就 应该发现它了。一个实用的方法是判断问题不可能出现在那些地方。 正如夏洛克·福尔摩斯说，“排除其他的因素，那么剩下来的一定是真相。”

• *Try to explain the problem to somebody else.* We all develop blind spots.

It is often the case that merely attempting to explain the problem to

someone will lead you to see things you have missed. A good thing to try

to explain is why the bug cannot be in certain places.

试着去解答一些他人的问题。我们都有盲点，当试着解答别人的问 题是才会让你看到自己漏掉的点。一个好的方法是解释为什么错误 不会出现在这个地方？

• *Don’t believe everything you read.* In particular, don’t believe the

documentation. The code may not be doing what the comments suggest.

不要全部相信你所读到的东西。特别是文献资料。程序很可能不是按照文献所说的情况来运行的。

• *Stop debugging and start writing documentation.* This will help you

approach the problem from a different perspective.

停止调试程序而去做一下思想记录。这会帮助你以一个不同的思维方式接触问题。

• *Walk away, and try again tomorrow*. This may mean that bug is fixed

**later** in time **than** if you had stuck with it, but you will probably spend a

lot **less** of your time looking for it. That is, it is possible to trade latency

for efficiency. (Students, this is an excellent reason to start work on

programming problem sets earlier rather than later!)

随他去吧，明儿再做。这意味着改正错误的时间肯定是比你深陷程序故障不能运行的时间要长，但这样你就不用花很多时间来找出程序中问题出现在哪里。很可能延迟和效率可以等价交换。（同学们，这很好解释了在解决程序调试问题时赶早不赶晚）

**6.2.4 And When You Have Found “The” Bug**

When you think you have found a bug in your code, the temptation to start coding and testing a fix is almost irresistible. It is often better, however, to slow down a little. Remember that the goal is not to fix one bug, but to move rapidly and efficiently towards a bug-free program.

当你在代码中发现了错误是，那种立刻纠正错误并改写代码的冲动几乎是不可抵抗的。但最好是先冷静下来，永远记住你的目标不是修正一个错误，而是让程序向着更快速，更高效，没有错误的方向前进。

Ask yourself if this bug explains all the observed symptoms, or whether it is just the tip of the iceberg. If the latter, it may be better to think about taking care of this bug in concert with other changes. Suppose, for example, that you have discovered that the bug is the result of having accidentally mutated a list. You could circumvent the problem locally (perhaps by making a copy of the list), or you could consider using a tuple instead of a list (since tuples are immutable),perhaps eliminating similar bugs elsewhere in the code.

问问你自己这个错误是否就是表面上的特征，或者是否这只是冰山一角。如果是后者，最好小心对待这个问题，看是否会带来其他的变化。假设一下，你发现错误导致改变了一个链表，你可以避开这个问题（比如复制一张链表），或者用元组代替链表（元组是不会变的），但这样也许在代码的其他地方类似的错误就会发生。

Before making any change, try and understand the ramification of the proposed “fix.” Will it break something else? Does it introduce excessive complexity?Does it offer the opportunity to tidy up other parts of the code?

Always make sure that you can get back to where you are. There is nothing

more frustrating **than** realizing that a long series of changes have left you

further from the goal than when you started, and having no way to get back to where you started. Disk space is usually plentiful. Use it to store old versions of your program.

在做任何改动之前，试着理解修改带来的衍生问题。它会破坏其他的东西吗？它会不会使程序过于复杂？是否有其他地方可以整理这些代码？永远确保你可以回到修改之前的地方。没有什么东西让你发现一系列的改动比你刚开始更加远离你所期望目标，并且不能返回到改动前更加令人沮丧了。磁盘的空间是充足的，试着将你的旧程序保存下来。

Finally, if there are many unexplained errors, you might consider whether

finding and fixing bugs **one at a time** is even the right approach. Maybe you

would be better off thinking about whether there is some better way to organize your program or some simpler algorithm that will be easier to implement

Correctly.

最后，世上有很多解释不了的错误。你可能会考虑一次找出并解决一个错误是否是正确的方法，如果你时常思考是否有更好的方法整合你的程序或者更简单正确的算法试用于你的程序，那样，你会变得更好。

# 感受

1. 你以为你以为不是你以为的。错误常常出现在你没有以为的地方，而你以为的地方，其实一点问题都没有
2. 遇到问题就解决，不行明天再来，不要pass it away，否则你将会在调试程序的时候付出十倍以上的时间。
3. 多去帮他人解决问题，这样比你自己编程序要学到的多，因为他人的问题恰恰是你忽略的点，只是你没有注意到。