作业四 谢云飞 03013436

翻译

6.2.3 When the Going Gets Tough

当程序进展变得困难

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.

据说肯尼迪总统的父亲约瑟夫·帕特里克·肯尼迪教育他的

孩子,“ 当事情变得艰难的时候 , 坚强者才能继续行路。” 但他从未调试过软件( 但这句话这对于软件也是成立的)。 本节包含了一些关于你在调试程序的时候出现困难的时候应该这么做给出了有实际意义的提示。

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

•寻找常见的错误。比如，你是否

o Passed arguments to a function in the wrong order,

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

have typed an uppercase one,

o Failed to reinitialize a variable,

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),

o 函数参数传递时弄错了顺序,

o 拼错名字 , 比如。你应该输入一个小写字母时却输入一个大写字母,

o 重新初始化一个变量失败

o 测试两个浮点值是否相等而不是近似相等 ( 记住, 浮点运算不是

和你在学校中所学到的算术不一样),

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)),

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

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

function into a function invocation,

o Created an unintentional alias, or

o Made any other mistake that is typical for you.

当你想要两个对象相等（比如id(L1) == id(L2)），检查他们的值是否相等o（比如通过两列写入的表达式id(L1) == id(L2) 来比较

o 忘了一些内置函数有副作用,

o 忘记加(), 这样一个类型的对象的引用会变成一个函数的调用,

o 创建了一个未定义的别名,或

o 犯其他你典型犯的错误。

•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.

不要一直问自己为什么你的程序不想你想的那样运行. 取而代之的是,问问自己为什么它这么运行。那应该是一个更容易回答的问题, 而且这可能会在找出如何修复程序迈出很好的第一步。

• 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.”37

请记住错误可能不会出现在你认为它在哪里的地方。如果真的在那里, 你可能很久以前就会发现它。 一个实用的方法来决定去哪里查看就是问问自己错误最不可能出现在哪。正如福尔摩斯说:“ 消除所有其他因素, 剩下的那个就是真相。”

• 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.

• Don’t believe everything you read. In particular, don’t believe the documentation. The code may not be doing what the comments suggest.

• Stop debugging and start writing documentation. This will help you approach the problem from a different perspective.

•尝试向别人阐述你遇到的问题。我们都有盲点。通常情况下,仅仅尝试阐述这个问题可能就有人引领你找到你没有发现的东西。一个好的方法就是去尝试解释为什么错误不会发生在特定的地方。

•不要相信所有你阅读的东西。特别的,不要相信文档。你的代码可能不会像注释那样建议的运行。

•停止调试并开始编写文档。这将帮助你从不同的角度看问题。

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!)

离开, 明天再试试。这可能意味着错误解决的时间比你一直坚持去找来得晚, 但你可能会花更少的时间去解决它。也就是说, 它就好比是用时间的延迟来换取效率的交易。( 同学们，这是一个很好的早一些而非晚些去开始程序设计课程练习题的理由！)

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.

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.

6.2.4当你发现那个错误的时候

当你认为你在代码中发现了一个错误的时候, 那种想要开始编程和测试修复的冲动与诱惑几乎是不可抗拒的。然而, 通常更好的方式是放缓下来一点。记住, 我们的目标不是修复一个错误, 而是要更快更有效率地的向一个没有错误的程序进发。

问问你自己是否这个错误能够解释所有观察到的错误现象, 还是它仅仅是冰山的一角。如果是后者, 那么你最好考虑要注意到这个错误和其他变化的同步与协调。假设，例如你有发现的错误是由于不小心改变了一个列表。你可以在本地智取这个问题( 也许通过做一份列表的拷贝), 还有你可以考虑使用一个元组来代替一个列表( 因为元组是不可变的), 也许同时可以消除代码中的其他地方的类似错误。

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?

做任何改变之前, 试着去想明白你所计划的修复可能衍生出的其他分支问题。想想它会不会破坏其他的地方？会不会引入过多复杂的程序? 它有没有提供整理代码其他部分的机会?

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.

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.

你一直要确保你可以回到你修改之前。没有什么比意识到一长串的修改已经使你远离目标但却没有办法回到一开始的地方这件事更令人沮丧了。磁盘空间通常是足够的。用它来储存你程序的旧版本。

最后,如果有许多无法解释的错误,你应该想想逐个找到并修改错误这个方法是否正确。或许你最好要想想是否有更好的方法来组织程序或者是否有一些更轻松并能更容易地实现程序的算法。