**原文&译文**

### 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.

6.2.3当过程开始变得艰难

据说，肯尼迪总统的父亲Joseph P. Kennedy是这样教育他的孩子的：“当情况变得艰苦时,只有吃苦耐劳的人能继续前进。”然而他还未曾解决过一个小小的软件bug。关于在排除程序故障时遇到困难时的应对办法，本小节提供了几条实用的提示。

Look for the usual suspects. E.g., have you不妨看看一些常见的猜测。例如：你是否

* Passed arguments to a function in the wrong order将参数通过错误的指令传递给了函数
* Misspelled a name, e.g., typed a lowercase letter when you should have typed an uppercase one拼错了函数名，如：将本该大写的字母打成小写
* Failed to reinitialize a variable没有进行变量初始化
* 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)检验两个浮点数值相等而不是近似相等（要记住，浮点算法和你在学校学习的算法是不一样的）
* 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))在想要表述对象相等（例如 id(L1)==id(L2)）时写为值相等（例如在对比两个列表时写下L1==L2）
* Forgotten that some built-in function has a side effect忘记了某些嵌入功能具有副作用
* Forgotten the () that turns a reference to an object of type function into a function invocation忘记了（）能够将一个函数类型对象的引用转至函数调用中
* Created an unintentional alias, or创建了一个无意的别名，或者
* Made any other mistake that is typical for you.犯了一些对你而言十分典型的错误
* 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.别再抱怨为什么你的程序做不成你想要它做的事情，相反地问问你自己，它为什么能够完成那些可以完成的任务。回答这个问题应该相对简单一些，同时它可能会是修复程序的很不错的第一步。
* 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.”时刻提醒自己，bug很有可能出在意想不到的地方。如果它出在你能想到的地方，那你早就发现它了。在决定从何处寻找bug时一个很实用的方法是，问问自己bug没可能出在哪里。就像卷福所说的，“排除所有干扰因素，最后剩下的那个必定是真相。”
* 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.试着向他人说明你的问题。每个人都会有自己的盲区。事实证明，经常尝试将自己的问题向他人说明，将会让你注意到之前遗漏的东西。在解释说明过程中不妨说说你为什么认为bug不可能出现在某些地方
* Don’t believe everything you read. In particular, don’t believe the documentation. The code may not be doing what the comments suggest.别相信你读到的所有东西，尤其是参考文件。你的代码或许并不会同它所建议的那样运行
* Stop debugging and start writing documentation. This will help you approach the problem from a different perspective.停止故障排除，转而开始写未见材料。这将从其他角度帮助你接近问题的解决。
* 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!)放下它，明天再试着解决。这可能意味着，你将会比埋头死磕它要稍晚一些解决它，但你或许可以花费少得多的时间去寻找问题所在。也就是说，将时间的延迟转变为效率的提高是完全有可能的。（同学们，这是你们尽早开始着手解决问题，而不是一拖再拖的极佳理由！）

### 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. 当你认为你已经发现了代码中的一个bug，心中那种想要开始改写和调试的诱惑已阻挡不了了。然而，脚步稍微放慢些反而会有更好的效果。要记住，你的目标不是处理好一个简单的bug，而是完成一个运行快速，高效，没有漏洞的程序

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. 问问自己，这一个漏洞是否能够说明程序所出现的所有病症，又或者这只是冰山一角。如果是后者，或许该注意在处理这个bug的时候顾及到你的其它改动。比如，你找到一个将表单意外无效化的漏洞，你可以局部性地规避它（或许通过拷贝一份表单），或者你可以考虑使用元组来替代表单（因为元组是无法被沉默的），这可能消除代码其它地方相似的错误。

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?在进行任何改动之前，尝试并理解修复计划的衍生作用。它会不会影响其它代码的运行？它会不会引入过多的复杂性？它会不会提供一个整合代码其它部分的机会？

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. 永远要确保你可以回到你改变前的状态。当你意识到你所做的一系列冗长的修改使你比最初更加原理成功，并且你在也无法回到原先开始的地方时，没有什么比这更让人绝望的了。磁盘空间总是很充裕的，要时常用它来存储你先前的程序。

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，如果有太多的无法说明的错误，你或许该思考一个一个地修复bug是否能够让你更加接近目标。或许该想想，也许有更好的构建程序的方式，或是更为简单的算法，能够使程序成功运行

## 个人体会

在曾经编写程序的过程中，我也遇到过许多文中提及的类似的问题，然而没能即使把它们整理出来，也没有系统去思考解决方法。读过材料中的内容后顿时感觉，那些原本隐隐有些体会的东西瞬间清晰明朗了许多。

当你费尽心思，花掉大量时间和精力，做出一个程序后，试运行时发现有不下几十个error时，那种令人抓狂的心情相信许多人也同我一样深有体会。一般这种时候，我都会直接跳到文中最后一个hint：明天再说！毕竟抱着不好的情绪去修复bug是十分低效的。倒不如等第二天养好精神，再慢慢处理。另外，文中提及的关于如何找到bug所在的方法我个人觉得也十分普适，基本涵盖了所有重要的环节。

一个好的程序员，应该在入门初始便养成良好的编程习惯，对细节的把握往往是能提高效率的关键因素。

其实，文中许多解决问题的办法，不仅在编程过程中适用，而且在解决其它问题时也同样具有参考意义。比如“尝试着把你遇到的问题向他人说明”这一点。平时做作业遇到不会的问题，在问别人的时候，往往就会自己突然茅塞顿开，然后向别人连声道谢，人家一脸哭笑不得的表情。世事都是相同的，相信当我们能够出色地完成程序设计后，也必定掌握了解决其它许多事情的能力。