Wang ChenLu

[公司名称][公司地址]

作业四 翻译及感想

# 一、原文

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.”( 36 He also reputedly told JFK, “Don't buy a single vote more than necessary. I'll be damned if I'm going to pay for a landslide.”)But he never debugged a piece of software. This subsection contains a few pragmatic hints about what do when the debugging gets tough.

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

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 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.

• 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.

• 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 ArthurConan Doyle, “The Sign of the Four.”)

• 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.

• 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!)

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.

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.

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.

# 二、翻译

6.2.3 当生活不顺利的时候

据说，肯尼迪总统的父亲，约瑟夫·P·肯尼迪这样教育他的孩子：“当生活不顺心的时候，只有意志坚强的人能够前进。”（36他也是这样告诉JFK的，“一张多余的选票都不要买，如果我要通过金钱堆造出压倒性的胜利，我会被惩罚的。”）但是他从来不修改软件程序上的错误。下面这个部分包含了许多当调试程序不顺利时能帮到你的一些实质性建议。

* 寻找一些经常性的错误。例如，你有没有：

传递函数变量的时候顺序出错

拼错单词，比如，应该输入大写字母的时候输入了小写字母。

重新初始化变量失败

检查两个浮点值是否相等（==）而不是近似相等。（记得浮点变量的计算法则和你在学校学习的是不一样的）

当你想让两个对象相等时（例如，id（L1）=id（L2）），检查他们的值是否相等（例如，通过写出两列表达式L1==L2来进行比较）

忘记一些内置函数具有一些副作用

忘记（）会将一个对函数类型对象的引用转化成堆函数的调用

创建一些没有定义过的对象。

犯一些你常犯的错误

* 不要再纠结为什么程序不按照你希望的那样运行。反过来，问问自己为什么它会这样做。那会是一个更容易回答的问题，可能会是发现如何修复问题的最好的第一步。
* 记住问题有可能不在你认为的地方。如果在，你可能早就发现了。一个最实际的决定检查哪里的方法是问问题有可能出在哪里。就像夏洛克福尔摩斯说的那样，“排除掉其他的可能，剩下的那个必然是真相”（37亚瑟王柯南道尔，《四签名》）
* 尝试着把问题解释给别人听。我们都有盲点。向别人解释问题的过程会让你发现一些遗漏掉的东西。一种解释为什么问题不在你认为的地方的最好方法。
* 不要相信你读到的所有东西。尤其是不要相信文献资料。你的代码有可能不按照它的意见运行。
* 停止调试并且开始写下标注。这会让你从另外一个角度看问题
* 关掉，明天再试。这可能意味着问题修复的时间上会比一直纠结的时间上晚一点，但你会花费更少的时间去解决。这就是，用延迟去换取效率。（同学们，这就是一个尽早开始做程序设计练习的最好的理由！）

6.2.4然后当你找到那个问题了

当你认为你在你的代码中找到了一个问题，那种立刻开始编写代码进行修复和测试的欲望是不可抵挡的。然而，更好的做法是放慢速度。记住目标不是修复这一个问题，而是快速高效地朝着没有问题的程序前进。

问问自己这个问题是否能够解释所有显示出来的结果，或者它只是冰山一角。如果是后者，要注意修改这个问题时和他相关的地方要做出相应的调整。设想，比如，你发现问题是偶然改变了一个列表导致的，你可以绕过这个问题（通过复制列表），或者你可以考虑用元组代替这个列表（因为元组是不可改变的），可能会在代码里其他地方排除相似的问题。

在做出修改之前，试着去理解设想的修复动作会带来的一系列的变化。它会不会破坏其他部分？它会不会使程序变得复杂？它能不能够使代码的其他部分简化？

经常确认你能够返回你原来的版本。没有什么比意识到一大串代码的替换已经让你离你的目标比开始的时候还远却没有办法回到最初更让人沮丧的了。硬盘空间一般都会很充足。要记得存储你项目的原始版本。

最后，如果有很多没办法解释的错误，你应该考虑一个一个地查找修复问题是不是正确的方法。也许你更应该想想是不是有其他更好的方式去编写程序或者其他更加简单的且能够正确实现的算法。

# 三、心得体会

刚开始听说编程其实我的内心是拒绝的。我的编程经验也只有大一时候简单的C++的学习，和这学期接触不深的单片机。在大一学习的过程中，也接触了一些专业术语，慢慢的从抽象，不理解开始一点点弄懂，最后达到了足以应付考试的程度。当时也曾经因为一些字母大小写，函数拼写错误，还有出现未定义变量等等小错误纠结到深夜，一遍一遍地找错结果还是失望的心情完全可以理解。所以在做这篇文章翻译地过程中引发出我很多的共鸣，也有点欣慰，因为那些错误并不是我一个人有，大家可能都会出现，想到这里心情就变得轻快起来。文章里说到的那些问题的解决方法我觉得是很正确的，尤其是当找不出错误的时候先放下，明天再看。这点我非常地赞同。可能一段程序纠结了好久，可是你反复地检查，其实就是一直在遁着之前的思路想法在看，根本不会有任何效果。不如先放下，让大脑放空，忘记之前的思路，以一个局外人的视角去审视这段程序，重新开辟新的解读思维，会更加容易跳出旧的思维模式，发现其中逻辑或者语法上的错误。

许多计算机专业的同学都说以后做了程序员头发会掉光。我觉得说的没错。编程重要的一点就是逻辑加细心。在调试的过程中就是不断地在质疑和否定自己，这是费神费力的一项工作。所以我们专业虽然看起来不要求多高的编程能力，但在学习的过程中也锻炼了我们的逻辑能力，也为以后的多元化发展打好基础。就像文章里那句话，编程习题要尽早开始练习，熟能生巧，练习的多了，自然就信手拈来，读程序的能力也自然得到了巩固和提升。

总的来说，我们专业虽然看起来和编程没什么关系，但从未来发展的角度看，即为我们的发展增添了一种可能，又可以锻炼我们学习能力和逻辑思维，一石二鸟何乐而不为？