神奇的调试艺术

 真正精通调试技巧的人是非常少的，这是我在多年企业系统管理的工作经验中所了解。如果说调试很简单并且有很多的人都在做，而且每个人都能够找到错误。然而现实情况是，大多数商店都有一个被称为“灭虫者”的“亲密接触者”，被请来寻找其他无法追踪的错误。我曾担任过产品经理和咨询经理，在任职期间，我们工作内容很大一部分都应付查找这些让人讨厌的错误并对其进行修复。

 我相信故障的排除是可以被传授，学习和掌握的技能。不幸的是，太多的人专注于工具和开发语言功能，而不是一个除去语言，平台或工具后，如何都会起作用的思维框架。有兴趣了解动态解决代码的秘密吗？请继续阅读！

 我从事Bug修复已有几十年了。我发表的第一篇专题文章其中之一就是一篇名为“ The Exterminator's Code”的故障排除文章，该文章在1999年的《 News / 400》杂志上刊登。我发现有效的Bug搜寻涉及多种技能。仅仅了解技术还不够。高效是有方法的。一些过程步骤是能够被学习，而且随着您在职业生涯中遇到更多系统，经验会不断增加。

总是令我惊讶的是，善于发现缺陷的人与那些不善发现缺陷的人之间的差距。您可能认为这将是一个连续的技能，但是我发现人们有些得到，有些没有得到。那些得到这些技术的人做事非常速度，且始终如一。那这些秘密是什么？

# 测试你的眼睛👀

 帮我个忙，做个小测验。阅读下面的文字，快速计算段落中“F”的数量。

 **Finished files are the result**

**of years of scientific study**

**combined with the experience**

**of years.**

 我一会儿再回答这个问题。如果我把它放在那太容易了。记下您的想法，然后再进行一些更深入讨论。这是另一套方法，请相信我，这一切都可以解决。您准备好参加另一场比赛了吗？

 我希望您观看一段很短的视频。该视频有一些说明。只要让它播放一次即可。不要试图暂停或快进。[这是链接，](https://youtu.be/vJG698U2Mvo)或者您可以按下面的视频播放。继续观看，然后写下分数（您会明白的）。

 到目前为止，我希望您开始明白我的想法，这是掌握调试技术的第一步。以我的经验，大多数开发人员调试代码的方式不对。当他们按下F5键并开始单步执​​行程序时，他们并没有注意到发生了什么事情。

 什么？我在开玩笑吗？他们设置了断点。有监视窗口。他们尽职尽责地敲击F10和F11以进入和退出子程序。我什么意思？这是正问题所在：

 他们正在等待程序执行他们期望的结果。而且并不难不，尤其是当您是编写程序的人时！因此当您逐步执行该代码块时，“是的，是的，我只是在这里初始化一些变量”然后快速按F10键，那您就错过了它，因为一个字符串文字拼写错误或引用了错误的常量。

“ F”测验的答案是6。大多数人算出是3，是因为他们在脑海中念出单词并聆听“ F”音，而不仅仅是看字母。这就是人们在凭着他感觉在调试时程序，而不是观察程序的真正运行。

# 认真：训练你的眼睛👁👁

 你看到大猩猩了吗？大多数人不会第一次。因为他们是按照指示行事的。他们正在数通行证，这正是演习的目的。但是你能相信当你看到它并且知道你在寻找什么时它是多么明显吗？你怎么会错过这样的事情？

希望到现在为止，我们已经确定你的大脑有一个很好的过滤器，并将尝试给你你想要的。所以，当你带着期望逐步完成代码时，你猜怎么着？你会看到调试器做了你期望的事情，而忽略了可能导致错误的真正发生的事情。

**但要怎么做呢？¯\(ツ)/¯**

 有几件事你可以做，以帮助磨练你的调试技能，我鼓励你尝试所有这些。

 请其他人调试您的代码，并提供调试他们的代码。了解如何查看代码并了解它在做什么的最好方法是逐步浏览您不熟悉的代码。一开始它可能看起来很乏味，但它是一种纪律和技能，可以帮助您学习如何以正确的方式遍历代码，而不做任何假设。

 尽量不要把代码当作块。换言之，当你有一个初始化变量的例程时，不要把它当作“初始化块”来单步执行，而要考虑每个语句。不要把这些语句看作句子，而是回到编程的根上，在等号的左边看到一组符号，在等号的右边看到一组符号。你会惊奇地发现这能帮助你快速适应错误或重复的作业。在MVVM中很常见，例如，对于开发人员来说，剪切粘贴并最终生成如下代码：

 private string \_lastName;

private string \_firstName;

public string FirstName

{

 get { return \_firstName; }

 set { \_firstName = value; RaisePropertyChanged(()=>FirstName); }

}

public string LastName

{

 get { return \_firstName; }

 set { \_lastName = value; RaisePropertyChanged(()=>LastName); }

}

你发现BUG了吗？如果没有，花点时间你会找到的。但当你编写的代码是这样的时候，这个就显得要困难得多了，因为你希望它“正常工作”。

# 回归基本🔎

 有了这些精美工具可以告诉我们如何重构代码和扫描类，有时我们忘记了必要进行故障排除的基本工具。
 我正在参与一个客户的一起对内存泄漏问题进行故障排除，发现自己开始于庞大的依赖关系，句柄和实例图。我可以看到某些对象被创建了太多次，但是看一下代码，它看起来就正确了。其他东西从哪里来？

所以，我回到了根本。我在构造函数中放入了一条调试语句，然后再次运行它。突然，我意识到有些实例正在如实地报告自己，而另一些则没有。这到底是怎么回事？啊 ...该类是从基类派生的。因此，我在基类中放置了另一个调试语句。果然，它也被实例化了。快速转储调用堆栈并解决了问题……不是通过图表和重构工具，而是良好的老侦探工作。

# 不要相信文档🔐

 我从早年学到的一个教训是不要相信文档。我正在写一本关于正在开发的新框架的书。该文档稀疏，多变，并且经常出错错误。在一个特定的部分中，我正在写某些应用程序应该如何工作。该文档非常具体地说明了如何将工作划分为多个线程。我的导师说：“不要相信它”，并鼓励我去调试。在逐步检查代码并观察实际发生的情况并不是预期的发生结果，我了解到架构是非常不同的。我能够帮助修复文档，并帮助开发人员避免不必要的代码开销。

我经常做一些小型项目来学习语言和平台。例如，考虑以下JavaScript：

const doSomething = (payload, fn) => { fn(payload); };

doSomething('This should echo', console.log);

let text = 'Some text';

doSomething(text, text => text += ' appended to.');

console.log(`The text after the call: ${text}`);

let textPayload = { text };

doSomething(textPayload, payload => payload.text += ' appended to.');

console.log(`The text after the payload call: ${textPayload.text}`);

 您可以预测控制台的输出内容吗？您可以通过运行[jsFiddle](https://jsfiddle.net/jeremylikness/3jnnv1sa/)来验证您的答案。

这是一段简单的代码，但是帮助我超越了概念化的范围，和对象在JavaScript中理解差异，从而亲身体验了它们的实际作用。

# 熟能生巧🏌️‍♂️🏌️‍♀️

 我经常调试正常的代码。我经常发现潜在未立即显示的问题，我可能会注意到初始化代码被多次调用（这可能会导致大规模的性能问题），或者某个事件被超出范围的实例注册而没有注销。这可能导致内存泄漏！

有时，甚至“简单的代码”也可能会超出自己的假设而泄露秘密。我从事了Angular.js编写的一些最大的项目（是的，需要$scope或controller as）。一些使用数据绑定的非常简单的页面在“后台”进行大量的工作。通过调试这些页面，我发现效率低下，例如运行多个“摘要循环”，这些随着时间的推移会影响性能。

 一个很棒的练习是下载一个开源项目，最好是一个实用程序或工具。花些时间查看源代码，以确定您认为代码的将如何运行。然后，启动调试器并逐步执行每个语句。千方百计！您可能会对发现和学习到的东西感到惊讶。您执行的次数越多，您就越有准备。你会：

1. 了解如何通过预览源代码来分析代码
2. 发现开发人员用来解决各种问题的模式
3. 潜在地发现作者尚未意识到的问题

此练习可能只发现了一个改进的地方，并作为开源贡献提交给项目。

# 最好的调试工具就是您的想法🧠

最后，我将会给您提供许多年前导师给我的同样建议，这是我多年前开始排除第一个企业问题时所给的。他告诉我，目标应该是永远不必启动调试器。每个调试会话都应该从逻辑上演练代码开始。您应该分析一下您期望的结果，然后实际地进行遍历... 如果我在这里通过了，这里我就理解，然后再去那里，这样就可以循环 ...帮助您理解代码。在我浏览源代码的十个错误中，有九个是错误的，而不必使用F5。

当我按F5键时，现在我对代码应该执行的操作有一个预期。当它做一些不同的事情时，通常更容易找出计划哪里出了问题以及执行代码如何脱离脚本。在许多不允许您完全运行调试器的生产环境中，此技能尤其重要。自那时以来，我一直受教并遵循这样的理念：源代码、位置良好的跟踪语句和深入思考的组合是修复最丑陋的错误所需要的全部。

 启动调试器并在您认为可能存在缺陷的位置设置断点是很容易的事。在紧要关头，这可能是最好的方法，如果您成功了，给你冒泡提示您！如果没有，请不要陷入相信您的假设的陷阱。相反，请退后一步，开始分析实际发生的情况，而不是您认为应该发生的情况。

我的步骤通常如下所示：

1. 尝试修复我认为的问题（如果我可以编写一个因缺陷而失败并在修复时通过的单元测试，则可加分）
2. 如果不是，请退后一步并分析代码以确定其实际作用
3. 打开调试器，但是不要在我认为可能是问题的地方设置断点，而是从头开始逐步进行操作，并注意发生了什么，而不是我认为应该发生的事情
4. 添加调试语句以帮助留下痕迹
5. 孤立地提取代码，以查看是否可以找到缺陷而没有完整解决方案的复杂性和开销

我已经做了足够长的时间，以至于我80％的时间都在第一步中找到了解决方案，最终在20％的时间中为最后几步花费了额外的时间。

# 这是一门你可以学到的艺术

 调试是一门需要耐心，专注和经验才能学习的艺术。我希望前面的练习可以帮助您理解那些有时会阻止您修复代码的筛选器，并且这些技巧将帮助您下次遇到问题时有所不同。请记住，没有不可修复的缺陷，而且没有比您的耳朵更强大的调试器。

您最喜欢的调试技巧是什么？请使用以下评论分享。