**JS10个调试和排错**

**摘要：**计算机编程可以说是一个要求较高的手艺，掌握这门手艺就得具备勤奋和不断学习的条件，因为新的事物和新的方法总是层出不群，所以不断地学习、提升技能是必不可少的。希望下面10个排错bug的工具对你的工作有用。

在空白的文本编辑器里打开一个崭新的文本，没有一行代码，出现在眼前的是一个充满了无限可能和希望的项目。可是，当数千行的代码写完之后， 整个项目因为bug的出现而被压垮了，更别说添加什么新功能了...这也许是对程序员的最大打击，在饱满的热情上浇了一盆冷水。其实，最好的软件程序员当 然知道怎样去发现并修复这些bug，在刚开始编程的时候就通过软件工程的最好方法来降低bug的出现概率。

几乎没有哪个程序员能够写出[一个bug都没有的代码](http://www.onextrapixel.com/2011/01/20/10-principles-for-keeping-your-programming-code-clean/)，但是解决方法总是比困难多得多。多实践和坚毅的决心是成功的关键，这样才能够写出清洁代码，保证软件系统的可靠性。

[](http://cms.csdnimg.cn/article/201308/14/520b29117a06c.jpg)

下面一起来看看这些可以镇压bug的工具箱。

**1. 输出语句**

代码调试的首要工具就是插入可靠地、真实的输出语句。当输出语句数量庞大且不易于管理的时候，在输出语句里恰当使用记录系统，这可以说是一个等效的好方案。许多编程语言里都配备了现成的类库，例如在Python里构建的记录库。

输出语句是程序员检查数据值和变量类型最快、最简单和最直接的方式。高效的输出语句能够帮助程序员通过一段代码来跟踪数据流，并快速识别bug源头。虽然先进的调试工具有很多，但是如果你想调试一段代码的话，这个普通的输出语句的方法应该是程序员最先考虑的方法。

**2. 调试器**

源代码调试器采用了输出语言方法里的逻辑推理。这样可以让程序员一行一行的单步执行代码，同时监测从变量值到底层虚拟机整个状态的一举一动。另外，大部分的编程语言都具有多个调试器，可以提供不同的功能，包括图形接口、终止程序的断点设置、执行环境内部任意代码的实施。

在许多情况下，调试器可以说是大材小用了，但如果合理利用的话，调试器绝对是一款高效率的工具。更多调试器的功能请看Python调试器：[pdb](http://docs.python.org/2/library/pdb.html" \t "_blank)。

**3. Bug跟踪系统**

在一些比较重大的软件项目里，使用bug跟踪系统是很有必要的。如果没使用bug跟踪器，最典型[](http://cms.csdnimg.cn/article/201308/15/520c8d1f0aa53.jpg)的状况就是程序员要整理以往的邮件或者是聊天记录来查找bug，更糟糕点儿的就是程序员根本不记得其它东西，印象里只有一点bug的文档。一旦这种情况发生，bug将必然充斥着整个代码编程，更加严重的是，想要识别出这些bug并确定它们的位置是很难的。

一 个简单的文本文件在项目里可以作为最初的bug跟踪系统。随着代码库的不断增加，bug衍生出一个文本文件并不需要太长的时间。有很多商业和开源的bug 跟踪软件提供的解决方案都是可以考虑的，选择哪一个bug跟踪软件首先要明确的部分就是要确保在编程项目里，那些非程序人员能够快速使用这个bug跟踪系 统。

**4. Linter**

在某些编程语言里，Linter可以执行对代码的静态分析，以便在代码编写和运行之前 识别出问题区域；在一些其它编程语言里，Linter工具对于语法检查和增强风格是很有帮助的。编程的时候在编辑器里打开一个Linter程序，或者是在 代码编写和运行之前通过Linter传递代码，这些都有利于程序员在使用软件之前发现并纠正更多的错误。因此，使用Linter可以帮你在节省宝贵时间的 同时揪出因语法错误、打字错误或数据类型错误而引起的bug源头。

想要知道什么样的Linter最适合你使用，看看Python的Linter工具：[Pyflakes](https://launchpad.net/pyflakes" \t "_blank)。

**5. 版本控制**

任何一个重大的软件工程项目里都不应该忽略使用版本控制系统。举例而言，像[Git](http://git-scm.com/)，[Mercurial](http://mercurial.selenic.com/" \t "_blank)和[SVN](http://subversion.apache.org/" \t "_blank)这类的版本控制允许不同的代码库版本在不同的基础上是可以分开的。

不同的控制版本可以被合并到一起，因此，多个程序员可以同一时间运行同一个代码库。版本控制在代码排错里同样有着举足轻重地位，可以让程序员回滚修改较早版本的代码，尽可能在错误出现之前，在代码库里对错误进行修复。

**6. 模块化**

缺少架构的代码是难以修复bug的主要源头。只要代码易于理解，而且理论上行得通，那么对于程序员来讲，找到并快速修复bug并不是什么棘手的事情。另一方面，越是重要的代码出现错误的几率就越大，找到这个错误相对也就比较困难。

设 计软件的组件经常需要考虑一点就是所谓的代码模块化，代码模块化可以帮助程序员更好的用两种方法来理解软件系统。第一，模块化能够创造出一定层次的抽象 感，在没有完全理解所有细节的情况下也能想象出系统的模型。比如，程序员正在构建一个商业系统，可能会考虑到信用卡处理模块，然后观察这个模块和其余代码 有什么联系，根本不用考虑信用卡处理模块的所有详细内容。第二，模块的详细说明，这个详细说明是不会和别的模块内容混淆的，就像每个卡只有一个卡号是一样 的。

**7. 自动化测试**

单元测试和其它类型的自动化测试跟模块化是有很大关联的，可以说是相辅相承。自动化测试就是一段代码用特殊的输入值来运行软件，以此来检测程序运行是否和预期的相符合。

单 元测试主要是用来检测单个功能的功能性，然而功能测试是用来检查特殊的程序性能，并且结合单元测试来检查软件系统的整体部分。有很多测试框架可以用来编写 测试程序，而且大部分受欢迎的测试框架都是由Kent Bent编写的JUnit类库衍生而来的，Kent Bent是“测试驱动开发方法”最早的支持者之一。 Python标准类库包括一个JUnit的Python版本，称之为PyUnit或者unittest的[单元测试框架](http://docs.python.org/2/library/unittest.html)。

**8. 泰迪熊方法（橡皮鸭调试）**

在软件编程界，就不得不提到传奇人物Brain Kernighan和Rob Pike，泰迪熊调试法源于一个大学计[](http://cms.csdnimg.cn/article/201308/15/520c8483f3e93.jpg)算 机中心，在这里，学生们遇到神秘bug的时候就可以先把问题解释给这只摆在桌子上的泰迪熊听，然后才能向老师或助教求助。所以，有的时候只跟熊聊天也能解 决问题。这一调试方法真的很管用，以至于风靡了整个软件工程行业，就像打印语句这一方，不管那些复杂的工具如何风起云涌，输出语句这一方法仍然在今天很受 欢迎。

同泰迪熊调试法相似的一种方法叫做橡皮鸭调试法，当你在向这只始终保持沉默的橡皮鸭子解释的过程中，你会发现你的想法、观点、思路和 实际的代码相偏离了，于是你也就找到了代码中的bug。一旦一个问题被充分地描述了它的细节，那么解决方法也是显而易见的。你觉得这个方法太“愚蠢”，太 “弱智”了？是的，看上去，会这样做的人脑子好像是有点毛病。不过，我要告诉你的是，这个方法的确有效。因为，这就是“Code Review”的雏形！

**9. 编写代码注释**

注释的功能就是在更易于理解的层次上解释代码的编写目的，尽可能多写一些：每行代码是干什么[](http://cms.csdnimg.cn/article/201308/15/520c85b33c782.jpg)的，怎么去完成，这些问题都应该在通读代码之后很容易找到答案才行。另外，给各个功能和变量取合理的名称也有助于简化代码实施的过程。在代码行下面的空白处填写注释来回答为什么要使用特殊的实现功能，或者一段代码怎样和程序的其余部分互动等等。

编写详细的注释可以说是软件工程里一步可靠地检验步骤，即使是在没有bug的代码里也是同样受用。这样，就算bug出现了也不用担心，注释会帮你节省数小时的排错时间。

**10. 编写文档**

代码注释是程序员以简单的方式和个人的观点编写的，而编写软件文档是用来描述软件系统的功能性，同时用户也可以看到这些软件文档。根据软件类型的不同，文档可以用来详述程序界面、图形界面或者工作流程。

编写文档还有一个好处就是，可以展示你对软件系统的理解程度，指出软件系统不够完善的部分或者有可能是bug源头的部分。