**传统调试技术**

软件开发者使用的典型的软件调试技术主要有两种:

1. 在程序中插入打印语句使得程序输出用以被分析的附加信息,可以对程序的运行时状态有一个粗略了解。一个通常的做法是插入打印语句以指示某个控制条件到达了某个特定程序点;另一个通常做法是插入打印语句以输出变量的值。当程序被执行的时候,程序生成可以被开发者检查的附加调试输出。**缺点：**调试输出可能相当的大,打印语句的放置和输出的检查都是非组织和临时的,分析和放置位置也通常是基于直觉的
2. 另一种技术是使用符号调试器。一个符号调试器是一个用来调试其他程序的计算机程序,符号调试器支持例如断点、单步跳过、状态修改等。断点允许程序员停止在某一个特定的程序点以检查当前状态;单步跳过允许程序员前进到当前断点之后的下一条指令,并且在那条指令设置新的断点;许多调试器还允许程序员不仅能够查看变量的当前状态,还能改变它的值然后继续执行。通常地,开发者会在他感觉可能是程序错误位置的地方设置断点,然后他会检查断点处的状态,他还可以单步跳过程序查看每一条语句在每一个执行时的状态变化。
3. 断言：断言是程序正确操作期间必须为真的程序。程序开发人员指定断言为条件语句如果断言为假，则程序停止。
4. Profiling通常是运行速度内存占用大小的分析指标，也可以用来调试程序

**主要的定位方法**

故障定位技术有很多种,但是根据定位故障的过程中“是否需要运行软件”的准则,可以将故障定位技术分为以下两类:

(1)基于静态分析的故障定位技术（SABFL):静态方法不用运行软件,而是依据程序语言的语法和语义,静态地分析软件结构和程序实体之间的依赖关系,以发现不符合系统约束的程序实体,从而进行故障定位。

(2)基于测试的故障定位技术(TBFL):该方法首先需要设计测试用例,然后运行软件程序,最后根据软件程序的动态执行信息和输出结果进行故障定位。其典型思路是:将程序的失败运行和成功运行进行对比,从而发现失败运行中的哪些点偏离了成功运行,找到的这些偏离点很有可能就是软件故障位置所在。这些不同定位方法都采用了各自不同的运行特征进行对比。

以下是基于测试的故障定位技术：

**（一）基于距离度量的故障定位方法：**

基本思想是:通过一定的距离计算方法,从众多的成功运行中找出与失败运行最相近的一个成功运行,利用某一种度量方法来计算此成功运行与失败运行之间的差异值,利用得出的最小差异值进行故障定位。该技术大都是在源程序上运行大量的测试用例,并收集程序运行过程中的覆盖信息,利用这些信息来进行故障定位。此方法包括

**（1）Delta调试技术**

把一次程序执行看做是一系列程序状态的转变(程序状态可以理解为在程序执行某个时刻出现的变量以及这些变量的取值)。程序从初始状态开始执行,每执行一步,程序当前状态就会发生改变,从当前状态转换到下一个程序状态,如此进行,最后到达了个错误的状态,标志着程序执行失败。每一个程序状态都它的前一个状态衍生而来,但是,可能只有一小部分状态与计算下一个状态是相关的,同理,也并不是所有程序状态都与执行失败相关。

Delta调试最重要的步骤就是找出与程序执行失败相关的那些状态,得到一条原因-结果链。比较一次成功执行和一次失败执行的状态序列,它们之间的差别就是一条原因-结果链。采用二分法将原因-结果链上的状态应用到成功的执行路径中,如果运行成功,就表示那一部分代码不存在故障,否则,那部分代码可能是可疑的故障点。经过这样的反复测验,最终可以将可疑故障点缩小到一个非常小的范围。

**优点：**delta调试完全依赖于对程序状态的研究,而不考虑对程序结构的分析不仅可以找出程序中的错误,还提供了纠错方案。**缺点：**对于大部分程序而言,一次执行涉及到成千上万个状态,比较程序状态的做法只适用于较小的程序。

**（2）“并集模型”:**在失败运行的覆盖语句集中查找可疑故障语句。其思想是,从失败运行的覆盖语句集中去掉那些出现在成功运行的覆盖语句集的并集。如果给定一个成功运行的集合S和一个失败运行f,将S中出现的语句覆盖集的并集记为失败运行中出现的语句覆盖集记为该模型的计算公式如式所示: 。

（3）**“交集模型”**:找到能判别这个失败运行的特征,其思想是:从所有成功运行的覆盖.语句集的交集中去掉那些出现在失败运行的覆盖语句集/中的语句,将得到的差异语句集作为可疑语句集。其计算公式如式所示:。

**缺点：**并集模型和交集模型的定位效果不是很理想。因为很多时候故障语句会同时出现在失败运行和成功运行中,当我们对这些语句集合进行集合运算时,故障语句经常会被误删。特别是当大量运行存在的情况下,利用该方法得到的结果集极有可能是空集或者根本就不包含故障语句。

**（4）近邻模型**

该方法首先选择一个失败运行,然后找到一个与此失败运行的覆盖信息最相近的成功运行。利用失败运行和与其最相近的成功运行,从失败运行的执行语句集中将那些出现在成功执行语句集中的语句剔除,剩余的其他语句集即可作为故障报告。近邻模型提供了两种方法来度量失败运行和成功运行之间覆盖信息的相似程度。一种叫做binary distancing,计算所选择的失败运行与一个特殊的成功运行之间的差异语句集合。该方法可以定义为“失败运行与成功运行执行语句集的不对称距离的基数或者对称距离的基数”。另一种是记录每一个运行所执行到的每一□条语句(或基本块)的执行次数,然后按照执行次数对语句进行排序,计算两个运行得到的有序序列之间的距离,该距离是将其中一个序列转化为另一个序列时所花费的代价。

**优点：**近邻模型得出的结论是:用近邻模型选择的成功运行通常比随机选择的成功运行更有利于故障定位,而且该模型在定位故障方面比并集和交集模型更好。

**缺点：**近邻模型通过计算两类运行(成功运行和失败运行)的语句集合之间的距离来寻找与失败运行最相近的成功运行,但是它却无法区分执行了相同语句但是语句执行顺序不一样的运行。

**近邻模型的改进**：Wang等人在的近邻模型基础上提出了一种新的差异度量方法,即利用程序控制流所对应的两条路径,求解两条路径之间的最小差异作为故障报告。它针对近邻模型不能识别语句执行顺序的不足提出将语句执行顺序作为衡量执行序列相似度的因素之一,将某条语句的一个执行实例看做一个事件,那么每个运行就可以看作是多个事件的执行序列,对失败运行的事件执行序列和成功运行的事件执行序列进行对齐比较操作,得出这两个运行中分支走向不同的分支事件,这就是Wang等人方法中使用的“差异”。还给出了定性判断任意两个差异之间的大小的差异度量方法。根据最小差异选择一个与失败运行最相似的成功运行,用二者之间的差异作为故障报告。**优点：**对分支语句故障的定位效果比近邻模型要好。**缺点：**对于赋值语句故障的定位效果却不甚理想。当程序故障并不包含在最小差异中时,利用该方法进行故障定位将得不到理想的效果。

**（二）基于排序的故障定位方法**

上面的方法的相同点在于:认为程序中只有一部分语句是可疑的故障点,赋予它们相同的故障可疑值,认为应该花费同样多的代价检查这些可疑点。而另一类研究则认为:程序中的每一部分都是可疑的故障点,根据它们在运行时的参与度和表现不同,赋予它们不同的故障可疑值,然后将故障可疑值降序排列,程序员按照排序由大到小检查程序,直至找到故障。

基于排序的故障定位方法的思想是:通过分析程序的动态运行信息来进行故障定位。它通常是比较两类运行(成功运行和失败运行)中程序状态特征的统计信息,直接定位到故障语句。该技术的区别在于使用哪些程序运行特征来进行对比分析。目前常用的程序实体主要有:语句和分支,它们的主要载体是程序频谱信息,主要利用程序频谱(信息来获取程序实体的特征信息。利用这些信息来计算程序实体(语句或分支)的可疑度,然后根据可疑度大小对被检测的程序实体(语句或分支)进行降序排序。 开发人员在调试的过程中可以根据排序表从前至后始查找程序故障,起到了一定的辅助作用。

**（1）Tarantula 方法**

釆用的思想是:如果一条语句在失败的执行中出现的次数比在成功的执行中出现的次数多的话,那么该语句是故障语句的可能性就比较大。试题的怀疑度具体算法如下：



其中,表示失效测试用例中执行语句数/失效测试用例的总数,%p(e)表示成功测试用例中执行语句数/成功测试用例的总数。

Tarantula是目前大家公认的最好的基于统计的故障定位方法。**缺点：**该方法要求两种测试用例要能够覆盖尽可能多的代码。此外,Tarantula方法难以发现像赋值语句这样的程序初始化错误。也不能区分一个失效测试用例在另一个测试用例中的作用,或者一个成功测试用例在另一个测试用例中的作用。

**多程序谱模型(LOUPE)：**这种方法是对Tarantula的一种改进,即在Tarantula所使用的程序谱上增加了依赖对的覆盖信息构成了多程序谱。所谓多程序谱就是加入了程序执行过程中控制依赖和数据依赖的覆盖信息,并根据这些信息来总体计算某个语句的可疑值。

**（2）Wong 方法**

Wong等人认为随着语句成功执行次数的增加,其对可疑度贡献率逐渐减小。该方法将语句i成功执行次数划分为3个区间[0,2],[3,10],[11,+00)。区间[0,2]贡献率权重为1,区间[3,10]的权重为0.1,区间[11,+00)贡献率权重为a (a <0.1),其可疑度计算公式为: 

表示语句i失败执行的次数，可以计算出语句i成功执行对可疑度的贡献值,最后用语句失败执行贡献值减去成功执行贡献值,即可得到该语句i的可疑度。例如,一个语句失败执行的次数为5,它对可疑度的贡献值为5,成功执行的次数为6,它对可疑度的贡献值为2.4,最后得到语句i的可疑度为5-2.4=2.6。**缺点：**受测试用例集中正确测试用例数和错误测试用例数之比影响较大。

**（三）程序切片方法**

**（1）静态切片**

程序切片是一种常用的调试方法,通过切片减少调试搜索的范围,一个测试用例执行结果失效是由于程序语句中执行了不正确的变量值而引起的,这种错误往往出现在与该变量语句配对的静态切片中。切片方法是一种最初的比较直观的方法,故障会出现在一个失效的测试用例执行的切片中。公式：

表示在测试用例中失效执行的语句的集合除去测试用例中成功执行的语句的集合，即于它们的差集。

**缺点：**这种技术的缺点是可能出现某些可疑语句不包含在切片中的情况,定位效果不理想。问题是如果切片规模较大时,有效性会降低,而且这种算法的时间开销会很大,最重要的是,他无法对在切片中的语句进行可疑度排名而直观的提供给开发人员作为参考。

**（2）动态切片**

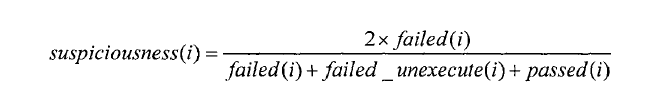
为了弥补静态切片的不足,引入了动态切片方法.

定义切片准则: 静态切片不关心程序的输入,它的切片准则形如<i, V>,其中i为程序中的一条语句,V表示该语句中使用的变量集合;与静态切片不同,动态切片需要一个特定的输入,它的切片准出形如<1,i, V>,其中I表示程序的一个输入,i和V所表示的意思与静态切片准则相同。

定义切片:根据切片准则,通过抽取与切片准则存在数据依赖或控制依赖关系的语句,得到的一个程序片段,叫做原程序的一个切片.减少了故障定位的检查范围。

**优点：**在故障定位过程中,往往需要忽略那些对暴露故障无关的语句,而程序切片正是对程序做了这样的提取,让程序员集中精力去检查那些与错误相关的代码。**缺点：**对于大型程序而言,经过各种切片算法得到的切片往往非常大,反而会降低其效率。

Naish等人将其应用到软件故障定位中,并根据切片方法提出语句可疑度计算公式:



总的来说,基于排序的故障定位技术大概有三个步骤:1)收集某种粒度的实体的程序执行路径;2)分析成功的执行路径和失败的执行路径,计算每个实体的故障可疑值;3)按照故障可疑值降序排列程序实体,作为程序员检查程序的依据。这种技术根据程序不同部分在程序执行时的表现行为和参与度,给每个部分赋予不同的故障可疑值,建议程序员对程序故障可疑值的大小花费不同的代价检查程序。

**（四）概率统计分析方法**

1. **基于统计的定位方法**SBI(statistical bug isolation)，通过计算一个谓词P能够蕴含程序失效的可能性Failure(p)来定位，用公式表示如下：



Passed(p)表示P背观测到为真的成功测试用例个数，failure(p)表示p被观测到为真的失效测试用例个数。**优点**：SBI方法能够隔离程序中的多重错误。**缺点**：只是记录一个实体在执行过程中是否被覆盖到,不足以捕获每`个测试用例的行为信息。

**（2）SOBER方法**：改善SBI的不足。它记录每一个运行中每个实体的执行次数,构造了一个失效和成功测试用例行为的统计模型,使用这个统计模型评估每个实体以确定它们是否可疑。SOBER定义赋值偏差来估计在每个运行时谓词被赋值为真的几率。更确切地,如果p是一个谓词,是它在每次运行时被赋值为真

可能性,=nt/(nt+nf) 其中nt表示尸被赋值为真的数量,nf表示尸被赋值为假的数量。SOBER计算成功和失败的运行的分布之间的差异,认为差异越大,p与这次运行错误就越相关。

**（五）模型诊断方法**

思想：基于模型诊断的故障定位技术,首先假设所有程序部件都均为正确执行,但是当使用测试用例执行时得到的结果是失效的就说明该测试用例所覆盖到的语句中包含故障点,然后利用所有失效用例的语句複盖信息进行计算求得故障点所在。然而该方法只能查找到独立的程序语句,而没有考虑程序语句间的依赖关系。

总结：

|  |  |  |  |
| --- | --- | --- | --- |
| 方法 | 按成功|失效差异 | | 按统计 |
| 交并模型 | 是 | |  |
| 近邻模型 | 是 | |  |
| Wang算法 | 是 | |  |
| SBI |  | | 是 |
| Tarantula |  | | 是 |
| SOBER |  | | 是 |
| Wong |  | 是 | |