文章编号: 1671-5896(2014)05-0528-06

基于程序执行轨迹与动态切片的错误定位研究

孙士明[1], 侯秀萍[1], 高 灿[2], 孙琳琳[1]
(1. 长春工业大学 计算机科学与工程学院, 长春 130012; 2. 苏州大学 附属第一医院, 江苏 苏州 215006)

摘要: 为解决程序调试过程中的错误定位问题, 将程序执行轨迹和动态切片技术应用于错误定位。程序执行轨迹中包含与错误无关语句, 影响错误定位的准确度。在执行轨迹的基础上, 通过使用动态切片技术降低不相关语句在错误定位时的影响。建立基于程序执行轨迹和动态切片的语句怀疑度计算模型, 使用该模型计算每条语句的怀疑度, 并根据怀疑度对每条语句进行排序, 给出查错的推荐方案。通过实验对比其他算法, 证明了基于程序执行轨迹与动态切片的错误定位方法是有效的。

关键词: 执行轨迹; 动态切片; 怀疑度
中图分类号: TP311 **文献标识码:** A

Research on Fault Localization Based on Execution Trace and Dynamic Slicing

SUN Shiming[1], HOU Xiuping[1], GAO Can[2], SUN Linlin[1]
(1. School of Computer Science and Engineering, Changchun University of Technology, Changchun 130012, China;
2. The First Affiliated Hospital, Soochow University, Suzhou 215006, China)

Abstract: The problem of faulty localization during program debugging is studied, and execution trace and dynamic slicing are applied to faulty localization. Irrelevant statements of fault localization are included in execution trace, accuracy of fault localization is affected by these statements. The influence of irrelevant statements is reduced by dynamic slicing technology based on execution trace. The model of suspicious degrees computing based on execution trace and dynamic slicing had been established, and suspicious degree of each statements could be computed by this model. After sorting these statements by their suspicious degrees, recommendation of fault localization would be reached. The experimental results are compared with other algorithms', and the method of faulty localization based on execution trace and dynamic slicing is proved effective.

Key words: execution trace; dynamic slicing; suspicious degrees

0 引 言

软件生命周期中的任何一个阶段的程序调试都需要程序员进行大量的人机交互[1]。为保证软件开发的质量, 工业界在软件测试阶段投入了大量的人力物力。据统计, 软件测试约占软件开发和维护成本的 50% ~75%[2], 其中最耗时、代价最昂贵的任务之一就是程序调试[3-4], 这是指对程序错误进行定位和修正的过程, 而错误定位又是程序调试中最耗时和困难的任务。

在程序的调试过程中, 程序员准确识别导致程序出错的位置称为错误定位[7]。错误定位的有效性取决于程序员对程序的理解程度, 以及逻辑判断能力和调试经验。错误定位研究可以分为两部分[8]: 使用

收稿日期: 2014-01-14
基金项目: 国家科技部 863 高技术基金资助项目(2011AA040602)
作者简介: 孙士明(1986—), 男, 山东莱芜人, 长春工业大学硕士研究生, 主要从事软件工程研究, (Tel) 86-13756116537(E-mail) sunshiming0634@qq.com; 通讯作者: 侯秀萍(1964—), 女, 长春人, 长春工业大学教授, 硕士生导师, 主要从事软件工程研究, (Tel) 86-18643194774(E-mail) houxiuping@mail.ccut.edu.cn。

某种技术识别可能含有错误的代码和程序员通过检查识别含有错误的代码。笔者研究的是识别可能含有错误的代码。

1　错误定位方案

在错误定位研究过程中，有很多种方法可以定位错误。常见的方法是对源程序进行相应的程序理解或动态跟踪出现错误的测试用例的执行轨迹并获取错误信息，利用人工智能的方法或统计的方法计算每条语句的怀疑度[9]，然后定位错误可能发生的位置。笔者根据程序的执行轨迹和动态切片信息进行错误定位。由于面向对象程序具有抽象、封装、继承和多态的特性，对于部分具有多态特性的程序，动态切片获取工具不能获取准确的切片[10]，如果只使用动态切片技术进行错误定位，这将导致动态切片中可能不会包含错误语句，使错误定位的精度降低。含有错误的语句一定会在执行轨迹中，但执行轨迹中包含大量与错误发生无关语句，利用动态切片技术降低执行轨迹中无关语句的影响，又不会遗漏执行轨迹中的错误语句。笔者进行错误定位的流程如图 1 所示。

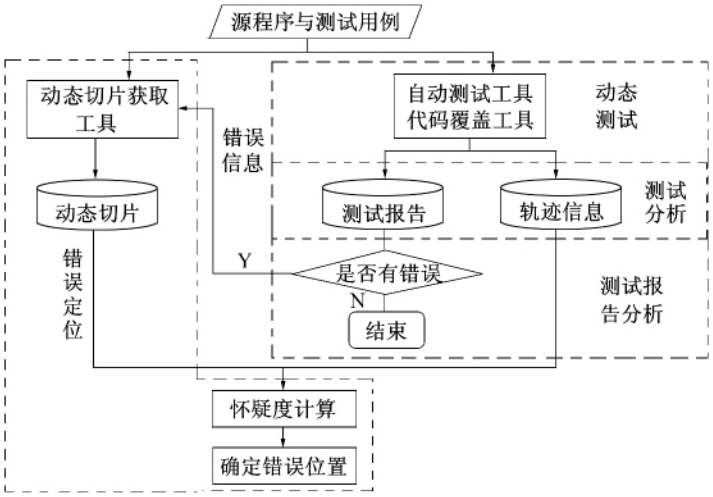

图 1　错误定位流程图

Fig. 1　Flow chart of fault localization

首先运行测试用例，在运行测试用例时记录程序的执行轨迹和生成测试报告。分析测试报告内容，报告中是否含有运行失败的测试用例，如果没有，则测试结束；如果有，将错误信息和参数信息传递给动态切片获取程序，计算每个参数信息对应的动态切片。最后利用怀疑度计算模型估计每条语句含有错误的可能性，给出推荐方案，确定错误发生的位置。

2　怀疑度计算模型

2.1　相关定义

错误定位的过程就是计算每条语句含有错误可能性大小的过程，也称作怀疑度计算。为计算怀疑度，做如下定义。

定义 1　**失败切片数 m_i：由运行失败的测试用例对应的参数计算得出的动态切片中覆盖到语句 S_i 的动态切片数。**

定义 2　**正确切片数 n_i：由运行成功的测试用例对应的参数计算得出的动态切片中覆盖到语句 S_i 的动态切片数。**

定义 3　**失败路径数 p_i：动态测试中，测试用例运行失败且路径中覆盖到语句 S_i 的测试用例数。**

定义 4　**正确路径数 q_i：动态测试中，测试用例运行成功且路径中覆盖到语句 S_i 的测试用例数。**

定义 5　**x：运行失败的测试用例总数。**

定义 6　**y：运行成功的测试用例总数。**

定义7 影响因子 α_i：α_i 是经过动态切片后，通过执行 S_i 并且运行失败的测试用例数与总的运行失败测试用例数的比值进行计算，即 $\alpha_i = m_i/x$。

定义8 影响因子 β_i：是经过动态切片后，通过执行 S_i 并且运行成功的测试用例数与总的运行成功测试用例数的比值进行计算，即 $\beta_i = n_i/y$。

2.2 怀疑度模型

运行成功或失败的测试用例对判断语句中含有错误均有贡献[11]。对每条语句 S_i，当测试用例运行失败且运行 S_i 时，则认为语句 S_i 含有错误的可能性增加；当测试用例运行成功且运行 S_i 时，则认为语句 S_i 含有错误的可能性降低。运行成功的测试用例和运行失败的测试用例的数量不同，运行成功的测试用例多时，会导致所有的语句含有错误的可能性降低。所以，不仅考虑运行成功或失败且经过 S_i 的数量，还要考虑其在总的运行成功或失败的测试用例中所占的比例。在获取的路径中含有导致错误产生无关的语句，为减少这些语句，对产生错误语句中的变量做动态程序切片。通过程序切片降低不相关的语句含有错误的可能性。

怀疑度 $S_{\text{Suspicious}}(S_i)$，语句 S_i 不同时，S_i 中含有错误的可能性大小也不一样，把这种含有错误的可能性定义为怀疑度，怀疑度的计算公式如下

$$S_{\text{Suspicious}}(S_i) = \alpha_i \frac{p_i}{x} - \beta_i \frac{q_i}{y} \tag{1}$$

3 Trace & Slicing 错误定位算法

笔者进行错误定位的思想是，先分析动态测试报告，检查是否有运行失败的测试用例，如果有运行失败的测试用例，根据运行过程中的错误信息获取程序的动态切片，再根据动态切片信息和执行轨迹信息计算每条语句的怀疑度，进行错误定位。算法所使用的类之间关系的设计如图2所示。

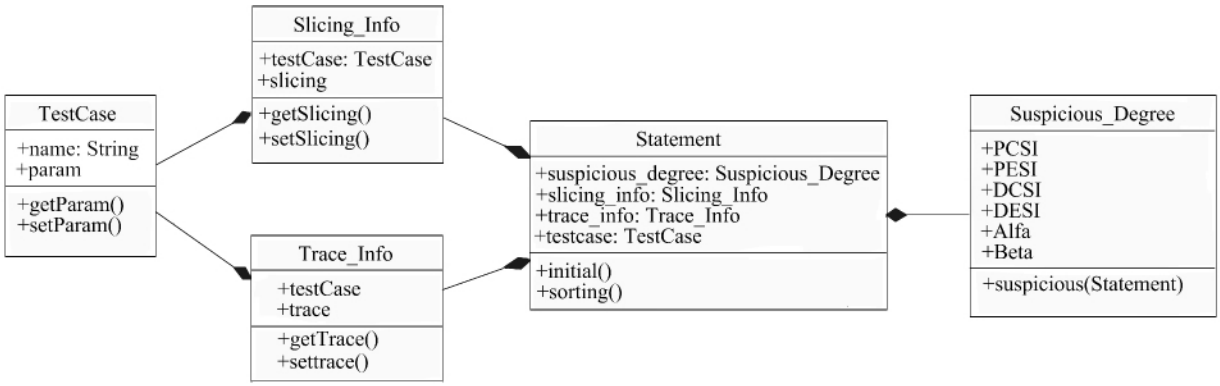

图2 类关系图

Fig. 2 Class diagram

在图2中 TestCase 为测试用例类，其中 name 为测试用例名，param 为测试用例的参数信息，该参数也作为获取动态切片的参数。Slicing_Info 为动态切片信息，每个测试用例的参数对应一条切片信息。Trace_Info 为执行轨迹信息，每个测试用例对应一项执行轨迹信息，该信息保存在测试报告中。Statement 存储每条语句动态切片和执行轨迹的覆盖信息。Suspicious_Degree 为怀疑度，其中 suspicious(Statement) 方法计算怀疑度，具体计算过程根据式(1)进行，每条语句对应一个怀疑度值，根据怀疑度值的大小判断含有错误可能性的大小。

3.1 算法描述

算法 Trace & Slicing 具体描述如下。

输入：测试用例集，源程序

输出：查错方案

initial TestCase，Path_Info and flag; //Path_Info 数组存储每个测试用例的路径信息，flag 标注是否有运行失败的用例

```
for( int i = 0; i < Path_Info. length; i + + ) {
    if( Path_Info [i] . result = = False) {
        computing dynamic slicing; //如果有运行失败的测试用例，则计算对应的动态切片
        initial Statement; //初始化每条语句的状态
        flag = true;
        break;
    }
    flag = false;
}
if( flag) { //如果有运行失败的测试用例，则计算每条语句的怀疑度，进行错误定位
    for( int j = 0; j < Statement. length; j + + ) {
        Statement [j] . suspicious_degree = Suspicious( Statement [j]) ; //计算每条语句的怀疑度
    }
        sorting Statement by suspicious_degree and recommending scheme of finding fault/* 根据怀疑度对语句排序并给出推荐方案* /
}
```

3.2　实例分析

图 3 所示的程序为计算输入的 3 个数的中间数，其中在第 7 行中正确的语句应该是 $m = x$，由于误写成 $m = y$，导致程序存在错误。下面将用此例介绍如何定位错误发生的位置。

```
1 public class name
2 {
3 public String str;
4     public void MiddleNum(int a,int b,int c){
5     int x = c;
6     if (b<c){
7       if (a<b)
8           x=b;
9       else if (a<c)
10          x=b;/*error: should be x=a;*/
11    }else{
12    if (a>b)
13        x=b;
14     else if (a>c)
15        x=a;
16    }
17    System.out.println(x);
18    }
19 }
```

图 3　MiddleNum 源码图

Fig. 3　Source code of MiddleNum

对图 3 中的程序编写了 8 个测试用例(见表 1) ，这 8 个测试用例分别选取不同的参数。T_1 是一组随机数，T_2 选取两个相等的参数，$T_3 \sim T_8$ 选取 3 个不同大小的数，对其进行排列组合作为测试用例的输入参数。表 2 中的数据是根据定义 2.1 中的定义进行计算的，怀疑度根据式(1) 计算得出，其计算结果如表 2 所示。

表 1　测试用例表

Tab. 1 Test cases

用例名	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
参数	(1,2,3)	(0,0,1)	(-1,0,1)	(-1,1,0)	(0,-1,1)	(0,1,-1)	(1,-1,0)	(1,0,-1)
运行结果	True	True	True	True	False	True	True	True

表 2　MiddleNum 怀疑度表

Tab. 2　Suspicious degree of middle num

语句	q_i	p_i	n_i	m_i	$\lvert\alpha_i\rvert$	$\lvert\beta_i\rvert$	怀疑度
S_1	7	1	7	1	1	1	0
S_2	7	1	7	1	1	1	0
S_3	7	1	0	0	0	1	0
S_4	3	1	0	0	0	1	0
S_5	2	0	2	0	0	0.285 7	0
S_6	2	1	0	0	0	0	-0.081 6
S_7	1	1	1	1	1	0.142 9	0.979 6

(续表2)

语句	q_i	p_i	n_i	m_i	$\lvert\alpha_i\rvert$	$\lvert\beta_i\rvert$	怀疑度
S_8	3	0	0	0	0	0	0
S_9	3	0	0	0	0	0	0
S_{10}	1	0	1	0	0	0.142 9	-0.020 4
S_{11}	2	0	0	0	0	0	0
S_{12}	2	0	2	0	0	0	-0.081 6
S_{13}	2	0	0	0	0	0	0
S_{14}	7	1	7	1	1	1	0
S_{15}	7	1	7	1	1	1	0

根据怀疑度排列语句可得出如下序列

$S_7 -> S_1 -> S_2 -> S_3 -> S_4 -> S_5 -> S_8 -> S_9 -> S_{11} -> S_{13} -> S_{14} -> S_{15} -> S_{10} -> S_6 -> S_{12}$

由上面的序列可得出语句 S_7 含有错误的可能性最大，程序员查找差错时可首先检查语句 S_7 是否含有错误。

4 对比分析

为评估所提出算法的有效性，本节将对比 Trace & Slicing 算法与 Tarantula[9] 算法的有效性(见图4～图6)。Tarantula 算法通过程序的执行轨迹信息计算语句含有错误的可能性，只考虑了运行成功与运行失败的测试用例的执行轨迹是否覆盖了该语句，没有考虑不相关语句的影响。该实验数据来自 http://sir.unl.edu，其中有 XML-Security，JDepend，CheckStyle 测试程序。

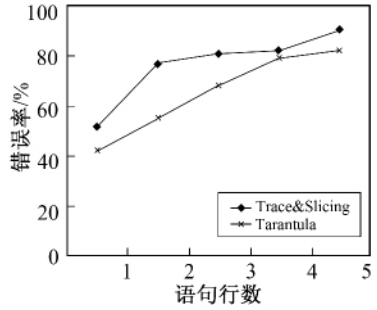

图4 XML-Security 错误定位速率

Fig.4 Rate of fault localization of XML-security

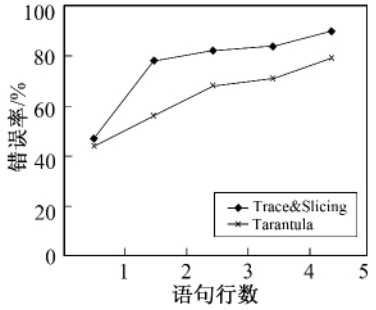

图5 JDepend 错误定位速率

Fig.5 Rate of fault localization of JDepend

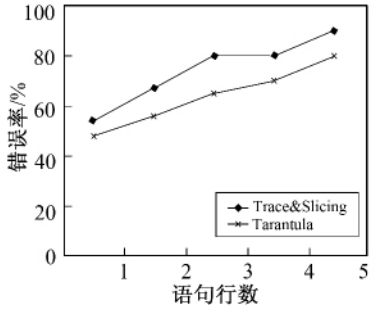

图6 CheckStyle 错误定位速率

Fig.6 Rate of fault localization of CheckStyle

实验对比了经过错误定位算法计算后，前5行推荐语句发现错误的百分比。通过图4～图6可以看出，Trace & Slicing 算法经过动态切片降低不相关语句的影响后，含有错误语句的排列被前置。在被测试的3组数据中，Trace & Slicing 比 Tarantula 发现错误的比例高，并且 Trace & Slicing 算法能在前3行定位到约80%的错误。

5 结 语

测试用例的运行信息可用于错误定位，并且导致错误产生的语句包含在用例的执行轨迹中。执行轨迹中含有大量与错误产生无关的语句，检查这些无关的语句是否含有错误会浪费程序员的时间。通过动态切片技术降低无关语句的影响，使可能对错误产生有影响的语句排列提前，有助于程序员提前发现错误。通过实验结果证明，笔者提出的算法提高了错误定位的效率。

参考文献:

[1] JONES J A, BOWRING F J, HARROLD M J. Debugging in Parallel [C] //Proceedings of the 2007 International Symposium on Software Testing and Analysis (ISSTA 07). New York, NY, USA: ACM, 2007: 16-26.

[2] HAILPERN B，SANTHANAM P. Software Debugging，Testing，and Verification [J]. IBM Systems Journal，2002，41(1)：4-12.

[3] JONES J A. Semi-Automatic Fault Localization [D]. Atlanta，USA：Computing Department，Georgia Institute of Technology，2008.

[4] SANTELICES R，JONES J A，YU Y，et al. Lightweight Fault-Localization Using Multiple Coverage Types [C] // Software Engineering，2009，ICSE 2009，IEEE 31st International Conference on. Vancouver，Canada：IEEE，2009：56-66.

[5] 虞凯，林梦香. 自动化软件错误定位技术研究进展 [J]. 计算机学报，2011，34(8)：1411-1422.
YU Kai，LIN Mengxiang. Advances in Automatic Fault Localization Techniques [J]. Chinese Journal of Computers，2011，34(8)：1411-1422.

[6] 张云乾，郑征，季晓慧，等. 基于马尔科夫模型的软件错误定位方法 [J]. 计算机学报，2013，36(2)：445-456.
ZHANG Yunqian，ZHENG Zheng，JI Xiaohui，et al. Markov Model-Based Effectiveness Predicting for Software Fault Localization [J]. Chinese Journal of Computers，2013，36(2)：445-456.

[7] WONG W E，DEBROY V. Software Fault Localization [J]. IEEE Transactions on Reliability，2010，59(3)：473-475.

[8] WONG W E，DEBROY V，CHOI B. A Family of Code Coverage-Based Heuristics for Effective Fault Localization [J]. Journal of Systems and Software，2010，83(2)：188-208.

[9] XIE X，WONG W E，CHEN T Y. Metamorphic Slice：An Application in Spectrum-Based Fault Localization [J]. Information and Software Technology，2013，55(5)：866-879.

[10] SAHOO S K，CRISWELL J，GEIGLE C，et al. Using Likely Invariants for Automated Software Fault Localization [C] // Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems. New York，NY，USA：ACM，2013：139-152.

[11] JONES J A，HARROLD M J，STASKO J. Visualization of Test Information to Assist Fault Localization [C] // Proceedings of the 24th International Conference on Software Engineering. Orlando，FL，USA：ACM，2002：467-477.

（责任编辑：刘俏亮）