一种动态监测软件行为的方法

王 鹏，李长云

（湖南工业大学 计算机与通信学院，湖南 株洲 412008）

摘 要：分析了传统测试的弊端和动态监测的现状，提出并实现了根据形式化逻辑描述的需求规约，判断软件运行行为是否符合预期的动态监测方法，在资源消耗较低的情况下保证了监测的正确性与及时性。最后通过实例数据证明了该方法的高效性。

关键词：动态监测；需求规约；形式化逻辑

中图分类号：TP311 **文献标志码**：A **文章编号**：1673-9833(2010)06-0041-04

A Method of Dynamic Monitoring Software Behavior

Wang Peng，Li Changyun

（School of Computer and Communication，Hunan University of Technology，Zhuzhou Hunan 412008，China）

Abstract：Analyzes the drawbacks of the traditional tests and the status of dynamic monitoring, proposes and implements dynamic monitoring method to determine whether software running behavior accords with the expected behavior based on formal logic description of requirements specification, and ensures the timeliness and correctness of the monitoring in the lower resources consumption. Finally，the efficiency of the algorithm is proved by practical case.

Keywords：dynamic monitoring; requirements specification; formal logic

0 引言

软件复杂度及自主化程度的日益提高使得错误与失败率也随之提高。传统测试是在软件开发阶段用穷尽的方法来检测可能发生的行为，耗时耗力，并缺乏对软件外部行为的监测，而开放动态、松散聚合和行为复杂的分布式软件则希望监测具有动态性，并能积极防御突发灾难性故障。为此本文提出了一种动态监测方法。动态监测是指跟踪所监测对象运行中的变化，看其是否符合需求规约所描述的预期行为，它只关注状态间的真实转变而不是可能发生的改变，其最大特点是随时都能获得监测对象属性或行为的信息，掌握软件实时行为，准确分析和定位软件故障[1-2]，提高了软件的可靠性、可用性、健壮性等可信性质，所以它是更适用未来发展的行为监测方法。

目前动态监测方法主要有以下几类：

1）基于AOP (aspect oriented programming)，采用关注点分离原则，监测模块和功能模块松散耦合，适合规模不断扩展的分布式软件[3]。缺点是目前大多采用AOP的静态织入方式，不支持动态织入，不适合于软件实体动态加入或离开的分布式软件。

2）基于截获器，可在不被应用感知的情况下动态地添加或删除。缺点是局限于基于CORBA平台的分布式软件[4-5]，不适合开放网络环境横跨多个异构中间件平台的分布式软件。

3）创建一个观察者进程来达到监测程序执行的目的。缺点是消耗资源多，并可能造成死锁。

本文的动态监测方法在算法上是用形式化逻辑语言进行事件演算，因为代数法和自动机都先假定行为不正确[6-8]、模型不完整来验证系统性能，可能陷入模

收稿日期：2010-09-19

通信作者：王 鹏（1983-），男，山东青岛人，湖南工业大学硕士研究生，主要研究方向为可信软件故障诊断，嵌入式应用，E-mail：wp19831014@sina.com

型检测和公理证明的复杂性陷阱中。结构上采取关注点分离，提高了松散耦合，更利于维护。

1 动态监测方法

图1设计了分布式软件动态监测方法的基本框架，包含人工执行、静态编译和动态执行。人工执行是根据需求规约中定义的形式化逻辑来人工编写脚本，包括监测脚本和判定脚本；静态编译是将这2种脚本进行编译，而且把监测脚本生成的包含所要监测事件的文件和源程序一起放在监测分析器中执行，监测分析器将源程序的方法、局部变量、全局变量和监测的事件和条件相对应，并产生一序列监测值；动态执行首先利用 filter（滤波器）从监测值中提取事件识别器所需的和抽象树节点相关的低层事件，然后事件识别器将这些低层次事件转化成高层次事件传给动态监测器，最后它根据记录和判定脚本的需求规约判定高层事件的行为是否符合预期行为。

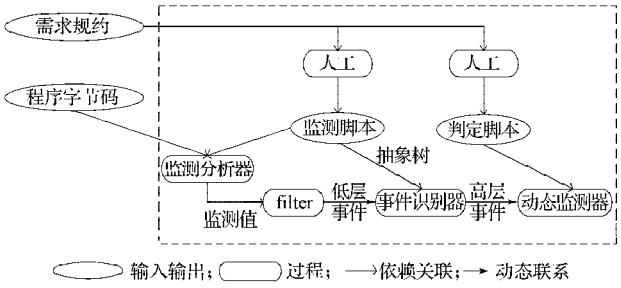

图1 系统结构框架图

Fig. 1 System structure framework

1.1 事件演算

动态监测方法要进行事件演算，就需要有一个动态监测执行点，这个点可以是源代码所在行，也可以是从方法的开始到结束。笔者采用后者，因为前者没有继承关系，而且遇到断行容易失误。方法中包含了脚本所需的事件和条件。事件是指在某个时间点系统状态发生变化，只有2个状态：出现和未出现；条件是指一段时间内特性不变的系统状态，有3个状态：正确、错误、不确定。动态监测执行点它是一阶谓词时序逻辑形式化语言，描述随时间而改变的动态系统的特性，事件关系必须有序且为偏序关系，不能用循环或递归。虽然可以提高识别事件的概率，但更能造成结点循环导致系统崩溃，所以内部事件演算机制是用一个拓扑结构的无环有向图[8]，其中每个节点代表一个事件，整个图构成了事件的逻辑运算。事件的全局状态可以是全部结点的子集，如果这个子集中还包括顶点，那么所有进入这个顶点的结点也属于这个子集。每一个节点也就是实际监测点，提取监测点需要把握好一个“度”：监测点的信息太多就会消耗资源，太少会丢失所需信息导致监测效果失常或失败。

1.2 形式化逻辑语言

采用形式化逻辑语言进行事件演算，表述简单直观，便于执行。

1.2.1 事件和条件基本逻辑

事件和条件定义如下：

1）如果 $C1$ 和 $C2$ 是条件，则!$C1$, $C1$&&$C2$, $C1$||$C2$, 都是条件。

2）如果 $E1$ 和 $E2$ 是事件，则[$E1$, $E2$)是条件。

3）如果 $E1$ 和 $E2$ 是事件，则!$E1$, $E1$&&$E2$, $E1$||$E2$, $E1\rightarrow E2$ 都是事件。

4）如果 C 是条件，则 start(C), end(C)是事件。

5）如果 E 是事件、C 是条件，则 Ewhen(C)是一个事件。

6）每个命题都是一个原始条件。

1.2.2 句法和语义

假定简单条件有限集合 $C=\{c1,c2,c3,\cdots\}$，它们是由源程序变量值组成的最基本布尔型表达式[4]；假定简单事件的有限集合 $E=\{e1,e2,e3,\cdots\}$，当监控变量的值发生变化时，相对应的事件就会发生；假定状态的有限集合 $S=\{s1,s2,\cdots\}$；假定一个模型 M 它是一个四元数组即 $M=\{S,\tau,L_C,L_E\}$，其中 τ 是 S 到时间域的映射，L_C 是一个全功能函数，是 S 到 C 的映射即每一个状态 si 都有一个 c 和它对应；L_E 是一个偏函数 S 到 E 的映射，即事件 e 在状态 si 的发生情况。

定义1 $(M,t\models c)$ 在模型 M 的 t 时刻条件 c 为 true。

定义2 $(M,t\models e)$ 在模型 M 的 t 时刻条件 e 出现。

定义3 $M,t\models[e1,e2)$存在 $t_0\leq t$, $M,t_0\models e1$ 并且对于任何 $t_0\leq t'\leq\leq t$, M, $t'\succ e2$。

定义4 $M,t\models c1\Rightarrow c2$ 即 $M,t\succ c1$ 或者 $M,t\models c2$。

定义5 $M,t\models$start(c) 如果存在 si，则 $\tau(si)=t$ 并且 M, $\tau(si)\models c$ 且 $\tau(si-1)\succ c$。

定义6 $M,t\models$end(c) 如果存在 si，那么 $\tau(si)=t$ 并且 M, $\tau(si)\succ c$ 且 $\tau(si-1)\models c$。

定义7 $M,t\models e$ when c $M,t\models$c 且 $M,t\models e$。

定义8 $M,t\models e1\rightarrow e2$ 如果存在 si 且 $t1>t$，则 $\tau(si)=t$ 并且 M, $\tau(si)\models e1$ 且 $M,t1\models e2$。

定义9 $M,t\models e1\leftarrow e2$ 如果存在 si 且 $t1>t$，则 $\tau(si)=t$ 并且 M, $\tau(si)\models e1$ 且 $M,t0\succ e2$。

定义10 $M,t\models e1\uparrow e2$ 如果对于任意 si 且 $t1>t$，则 $\tau(si)=t$ 并且 M, $\tau(si)\models e1$ 且 $M,t1\models e2$。

定义11 $M,t\models e1\downarrow e2$ 如果对于任意 si 且 $t1>t$，则

$\tau(si)=t$ 并且 $M, \tau(si) \models e2$ 且 $M,t1 \succ e1$。

2　数据存储和降低资源消耗

动态监测不仅需要能在运行中进行软件行为的监测，更需要在监测时避免产生过多文件浪费资源与空间。采用全路径的形式来记录程序，即用“类.方法名.变量名”的形式来识别需要监测的变量，以避免额外的内存防止引用冲突。

2.1　数据存储方式和抽象树演算

当需求规约被执行后，它生成变量 / 方法表、事件 / 条件表，它们和一个事件 / 条件树相联系，如图 2 所示，这些全部保存在一个事件识别器里，它负责提取高层次事件并发送到一个动态监测器。这棵抽象树保存了事件和条件在需求规约中的依赖关系。每棵抽象树的根结点都有一个算法使用的标记，如在事件 / 条件表中有一个 present-flag，如果事件存在则设置为 present；而条件有一个 truth-value 标记，当条件为真时为 true。抽象树的演变依赖关系如下：1）清空所有事件 / 条件树根结点的标记；2）为每一棵事件 / 条件树重新设置被算法使用的标记；3）如果结点是连接或操作符，执行相关操作获得子树的值，如果子树也是连接或操作符重复这个步骤；4）如果结点是来自一个变量 / 方法表中的引用，就从变量 / 方法表中获取其值；5）如果结点是来自事件 / 条件表中的引用，就需要得到这个结点的 present-flag/truth-value 值；如果根结点已经被标记，则记录 present-flag/truth-value 值；如果根结点还没有被标记，重复步骤 3）~5）。

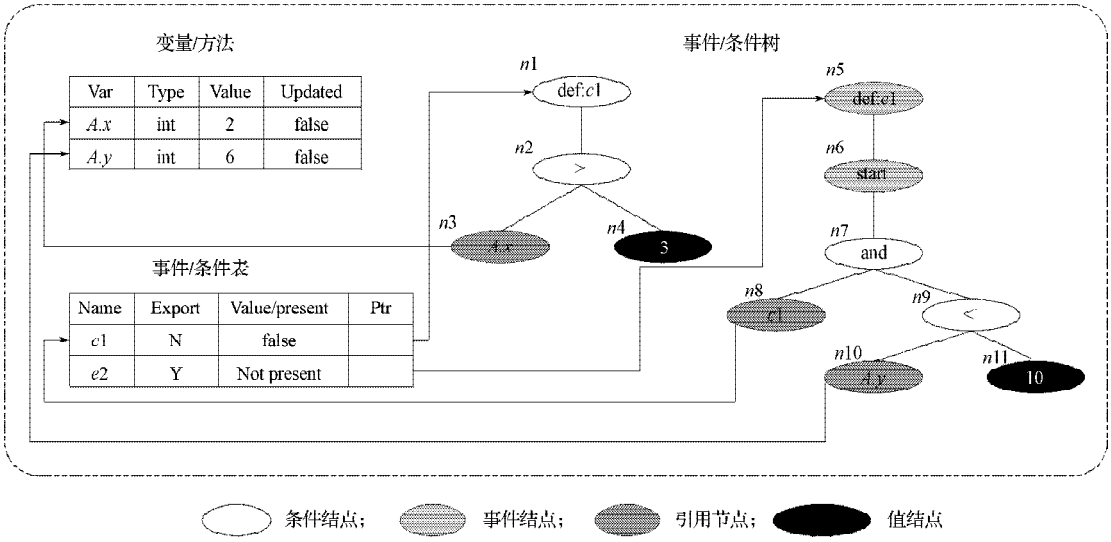

图 2　数据存储

Fig. 2　Data store

2.2　减少资源消耗

在 filter 提取与抽象树存储有关的信息，并将这些信息以快照的形式传送给事件识别器，此过程中会消耗过多资源，因此这里要考虑降耗问题。每次传送的快照只记录监控变量的 ID(1 B)，变量的值(1~8 B)，不同时传送时间戳（8 B）。如果每次传送时间戳，其资源占总资源的 47%~80%，故采用周期发送时间戳以减少资源开支。如发送周期是 100 ms，快照 1 次 /ms，那么持续发送快照时，1 秒内发送 8 000 B，而周期发送快照时，1 秒内发送 100 B（周期发送必须多加 1 个 ID）。

3　实验例证

该实验是随机产生 10 个整数，如果随机产生的 2 个相邻数字是 5 和 6，则该动态监测方法就认为此情况属于违例，出现报警并给予提示，监测脚本主要代码如下：

```
monobj int RN.t;
event startPgm = start(RN.t==5);
event startPgm6 = start(RN.t==6)
event periodStart = startM(RN ());
event mainStart=startM(RN.main(String[]));
event endMainStart=endM(RN.main(String[]));
```

判定脚本主要代码如下：

```
property safeRRC2 =ic ;
condition ic =startPgm6Num!=1 || startPgmNum==0;
mainStart->{initTime=0;print"mainStartTime:"+initTime;
a2=0;startPgmTime=0;connectNum=0;startPgmNum=0;}
periodStart -> {startPgm6Num=0;connectNum=
connectNum+1; print "There is periodStart:connect() is be-
gin call "+ connectNum;}
```

startPgm6 -> {startPgmTime6=time(startPgm6); print "appear6connectNum:"+connectNum+"appear6time:"+ startPgmTime6;startPgm6Num=connectNum-startPgmNum;}

startPgm ->{startPgmTime = time(startPgm); startPgmNum=connectNum; print " appear5connectNum: " + connectNum+" appear5time :"+startPgmTime;}

endMainStart->{a2=time(endMainStart);print "endMainStartTime: " + a2 ;}

源程序执行结果和违例结果如图 3 所示：

图 3 源程序执行结果

Fig. 3 Results of source execution

4 结语

本文的监测方法是对事件进行跟踪（有利于捕获信息），以程序方法的开始和结束作为监测的执行点（有利于利用方法继承关系得到事件关联），并通过程序运行时产生的大量监测值，判断这些值的正确性，最终判定其行为是否符合预期行为。进行软件行为的动态监测时充分考虑了耗资问题，在判定准确性、执行效率、违例响应速度方面做了折中，从而在行为准确判定和软件正常执行间达到平衡，克服了非软件行为引起的故障。

参考文献：

[1] 郭长国，朱 俊，初 宁. 一种分布式软件运行时监控机制[J]. 计算机与数字工程，2008，36(11)：33-35.
Guo Changguo，Zhu Jun，Chu Ning. A Distributed Software Runtime Monitoring Mechanism[J]. Computer and Digital Engineering，2008，36(11)：33-35.

[2] 万灿军，李长云. 动态演化环境中可信软件行为监控研究与进展[J]. 计算机应用研究，2009，4(26)：1201-1204.
Wan Canjun，Li Changyun. Research and Development on Behavior Monitoring and Controlling of Trusted Software in Dynamic Evolution Environment[J]. Application Research of Computers，2009，4(26)：1201-1204.

[3] Avgustinov P，Tibble J，Bodden E，et al. Aspect for Trace Monitoring[C]//Proc of Formal App Roaches to Testing Systems and Runtime Verification. SharpCrafters：[s. n.]，2006：20-39.

[4] Kim M，Kannan S，Lee I，et al. Java-MaC：A Run-Time Assurance Tool for Java Programs[J]. IEEE Proceedings of Data Compression Conference，2005，23(10)：279-295.

[5] Fabiok，Manuel R. Monitoring，Security，and Dynamic Configuration with the DynamicTAO Reflective ORB[C]// Proc of Middleware 2000. NewYork：[s. n.]，2000：121-143.

[6] Ann Q，Gates，Steve Roach，et al. A Taxonomy and Catalog of Runtime Software-Fault Monitoring Tools[J]. IEEE Computer Society，2006，166(2)：220-235.

[7] Payne M. Automating Instrumentation：Identifying Instrumentation Points for Monitoring Constraints at Runtime. Master’s Thesis[D]. Texas：The University of Texas，2008：200-250.

[8] Havelund K，Pressburger T. Model Checking Java Programs Using Java PathFinder. International Journal on Software Tools for Technology Transfer[J]. IEEE Transactions of Software Engineering，2008，2(4)：366-381.

（责任编辑：罗立宇）