第33卷 第18期 计 算 机 工 程 2007年9月
Vol.33 No.18 Computer Engineering September 2007
·软件技术与数据库· 文章编号：1000—3428(2007)18—0048—03 文献标识码：A 中图分类号：TP391

面向 Java 的对象行为建模及可视化方法

贾晓霞[1]，吴 际[1]，李郭欢[2]，金茂忠[1]
(1. 北京航空航天大学计算机学院，北京 100083；2. 西南通信研究所，成都 610041)

摘 要：程序行为的描述和“可视化”方法是理解程序的重要手段。该文基于对象的方法调用和被调用序列，描述了对象生命周期内的行为，定义了 Java 对象生存期行为模型，并使用活动树进行可视化。实例分析表明，LBM 可以帮助用户理解不同对象行为间的差异。
关键词：对象行为建模；软件可视化；Java

Java-oriented Object Behavior Modeling and Visualization Method

JIA Xiao-xia[1], WU Ji[1], LI Guo-huan[2], JIN Mao-zhong[1]
(1. School of Computer, Beijing University of Aeronautics and Astronautics, Beijing 100083;
2. Southwest Communication Institute, Chengdu 610041)

【Abstract】 How to describe and visualize the program behavior is an important measure to comprehend the program. This paper presents a novel object behavior model named lifetime behavior model based on the method invocation and invocated sequence of object, and visualizes the model in LBMTree. The model and visualization is displayed through a demo program, which demonstrates that the LBM is helpful to understand the differences between the different objects.
【Key words】 object behavior modeling; software visualization; Java

如何更好地理解 Java 程序，提高 Java 程序性能，成为值得关注的问题。其中，如何描述和“可视化”对象行为，是一个重要的研究方向。

笔者针对 Java 对象的特点，基于对象方法的调用和被调用序列，提出了 Java 对象生存期行为模型(lifetime behavior model, LBM)，并对其进行了可视化(LBMTree)。

1 相关研究

在程序运行过程中，期望通过对程序行为的分析来理解或解决问题。对程序行为进行描述及“可视化”便是其中一种方法，相关的研究和工具非常多。

(1)BLOOM[1]是一个全新的支持Java的可视化系统。

(2)JIVE[2]是BLOOM系统的一部分，分别在类及线程级别中，记录并“可视化”程序行为。

(3)JOVE[2]则关注更细节的信息，在一定的间隔下，记录每个线程的基本块信息，并对信息进行累加。

(4)STOOP[3]是一个类似于BLOOM的系统。Georgia Tech 大学“程序可视化”工作小组开发了很多程序可视化工具，帮助用户理解程序行为[4,5]，类似的系统还包括IBM的PV[6]。

上述研究和工具并没有着力描述对象或者程序的行为；而笔者基于 Java 对象的特点，提出对象行为模型，描述了 Java 对象的行为，并对模型进行了“可视化”。

2 对象行为模型

Java 程序运行时，一个类往往会实例化出多个对象，每个对象都拥有自己的生存期活动逻辑。

对象生存期的行为逻辑可以由对象生存期行为模型来描述和定义。对于 Java 程序而言，对象行为可以用对象所响应的外部事件以及所发出的请求事件来刻画。设 o 为一个对象，LBM(o)为 o 的生存期行为模型，其可以形式化地定义为

$$LBM(o) := \langle e_1, e_2, \ldots, e_n \rangle,\ 1 \le i \le n$$

$$e_i := [signature, t],\ e_i \in EI(o) \cup EC(o)$$

$$signature := invocation@site\$Thread$$

其中，n 是 o 的生命周期内的事件个数；*Thread* 是事件发生时所在的线程。

任意 $e \in EI(o) \cup EC(o)$，e 定义为

$$[signature, t]$$

其中，*signature* 为 e 的内容；t 为 e 发生的时间点。

$$EI(o)=EIE(o) \cup EIX(o)$$

其中，EIE(o)为对象 o 的调用进入(Invocation Entry)事件集；EIX(o)为对象 o 的调用退出(Invocation Exit)事件集。

EIE(o)事件可描述为

$$[f>@a.g\$Thread,t_1]$$

EIX(o)事件可描述为

$$[f<@a.g\$Thread,t_2]$$

其中，a为发出请求或调用的对象；g为a发出此请求所在的方法；f是o的方法或接口名；t_1和t_2分别是事件发生的时间。

EC(o)则定义了对象 o 所发出的请求事件，一般，该类事件可以定义为

$$[b.m@n\$Thread,t]$$

其中，b 为被请求的对象；m 为请求事件的名称；n 指出了 o 发出此请求事件的场所；m 和 n 为典型的对象操作或接口；t 是事件发生的时间。

传统的 Profiler 工具也关注方法调用关系，如 JProbe Profiler 提供了方法调用链。方法调用链关注方法的调用关系，通常表示为一个方法是否调用了另外一个方法，以及调用的次数。可以看到，这种叠加的方法调用信息无法描述方法调用时，发生的上下文对象信息，即无法提供一个方法的

基金项目：国家自然科学基金资助项目(60373016)
作者简介：贾晓霞(1976－)，女，博士研究生，主研方向：软件测试，软件故障定位；吴 际，博士、讲师；李郭欢，硕士研究生；金茂忠，教授、博士生导师
收稿日期：2006-11-14 **E-mail**：xiaoxiajia@buaa.edu.cn

所属对象，也无法给出是哪个对象发出调用请求。由此可见，LBM(*o*)和传统 Profiler 工具的方法调用链有本质区别。

笔者采用例子程序 ThreadTest 来展示模型及可视化效果。ThreadTest 是一个典型的生产者-消费者问题，共有 4 个类：ThreadTest,Producer,Consumer,Soup。其中，Producer 和 Consumer 继承自 Thread，通过 notify,wait 对共享对象 Soup 进行访问。图 1 是原型工具显示的对象 Soup 的 LBM 片段。

图 1　对象 Soup 的 LBM 片段

其中，“ add > @ 273732112:Producer.run $ 273732112 ! 32 ”表示在程序运行 32ms 后，对象 Producer(对象 id 为 273732112)的方法 run 调用进入 Soup 对象的 add 方法；$表示该事件发生和 id 为 273732112 的线程有关，也就是 Producer 线程，这在后面的 LBMTree 中将予以说明。

3　LBM 的可视化

基于 LBM，笔者提出了 LBMTree 可视化模型和 3 种不同的可视化策略：面向线程的策略，面向对象交互的策略和面向方法的策略。LBMTree 用到的图标见表 1。

表 1　　LBMTree 图标表示

图标	*表示*	*图标*	*表示*
	类		方法
	对象		方法调用
	线程		调用位置

3.1　面向线程的 LBMTree

面向线程的 LBMTree 展示了对象在哪些线程中“ 被调用 ”和“ 活动 ”，其根节点为被关注对象，其子节点为根节点的活动场所(线程)，该子节点下的节点，则体现了关注对象的具体行为。其可视化算法如下：

```
LBMTree ThreadLBMTreeConstruct(lbm: LBM, o: Object)
{lbmTree = new LBMTree(o);
//以关注对象 o 为根创建 LBMTree
Foreach evt in lbm{
    ThreadInfo thread = lbm.ExtractThread(evt);
    if (thread != null){
       Node node = new Node(thread);
       lbmTree.Append(lbmTree.root, node);
    }}
//遍历 o 的 LBMTree，将所有包含 o 的活动的线程作为直接子
//节点，挂在根下
Foreach node in lbmTree.root.Children{
  EventList evtList = lbm.ExtractEvent(node);
//从 o 的 lbm 提取所有在根节点的子节点所代表的，在线程中发
//生的事件
   Foreach evt in evtList{
         Node internode = new Node(evt);
         LBMTree.Append(node,internode);
   }}
//依次遍历孩子节点，按在 LBM 中出现的时间顺序，提取当前
//根节点的活动事件，将事件的详细信息作为子节点，挂在当前节
//点下；}
```

对象 Soup 的面向线程的 LBMTree 的例子见图 2。线程 273673056 为 main 线程；线程 273732112 为 Producer 线程对象；线程 273736168 为 Consumer 线程对象。

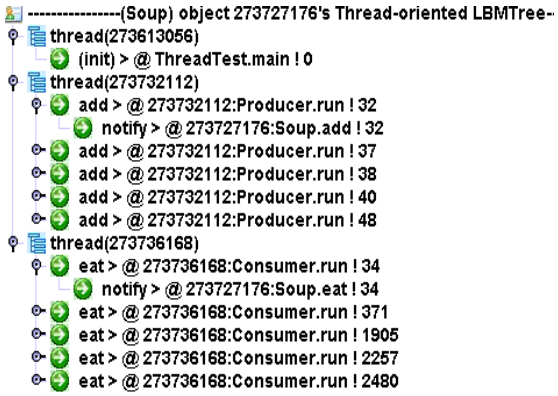

图 2　对象 Soup 的面向线程的 LBMTree

3.2　面向交互的 LBMTree

在面向交互的 LBMTree 中，根节点为关注对象，其子节点为与其交互的对象，叶子节点则为具体的调用事件。其可视化算法类似面向线程的策略，这里不再赘述。图 3 中，例子程序的 Main 类 ThreadTest 创建了该 Producer 对象，并调用其 start 方法。

图 3　Producer 对象的面向交互的 LBMTree

3.3　面向方法调用的 LBMTree

对任一关注对象，给出“ 以被调用的方法为中心 ”的视图，称为面向方法调用的 LBMTree。其根节点为关注对象，子节点为该对象被调用的方法，叶子节点则为方法被调用的位置。

图 4 中的叶子节点，显示了其父节点所代表的对象方法，被调用的位置和调用时间。

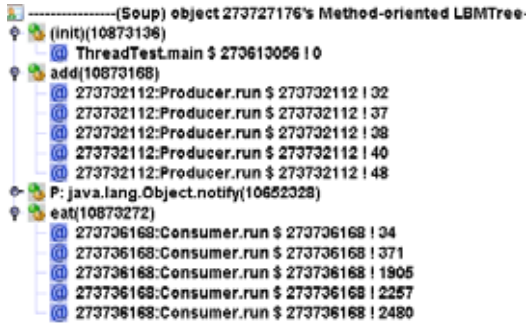

图 4　Soup 对象的面向方法调用的 LBMTree

多种角度的 LBMTree 向用户展示了丰富的信息，体现了对象生命周期内的行为。

4 工具实现

JVMPI (Java virtual machine profiling interface)是 Java 提供的用以收集程序执行性能数据的函数调用接口。一方面，虚拟机通知剖析代理各种事件；另一方面，剖析代理通过 JVMPI 控制和请求更多信息。

笔者所要实现的原型系统的基础是 JVMPI，Myprofiler 工具的结构见图 5。

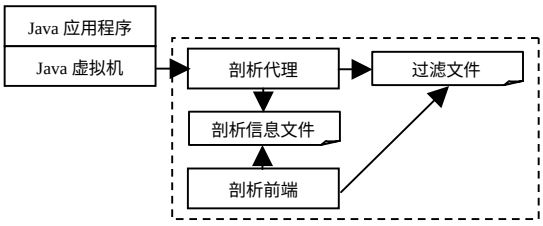

图 5 Myprofiler 工具结构

原型工具的核心部分是剖析代理和剖析前端。剖析代理主要负责测试数据的获取、组织和转储。其中，对测试数据的获取是通过 JVMPI 来完成的。图 6 便是为信息转储而定义的 XML Schema 文件格式的一部分。剖析前端负责在程序运行完毕后，读入剖析信息文件，把剖析信息可视化的呈现给用户。

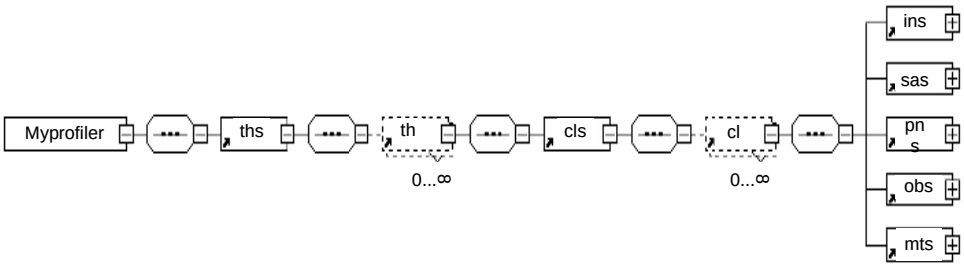

图 6 XML Schema 文件格式

5 实例分析

传统情况下，Java 程序员只能通过在程序中添加一些输出语句，追踪程序的执行和特定变量的取值及其变化，并基于这些信息对程序进行调试。JDK1.2 支持 JVMPI，基于 JVMPI 进行 profiling 的工具大批出现。本小节通过实验证明，在解决特定问题时，LBM 模型对 JProbe 提供的信息进行了有益的补充。

QESat/Java(quality engineering software analysis and testing tool)是北航软件所开发的 Java 程序理解和辅助测试工具，可以进行代码审查，动态覆盖率测试等工作。在 QESat/Java 的开发过程中，可以应用 JProbe 来帮助理解“程序对内存的使用”以及解决潜在的性能问题。

用 JProbe Memory Debugger 对 QESat/Java 进行分析，部分结果见图 7。seibuaa.safepro.util.path 类累积了 6 个实例。对其中的每个实例，Jprobe 提供了一些信息，如其创建时间、引用者(referrer)、引用对象(reference)、创建位置等，这些信息有助于程序员的理解及对类 path 实例的使用。

Package	Class	Count	Count Change	Cumulative Count	Aggregate Memory
Total		**1,203**	**+955**	**9,157**	**4,577,968**
seibuaa.safepro.util	ResourceReader$ReaderHander	1 (0.1%)	+1	1 (100.0%)	136 (0.0%)
seibuaa.safepro.util	ResourceReader	2 (0.2%)	+2	2 (100.0%)	159,392 (3.5%)
seibuaa.safepro.util	Path	8 (0.7%)	+6	177 (4.5%)	744 (0.0%)

图 7 JProbe Memory Debugger 分析结果

上述信息从一定的角度展示了对象信息，但对于对象在在整个生存期内的行为；对象如何和其他对象发生相互调用；同一类的对象间的行为特征和相互间的差异等，JProbe 并不能很好地对其进行刻画。

LBM 模型描述了对象整个生命周期内的行为。对于图 7 的情况，在用户发现该段程序执行时间内，Path 的实例有所增加，和预期不符。除了可以利用 JProbe 提供的信息之外，LBM 所描述的对象生存周期内的行为和 LBMTree，也可以帮助用户深入理解 Path 类的每个对象的行为。

6 总结

在进行软件故障定位、分析 Java 程序内存低效使用的研究中，笔者对已有 Java 程序行为的定义和表示进行了分析，并提出了对象生存期行为模型和相应的“可视化”方法。对样例程序的分析可以看到 LBM 及 LBMTree 提供了很多有价值的信息。实例分析也说明了 LBM 的使用场景。基于 LBM，笔者提出了描述对象活动状况的活动谱图模型(activity spectrum model)，以便分析程序中对象的内存使用性能。同时，基于 LBM，下一步将研究基于对象行为模式的软件故障定位技术。

参考文献

1 Reiss S P, Renieris M. The Bloom Software Visualization System, Software Visualization——From Theory to Practice[M]. Cambridge, MA: MIT Press, 2002.

2 Reiss S P, Renieris M. Demonstration of JIVE and JOVE: Java as It Happens[C]//Proceedings of the 27th International Conference on Software Engineering. 2005.

3 Brown R, Jorgensen J, Wang Q, et al. STOOP: The Sable Toolkit for Object-oriented Profiling[C]//Proc. of Object-oriented Programming Systems, Languages, and Applications Conference. 2001-10.

4 Jerding D, Stasko J T, Ball T. Visualizing Interactions in Program Executions[C]//Proceedings of 19th International Conference on Software Engineering. 1997: 360-370.

5 Kraemer E. Software Visualization: Programming as a Multimedia Experience[M]. Cambridge, MA: MIT Press, 1998.

6 Kimelman D, Rosenburg B, Roth T. Visualization of Dynamics in Real World Software Systems[M]. Cambridge, MA: MIT Press, 1998.