基于事件处理的 JAVA 切片

JAVA Program Slicing based on Event Handling

王　欢　吴伟民　（广东工业大学 计算机学院 广东 广州 510006）

摘　要： Event 是一类特殊的对象，它在 Java 中与被创建的控件相关联，并通过对应的监听器接收消息而激发相应的事件处理程序。本文提出了一种合理的新方法，在系统依赖图中表示事件处理模块，利用图可达性算法计算事件处理切片。

关键词： 程序切片 JAVA 系统依赖图 事件处理 监听依赖 隶属依赖 事件依赖

程序切片是一种程序分析技术，最初的概念是由 Mark Weisert 于 1979 年在他的博士论文中提出的。它由程序中直接或间接影响某个兴趣点变量值的语句集合组成。最初，程序切片的计算需要在程序的控制流图上建立数据流方程，然后，求解数据流方程来获得相应的程序切片。由于该方法只能解决包含基本结构的程序的切片计算，而且效率较低，J. Ferrante 等人提出基于程序依赖图的图可达性算法，从而解决了过程内后向切片的计算问题[1]。随后，S. B. Horiwitz 等人提出利用系统依赖图计算过程间切片的两步遍历图可达性算法。从而解决了包含多个过程程序的切片计算问题[2]。由于面向对象语言的不断盛行，Horrold[3,4] 等人将系统依赖图扩充成面向对象系统依赖图，用于描述面向对象程序，利用两阶段的图可达性算法遍历面向对象系统依赖图来计算程序切片。

由于事件处理的特殊性，传统程序切片算法都没有考虑事件处理。实际上，事件的产生会影响程序执行的控制流，在切片中就应当考虑事件处理所带来的数据和控制依赖。本文综合了前人方法的优点，提出了一种新的方法在面向对象系统依赖图中表示事件处理模块，避免了前人在事件处理上存在的问题，并能精确地表示事件对象的数据依赖关系，从而提出一种可以进行事件处理的 Java 程序切片方法。

1　JAVA 事件处理机制分析

事件处理机制是可视化程序编程所特有的，它是基于特定编程语言实现的。JAVA 处理事件的方法是基于授权事件模型的，这种模型定义了一致的事件产生和处理机制：一个控件产生一个事件并把它送到一个或多个的监听器。在这种机制中，监听器简单地等待，直到它收到一个事件。一旦事件被接受，监听器将处理这些事件，然后返回。一个用户接口元素可以授权一段特定的代码处理一个事件。在授权事件模型中，监听器为了接受一个事件通知必须注册，因此，通知只被发送给那些想接受的它们的监听器那里。在授权事件模型中，一个事件是一个描述了事件源的状态改变的对象。它可以作为一个人与图形用户接口相互作用的结果被产生。一个控件是一个产生事件的对象。当这个对象内部的状态以某种方式改变时，事件就会产生。控件可能产生不止一种事件。一个事件源必须注册监听器以便监听器可以接受关于一个特定事件的通知。每一种事件有它自己的注册方法。一个事件监听器是一个在事件发生时被通知的对象。它有两个要求。首先，为了可以接受到特殊类型事件的通知它必须在事件源中已经被注册。第二，它必须实现接受和处理通知的方法。下面是示例程序的运行结果（图 2 所示）。该程序用于将发送文本框中的信息发送到上面的显示文本框中。

2　扩展 JAVA 系统依赖图

JAVA 系统依赖图（JSDG）是用于描述 JAVA 程序的系统依赖图。JSDG 是过程依赖子图、类依赖子图、类层次子图、控制依赖子图和数据依赖子图等几种表示方法的并集. 过程依赖子图：把一个过程表示成一个图，其中结点表示语句或谓词表达式，数据依赖边表示语句或表达式之间的数据流，控制依赖边表示一个语句或表达式执行时依赖的控制条件. 类依赖子图：类依赖子图确定一个类中的控制依赖和数据依赖关

系. 在类依赖子图中，方法由过程依赖子图来表示. 类依赖子图也包含一个类入口结点，并通过类成员边把它和类中每个方法的入口结点连接起来. 类层次子图：用图形来表示类间的继承关系. 类层次子图包含每个类的首部结点和在类中定义的每个方法的首部结点. 类层次子图中的边把每个类的首部结点和与其具有继承关系的类的相应的首部结点连接起来。

```
.....
JPanel p=new JPanel();
jtfName=new JTextField(10);
p.add(new JLabel("Title: Write Once, Run Anywhere."));
.....
jbClear.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){
                jtaInput.setText("");

       });
Action sendMessage = new AbstractAction() {
        public void actionPerformed(ActionEvent e){
                        replaceMessage();
                }};}
private void replaceMessage(){
        .....
                }
```

图 1　示例程序片段

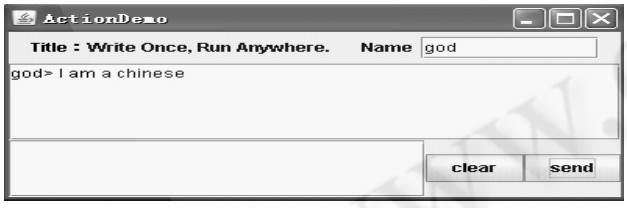

图 2　示例程序运行结果

由于在 JSDG 中要加入表示事件处理的模块，因此，需要在原 JSDG 中加入事件依赖图。事件依赖图包含以下元素：用于表示事件的事件节点，用于表示监听器的监听器节点，用于关联控件和其所属事件的隶属依赖边，用于表示事件和监听器之间监听依赖边，用于关联控件和监听器之间的事件依赖边。由于消息事件并不存在于源程序中，它只是在程序执行时由用户产生的，因此，在程序中它属于一种虚拟节点。图 3 所示的是示例程序的 JSDG 的一个组成部分，由于篇幅有限，其中省略了相互调用方法之间传递的参数节点和相应的依赖边，仅仅反应了用于描述事件处理的节点和及其存在的各种依赖关系。

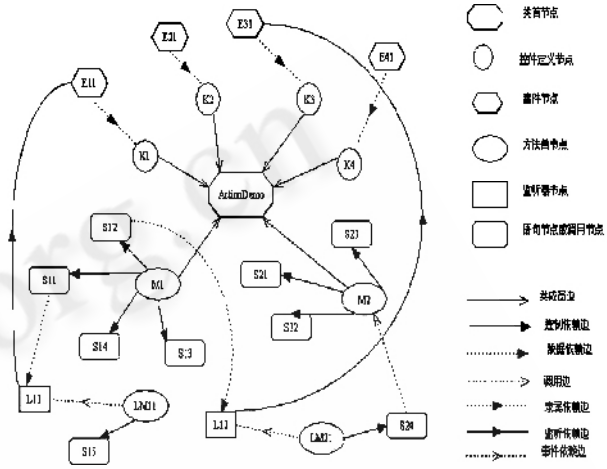

图 3　示例程序的部分 JSDG

3　事件处理的表示

在传统的切片计算中所定义的切片准则是一个二元组 <n,V>，其中 n 是程序中的一条语句，V 是程序 P 中变量的一个子集。程序中的语句之间存在的依赖关系仅包括数据和控制依赖。由于事件处理具有特殊的处理机制，因此，利用原有切片准则，显然不适用于计算事件处理切片。为了便于计算事件切片，需要补充如下定义：

令 P 表示一个 JAVA 面向对象程序，C 表示程序 P 中所有控件定义节点的集合，E 表示对应于 P 中所有控件发生的事件集合，L 表示 P 中所有控件的监听器的集合，V 是 P 中所有变量的集合。

定义 1　（控件切片准则）程序 P 的控件切片准则是一个三元组（c,Es,Vs）其中 $c \in C, Es \in E, Vs \in V$。

定义 2　（监听依赖）若 $Y \in L, X \in E$，且 Y 监听 X 的发生，则称 Y 监听依赖依赖于 X，记作 $Y \xrightarrow{L} X$。其中，当添加了监听器 Y 的某个源，产生一个事件 X 时，Y 接收该消息并触发执行相应的处理程序。

定义 3　（隶属依赖）若 $Z \in C, X \in E$，且 X 可以发生在 Z 上，则称 Z 隶属依赖于 X，记作 $Z \xrightarrow{B} X$。其中 Z 就是用来改变源 X 的状态的某种操作。

定义 4 （事件依赖）若 $Z \in C, X \in E, Y \in L$，且 $Z \xrightarrow{B} X$，$Y \xrightarrow{L} X$，则称 Y 基于 Z 监听依赖于 X，记作 $Y \xrightarrow{Z} X$。

4 计算程序切片

给定切片准则 <c,Ms,Vs>，如果节点 c 是普通语句节点，不包含任何控件的定义，则 Ms 为空集，说明控件切片准则已经退化称传统切片准则 <n,Vs>。在这种情况下，可以以用传统的切片算法解决切片计算问题。如果节点 c 是包含控件定义的节点，可以基于 JSDG 利用图可达性算法计算包含事件处理的切片。第一步，以控件定义节点 c 为起始节点，沿着隶属依赖边、监听依赖边和事件依赖边逆向遍历，并标记所有在 JSDG 中被遍历过的节点，并删除其中的事件节点和监听器节点；第二步，以上一步所标记的节点集合中的方法节点为起点，利用传统的 tow－pass 切片算法来计算程序切片[3]。计算所得的切片就是第一步和第二步所标记的节点的并集。

切片算法如下：

输入：程序 P 的 JSDG 和切片准则 p <c,Es,Vs>

输出：程序切片 SliceList

```
Procedure computingslice(G,p)
Declare
G: a Java system dependence graph
p: a slicing criteria by the form of <c,Es,Vs>
worklist,tempLs,tempCs,tempEs: a set of vertices in G
Begin
    SliceList →∅
    mark c as visited
    insert c into SliceList
    for each node m there is an edge
    mc --B--> do
    Insert m into tempEs
    od
    tempEs←tempEs∩Es
    for each node s there is an edge
    s --L--> m1 and m1 exists in tempEs do
    Insert s into tempLs
    od
    for each node t there is an edge
    t --c--> s1 and s1 exists in tempLs do
    Insert t into SliceList
    od
    WorkList←SliceList
    for each x in WorkList do
    temp ← <x,Vs>
    ∪SliceList←Two－pass(G,temp) SliceList
    od
  end
end
```

将事件处理模块的切片归结到响应该事件的处理方法上，并以该方法的头节点构建新的切片准则。这样，就可以应用传统的切片算法，计算事件处理切片问题。此方法既没有增加程序代码，又没有增加新的变量。所以，由传统切片算法计算出来的程序切片依然是源程序切片，本文采用的 two－pass 算法详见文献[3,4]。

5 结束语

本文针对 Java 事件处理机制的特点，将依赖于控件所存在的事件和监听器视为虚拟节点，合理地表示了由于事件处理所带来的隶属、监听和事件依赖关系；对原有的 JSDG 进行了扩展，使之能够表示事件处理机制；合理地改进了已有的程序切片算法，为包含事件处理的 Java 程序提供了较为准确的切片方法。

参考文献

1 Ferrante J, Ottenstein K J, Warren J D. The program dependence graph and its use in optimization. ACM Transactions on Programming Languages and Systems, 1987, 9(3):319－349.

2 Horwitz S, Reps T, Binkley D. Interprocedural slicing using dependence graphs. ACM Transactions on Programming Languages and Systems, 1990, 12(1):26－60.

3 Larsen L. Harrold M. Slicing object oriented software. In 18th International Conference on Software Engineering, 1996, (3):495－505.

4 Liang D. Harrold M. Slicing objects using system dependence graphs. International Conference on Software Maintenance, 1998, (11):358－367.