一种静态的 Java 程序函数调用关系图的构建方法①

古 辉，李荣荣

(浙江工业大学 计算机学院，杭州 210023)

摘 要：在程序理解中，函数之间的调用关系是程序理解研究的重要内容。一个函数往往代表了一种具体功能或问题求解的实现，构建出函数调用图有助于对程序的理解。以 JAVA 语言为研究对象，介绍了几种函数调用图的构建方法，并比较了它们的优劣性，并在此基础上提出了一种函数调用图的构建方法

关键词：程序理解；函数调用图；构建函数调用图

A Static Function Calls Relationship Chart Building Methods of Java

GU Hui, LI Rong-Rong

(School of Computer, Zhejiang University of Technology, Hangzhou 210023, China)

Abstract: In program understanding the research of function relationship occupies a very important position. A function often represents a specific performance or problem solving realization. The constructing of function invocation figure is very helpful to the understanding of program. This paper uses Java as research object, introduces several construction methods of function's call graphics and compares with their inferiority and proposed in this foundation a function call figure building methods

Key words: program understanding; functions'call graph; construction of the function call graph

1 引言

每个大型的程序都是通过对函数的组织和调用来实现整个程序的功能要求。因此，掌握程序中各个函数之间的关系以及每个函数的工作流程对理解程序是非常有帮助的。

函数调用图是对程序中函数调用关系的一种静态描述。在函数调用图中，节点表示函数，边表示函数之间的调用关系[1]。通过函数调用图，可以了解程序中的函数以及函数之间的条用关系，从而辅助理解程序的基本结构及其功能。

由于面向对象程序设计中的多态性以及 JAVA 语言语法规则，在很多情况下无法在编译期间确定某些函数调用点中接受对象的实际类型，所以 JAVA 程序的函数调用图只能大致表示出实际运行时函数的调用关系。如何提高函数调用的解决效率，使构建的函数调用图能够更准确的反应实际执行程序时函数之间的真正的调用关系，一直是程序理解领域所关注的热点和难题。

2 JAVA程序的函数调用图典型实现方法

类层次分析（CHA，Class Hierarchy Analysis)、快速类型分析（RTA，Rapid Type Analysis）和独立的方法和域可用类型集合分析（XTA，Separate sets for method and fields Analysis ,对 RTA 的改进算法)，以上三种是典型的 JAVA 程序的函数调用图的构建算法。

前两种技术是属于流不敏感分析（不用考虑类型传播直接计算函数调用点中接受对象的运行时可能类型的分析方法)，这种分析方式容易实现，且算法较简单。后来，F.Tip 教授在此基础上又提出了对 RTA 方法的改进算法，即 XTA，进一步提高了精度。

2.1 类层次分析

类层次分析首次是是由 J.Dean 教授提出，它是由

① 收稿时间:2011-08-11;收到修改稿时间:2011-09-19

© 中国科学院软件研究所 http://www.c-s-a.org.cn

程序的类层次图得来的。在类层次图中节点表示程序中声明的各个类，边表示他们的继承关系。根据得到的类层次图，如果一个对象的声明类型是父类，在运行时的可能类型就包含父类及其所有的子类。对某个特定的函数调用点来说，可能的类型集合就包括父类以及子类中所有的跟该函数有关的方法。

2.2 快速类型分析

RTA 分析法是由 David F.Bacon 教授提出，是在类层次分析的基础上进行改进得来的。它的主要思想是根据程序中的实例化信息来进一步约减接收对象的可能类型集合，从而提高构建函数调用图的效率。

2.3 XTA

XTA 是对快速类型分析的一种有效改进算法，是采取增量式的方式来分析。它对程序中的每个声明变量和函数都给定一个可用类型集合。每个可用类型集合包括在该函数中实例化的对象类型以及通过参数传递从调用该函数的可达函数中传进来的可用类型，后一部分可用类型是指调用者的可用类型集合与被调用函数的形参的声明类型及其子类型集合的交集。同时如果被调用函数存在返回类型，那么该函数的返回值类型也要添加到调用该函数的可达函数的可用类型集合中来。

3 一种XTA方法的改进算法

类型传播是一种只关注对象类型的简易数据流分析方法，包括过程内的类型传播和过程间的类型传播[2]。过程内的类型分析是指在类型的传播依赖于函数体内从类型事件到函数调用点间的简易数据流分析，从而改进对于函数调用点中的接受对象类型集合的约近方式[3]。

本改进算法是根据类型传播的原理来设计的，且主要是过程内的类型传播。

3.1 算法设计

算法中，R 表示可达函数的集合，用 StatementType（e）表示变量 e 的声明类型；SonTypes(t)表示类 t 的所有子类集合；StaticLookup（C,m）表示在类 C 中静态地查找其是否有声明为 m 的函数； S_M 是对程序中的每个函数 M 所给定的一个特殊集合；S_X 是对每个域 x 给定的一个特殊集合；ReturnType（M）表示函数 M 的返回类型；ParamTypes（M）表示函数 M 的参数的声明类型的集合。对于可达函数 M 中的每个实例化类 C 给定一个特殊的集合 R_C，R_C 表示函数 M 中类 C 的实例化信息可达到的变量的集合。

步骤 1：从函数 main()开始，main∈R

步骤 2：对于每个函数 M 以及 M 中的每个函数调用点 e.m()，如果在 M 中存在一系列赋值语句：v=new C( ),x1=v,x2=x1,……，$x_n = x_{n-1}$，e= x_n，即类 C 的实例化语句存在于 M 中，且通过过程内的类型传播，类 C 的实例化信息可达到 e，那么：（M∈R）=>（e∈ Rc），这就代表变量 e 属于集合 Rc

步骤 3：对于每个函数 M，如果 M 中包含 new C（）这样的实例化信息，那么（M∈R）=>（C∈ S_M），这是初始化 S_M 为在可达函数 M 中实例化的对象类型

步骤 4：对于每个函数 M，以及 M 中的每个调用点 e.m(),如果存在类 C∈SonTypes（StatementType（e）），并且有 StaticLookup（C,m）=M’，如果可用类型集合 S_M' 包含目标函数的类，且对于该函数调用点中的接受对象来说，通过过程内的类型传播，实例化信息是可达的，那么符合条件的目标函数 M’可以加入可达函数的集合 R 中，同时可以通过参数传递将调用函数的可用类型集合与被调用函数的形参声明类型及其子集的类型集合的交集传递给被调用函数，作为其可用类型集合的子集，又如果被调用函数的返回值类型不为空，则可以将其与被调用函数的可用类型集合的交集作为子集传递给调用函数的可用类型集合。对于在调用函数中的实例化对象类型，它同时也属于被调用函数的可用类型集合

步骤 5：在每个函数 M 中，如果存在对域 x 的读取，那么：（M∈R）=>($S_X \subseteq S_M$)，即代表 x 可能指向的对象类型集合是该可达函数类型集合的子集

步骤 6：在每个函数 M 中，如果存在对域 x 的写入，那么:（M∈R）=> （SonTypes（StaticType（x））∩ $S_M \subseteq S_X$，即如果可达函数中有写入 x，那么 x 可能指向的对象类型集合包含 x 的声明类型及其子类型集合与该可达函数的可用类型集合的交集。

3.2 改进算法的应用

假设程序实例中有五个类 A、B、C、D 和 Test，如图 1 所示。

```
class A {
    static A e;   //定义了类 A 的对象 a
    void m(){      //类 A 中的方法 m
            A a=new A();
```


© 中国科学院软件研究所 http://www.c-s-a.org.cn

```
            e=a;
            System.out.println("is A");
             }
      }
class B extends A{   //类 B,是类 A 的子类
      void m(){              //类 B 中的方法 m
      System.out.println("is B");
                }
}
class C extends B{   //类 C，是类 B 的子类
      void m(){              //类 C 中定义的方法 m
     System.out.println("is C");
            }
}
class D extends B{    //类 D，是类 B 的子类
      void m(){             //类 D 中的方法 m
     System.out.println("is D");
            }
}
class Test(){    //Test 类，测试这几种方法的优劣性
      public static void main(String args[]){    //主函数
      A f=new A();        //定义了一个类 A 的实例 f
      B b=new B();        //定义了一个类 B 的实例 b
      f(e);           //调用 f 方法
      g(b);           //调用 b 方法
}
Static void f(A a2){    //定义了方法 f，调用了 m 方法
      a2=b;
      e=a2;
      e.m();
      }
Static void g(B b2){    //定义了方法 g，调用 m 方法
      B b3=b2;
      b3=new C();
      b3.m()
      }
```

可以看出，类 C 和 D 都继承于 B，而 B 继承于 A，且在四个类中都定义了 m 方法，用 M1、M2、M3 和 M4 表示。在类 Test 中的 main 方法中实例化了 A、B，调用了 f 和 g 函数。

首先用类层次分析法分析：从 main 函数开始，根据类层次图（如图 1），得到图 2 所示的函数调用图；根据快速类型法的原理，由于程序中只实例化了类 A、B 和 C,那么 e.m()的可能类型集合就约减到 A、B 和 C，b3.m()就约减到 B 和 C，如图 3 所示。

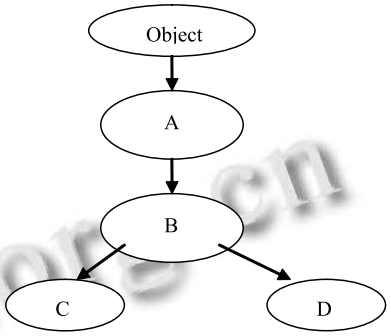

图 1 类的层次图

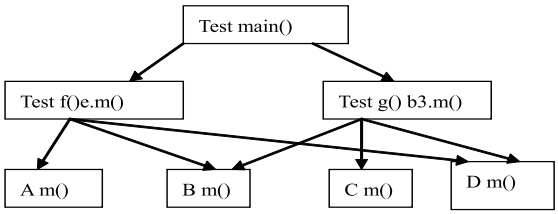

图 2 类层次分析得到的函数调用图

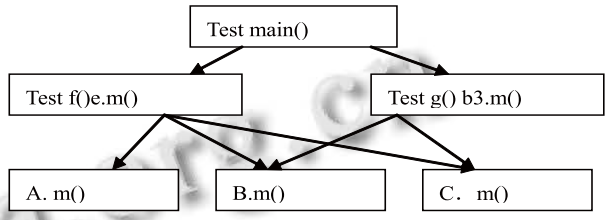

图 3 快速类型分析得到的函数调用图

再用 XTA 方法分析：在 main 中实例化了类 A 和 B,其初始可用类型集合有 A、B。对函数调用点 e.m()来说，e 的类型是 A， B 和 C 是 A 的子类，即 e 在运行时的可能类型集合包含 A 和 B,所以它可能调用的函数就有 M1、M2，又因为 M1、M2 的返回类型都为空，所以没有对可用类型集合构成影响，那么此函数调用点的目标函数就包括 A、B 中的 m()方法。而对 b3.m()来说，因为包含了类 C 的实例化语句，所以可能类型集合就包括了 B 和 C，且返回值为空，不对其构成影响，所以可能的目标函数就有 M2 和 M3（如图 4）。

最后用改进的 XTA 方法分析，考虑 main 函数中类 A、B 和 C 的实例化信息对于接受对象 e 的类型可

© 中国科学院软件研究所 http://www.c-s-a.org.cn

达性，对于 e 可达就说明在函数 M 中存在一系列这样的赋值语句：v=new C(),x1=v,x2=x1,……，$x_n = x_{n-1}$，e= x_n。显然，在 main 中，类 B 的实例化信息对 e 而言是可达的，而 A 和 C 是不可达的，所以据此可知，e 的运行时的可能类型就只包括 B，那么只有类 B 中的 m 函数是可达的。从以上分析，我们可以看到改进的 XTA 方法能够取得最好的效果。对 b3.m()这个函数调用点来说，因为没有类型传播，所以结果跟 XTA 是相同的（如图 5）。

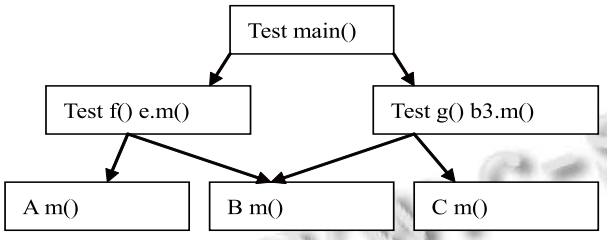

图 4　XTA 分析得到的函数调用图

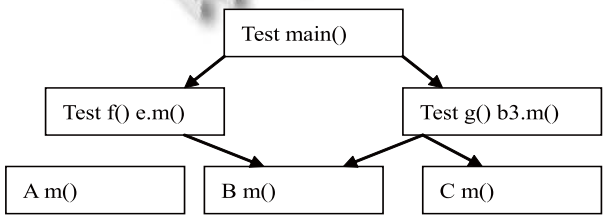

图 5　改进算法得到的函数调用图

4　结语

在类层次分析、快速类型分析和 XTA 三种的构建函数调用图的算法中，类层次分析是基础。这三种方法比较起来，XTA 方法是对快速类型分析的一种有效改进，能够更好的满足实用性和程序扩展性的要求，也获得了很好的分析精度。但本文所提到的对 XTA 算法的改进，在 XTA 的基础上，能够更好的使函数调用图得到归类，提高了绘制函数调用图的效率。

参考文献

1 Ryder BG. Constructing the call graph of a program. IEEE Trans. on Software Engineering, 1979,5(3):216– 226.

2 Gagnon EM, Hendren LJ, Marceau G. Efficient inference of static types for Java bytecode. Static Analysis Symposium 2000. Santa Barbara, 2000: 199–219.

3 Diwan A, Eliot J, Moss B. Simple and effective analysis of statically-typed object-oriented programs. Proc. of the Conference on Object-Oriented Programming Systems, Languages, and Applications. New York, l 996: 292–305.

(上接第 215 页)

做法切实可行。

参考文献

1 GeneralPlus. GPL32600A Programing Guide. Taiwan: GeneralPlus, 2009.

2 LaMothe A.Windows 游戏编程大师技巧.沙鹰,译.第 2 版.北京:中国电力出版社,2004.

3 李志敏.基于嵌入式平台的 2D 游戏引擎的研究与实现[硕士学位论文].武汉:武汉理工大学,2006.

4 孙乾恒.基于凌阳μ'nSPTM 系列单片机的电视游戏设计与实现.西安:西安电子科技大学,2009.

5 GeneralPlus. G+ Director User Guide. Taiwan: GeneralPlus, 2009.

6 Dalmau DSC. Core Techniques and Algorithms. New Riders Publishing, 2003.

7 Labrosse JJ.嵌入式实时操作系统μC/OS-II.邵贝贝等,译.第 2 版.北京:北京航空航天大学出版社,2005.

8 李玉刚等.嵌入式操作系统 uC/OS-II 在 ARM 上的移植研究.微计算机信息,2010,8(2):97–98.

9 李晓玫,杨小平,等.基于μc/os-Ⅱ的 I/O 系统的设计与实现.微计算机信息,2010,12(2):77–79.

10 王田苗.魏洪兴.嵌入式系统设计与实例开发.第 2 版,北京:清华大学出版社,2008.

© 中国科学院软件研究所 http://www.c-s-a.org.cn