基于数据切片度量JAVA内聚性*

李必信, 朱 平, 谭 毅, 李宣东, 郑国梁

(南京大学 计算机软件新技术国家重点实验室,江苏 南京 210093);
(中国科学技术大学 计算机科学技术系,安徽 合肥 230056)
E-mail: zhenggl@nju.edu.cn; libx@seg.nju.edu.cn
http://www.nju.edu.cn

摘要: 面向对象的程序切片在程序分析、程序理解、软件测试和调试以及软件维护方面有着广泛的用途.首先建立了抽象数据切片和类内切片的概念,然后基于这两种切片讨论了JAVA语言中存在的内聚问题,通过分析这些切片与数据、方法、类之间的关系来度量数据、方法以及类的内聚性问题.

关键词: 抽象数据切片;类内切片;数据类聚;类内聚;JAVA
中图法分类号: TP311 **文献标识码**: A

软件开发的目标是开发出高内聚和低耦合的系统.内聚应该是模块成分之间的相互关系.一个高内聚的模块就是具有一个基本功能的模块.一个内聚性很好的模块是很难被分离的.根据内聚的等级把内聚分类,其中一致内聚是最低级的内聚,也是软件开发者最不想要的内聚;功能内聚是最高级的内聚,也是开发者追求的内聚.内聚标志着一个模块内各个元素彼此结合的紧密程度,也就是说,它度量单个模块所完成的诸项任务在功能上相互关联的程度.软件设计力求做到高内聚,理想内聚的模块只完成一项相对完整的任务.通常,中等程度的内聚也是可以采用的,而且效果和高内聚差不多,低内聚不理想,一般不使用.

内聚和耦合是密切相关的.模块内的高内聚往往意味着模块间的松耦合.内聚和耦合都是进行模块化设计的有力工具,但实践表明内聚更重要,应该把更多注意力放到提高模块的内聚度方面.在结构化程序设计中,主要有几种内聚,按照内聚度从低到高的顺序排列,它们是:偶然内聚、逻辑内聚、瞬时内聚、过程内聚、通信内聚、顺序内聚和功能内聚.

Java语言是一种面向对象语言,其基本语言构造是类,类由方法和成员变量构成,在Java中是不存在自由函数或自由域的.实现功能的唯一办法是在一个类内部,所以度量的主体就是这些Java类构造.为了研究面向对象程序设计中的内聚问题,特别是像Java语言这类纯面向对象语言的内聚问题,这里引入了数据内聚和类内聚的概念.同时,我们使用数据切片和类切片来作为我们内聚度量的基础.

类似于过程性软件,我们可以把切片的相关概念运用到面向对象软件的属性度量.另外,内聚似乎是最适合切片分析的属性.与在过程范型内使用一样,内聚是一种可以被直接运用到面向对象世界的方法.功能内聚是过程代码的一个合适度量(这里,基本元件单元是过程和函数).在面向对

* 收稿日期: 2000-04-12; 修改日期: 2000-07-12
基金项目: 国家863青年基金资助项目(863-306-QN2000-2)

作者简介: 李必信(1969-),男,安徽庐江人,博士,主要研究领域为面向对象支撑技术及其环境和工具,程序理解;朱平(1978-),男,江苏扬州人,硕士生,主要研究领域为程序切片;谭毅(1978-),男,湖南衡阳人,硕士生,主要研究领域为程序切片;李宣东(1963-),男,湖南邵东人,博士,副教授,主要研究领域为面向对象技术,形式化技术;郑国梁(1937-),男,浙江桐乡人,教授,博士生导师,主要研究领域为软件工程,面向对象技术.

象的软件中,基本设计单元是类,它是成员变量和方法的集合体.功能内聚不能直接应用到类中.Fenton 认为,我们这里关心的应是另一种形式的内聚,即数据内聚,而不是功能内聚[1].文献[2]研究了各种程序切片技术,并提出分层切片思想.在本文中,我们基于程序切片来研究面向对象的内聚度量,一种方法是对功能内聚的直接扩展;另一种方法是从 Chidamber 和 Kemerer 的类内聚缺乏度(lack of cohesion in methods,简称 LCOM)的度量发展而来的.在这两种方法中,类是基本单元,成员变量是连接类中方法的"粘合剂".两种方法所不同的是分析的粒度.

本文首先计算 Java 程序的抽象数据切片和类切片,通过这些切片和数据、方法、类之间的关系来度量数据、方法、类的内聚性问题.我们认为这种方法起到了很好的效果.

1 数据切片

在文献[3]中,Mark Weiser 定义了几种基于切片的度量.Longworth 首先把切片应用到内聚度量中[4].Thuss 通过使用度量切片(metric slice)的概念消除了 Longworth 注意到的某些不一致性[5].一个度量切片考虑使用和被使用两种数据关系,也就是 Horwitz 等人所说的后向切片和前向切片的并[6].后向切片的计算从方法的最后一条语句开始,前向切片的计算是从后向切片的"顶"开始的.

为了分析修改对切片度量的影响效果,我们使用数据标记(data token,包括变量、常量定义和引用等)而不是语句作为基本单元来修正度量切片这个概念.我们称这种基于数据标记的切片是变量级切片或数据切片.

```
1. SumAndProduct{
2.     int [SumN]=[0];
3.     int ProdN=1;
4.     int [N],[I];
5.     for ([I]=[1];[I]<=[N];[I]++)
       {
6.       [SumN]=[SumN]+[I];
7.       ProdN=ProdN*I;
       }
   }
```

Fig. 1 The parts with pane shows data slice of *SumN*

图 1 带方框部分表示 *SumN* 的数据切片

使用数据标记作为切片的基础确保任何有影响的变化都至少会在方法的一个切片中引起改变.我们认为,一个有影响的变化是能够对方法的内聚产生影响的任何变化.这里,有影响的变化包括加入代码、删除代码或改变在给定上下文中的变量.这些变化的每一种都会导致一个数据切片的改变.非形式地说,我们认为数据标记 v 的一个数据切片是所有数据标记的序列,这些数据标记在 v 的前向和后向切片的语句中.我们为方法的每个输出计算一个数据切片.一个输出就是任何清晰地输出到一个文件的单个值(或用户输出)、一个输出参数,或对一个全局变量赋值等.如图 1 所示是一个嵌入在一个程序中的数据切片的例子.

1.1 切片抽象和抽象数据切片

切片抽象把每个方法模型化成一组数据切片,数据切片又可模型化为一列数据标记.如上例有 $SliceAbstract(SumAndProduct)=\{DataSlice(SumN),DataSlice(ProdN)\}$.

定义 1(抽象数据切片).在数据切片中,不考虑数据的出现次数而计算的切片称为抽象切片,抽象数据切片中的数据称为抽象数据.

例如,$SumN$ 的抽象数据切片为 $AbstractDataSlice(SumN)=\{N,SumN,I,0,1\}$,$SumN$ 等都是抽象数据.抽象数据切片更能反映切片的本质.

1.2 抽象数据切片算法设计

在介绍计算数据切片的算法之前,还需引入几个概念.

定义2(切片变量). 把程序的所有有意义的输出变量都称为切片变量.有意义是指对这些输出变量计算切片是有意义的.例如,上例中的$SumN$和$ProdN$就是切片变量,而N,I等就不是切片变量.

定义3(数据依赖图(DataDG)). 数据依赖图是一个二元组(D,E),其中结点D是程序中所有数据的集合(包括变量、常量的定义和引用等),E表示结点之间的流依赖(包括定义-引用、赋值、值传递等)和控制依赖关系.

定义4(抽象数据依赖图(ADataDG)). 抽象数据依赖图是一个二元组(AD,E),其中结点AD是程序中所有抽象数据的集合(包括变量、常量的定义和引用等),E表示结点之间的流依赖(包括定义-引用、赋值、值传递等)和控制依赖关系.如图2所示.

→ Dependence relationship (data or control dependence)①

The shadow parts show abstract data slice of $SumN$②

①依赖关系(数据或控制依赖).②阴影部分表示关于$SumN$的抽象数据切片.

Fig. 2

图2

算法.

输入:单个方法;

输出:抽象数据切片.

步骤:

(1) 确定切片变量(主要在输出变量中查找)以及变量之间的流依赖和控制依赖关系,画出数据依赖图;

(2) 从切片变量出发,沿着流依赖和控制依赖边正向遍历,标记所有到达的结点.由所有可达结点构成的集合就是该方法关于某个切片变量的抽象数据切片.

2 胶水、强力胶水和粘性

在上例中,可以看到有些数据标记是多个数据切片所公有的,例如N,I,1等同时存在于$SumN$和$ProdN$的切片中.这种标记就是切片之间的联系,我们称为联系切片的Glue.

定义5(胶水(glue)). 在方法M的一个切片抽象$SA(M)$中,我们定义Glue为存在于该切片抽象的多个数据切片的数据标记的集合,记为$G(SA(M))$.一个glue token就是一个存在于多个数据切片中的标记.

定义6(强力胶水(super-glue)). $SA(M)$是方法M的一个切片抽象.如果某个数据标记存在于$SA(M)$的所有数据切片中,则称该数据标记是$SA(M)$的Super-Glue标记,记为$SG(SA(M))$.

显然,$SG(SA(M))\subseteq G(SA(M))$,即所有Super-Glue标记也是Glue标记.

定义7(粘性(stickiness或adhesiveness)). 由于Glue token和Super-Glue token表示单个切片之间联系的紧密程度,它们的作用就是绑定切片.个别的glue token对内聚的影响程度和它绑定的切片数目有关系.为了描述Glue token对内聚的这种相对影响程度,我们引入粘性(stickiness或adhesiveness)的概念.

抽象模型如表1所示.

Table 1 The abstract model shows the connections among slices

表1 表示切片之间联系的抽象模型

Program	Connection	Slice 1	Slice 2	Slice 3
Data 1	Super-glue	*	*	*
Data 2	Glue	*	*	
Data 3	Super-glue	*	*	*
Data 4		*		
...				
Data n	Glue		*	*

3 内聚度量

在面向对象范型中时,按照切片抽象(slice abstractions)、数据标记(data tokens)、胶水标记(glue tokens)和强力胶水标记(super-glue tokens)的术语,数据标记、胶水标记以及强力胶水标记的基本定义类似于那些在过程范型中使用的定义[7~9],即胶水标记是那些属于多个数据切片的数据标记,强力胶水标记是那些属于所有数据切片的胶水标记.在下面的定义中,符号#表示取集合中元素个数的运算,例如$\#\{a,b,c\}=3$.

定义8(强数据内聚(strong data cohesion,简称SDC)). 我们定义SDC为方法M中强力胶水标记(super-glue token)数和总标记(token)数的比,即$\mathrm{SDC}(M)=\frac{\#(SG(SA(M)))}{\#(tokens(M))}$.

定义9(弱数据内聚 weak data cohesion,简称WDC). 我们定义WDC为方法M中胶水标记数和总标记数的比,即$\mathrm{WDC}(M)=\frac{\#(G(SA(M)))}{\#(tokens(M))}$.

定义10(粘性(adhesiveness)). 胶水标记的粘性与每个胶水标记能够粘住的切片的相对数有关,即

$$\alpha(t,M)=\begin{cases}\dfrac{\#slices\ in\ p\ containing\ t}{\#(SA(M))} & \text{if } t\in G(SA(M))\\ 0 & \text{otherwise}\end{cases}$$

一个SA(切片抽象)的总的粘性A是方法M中数据标记粘性的平均值,即

$$A(M)=\frac{\sum_{t\in tokens(M)}\alpha(t,M)}{\#(tokens(M))}.$$

等价地,总的粘性也可以通过计算粘性的量和可能粘性的总量的比来计算,亦即

$$A(M)=\frac{\sum_{t\in G(SA(M))}\#slices\ containing\ t}{\#(tokens(M))\times\#(SA(M))}.$$

一个类的强力胶水标记表示为$SG(SA(C))$,是类的每个方法的强力胶水标记的并.类似地,类的胶水标记的集合,记为$G(SA(C))$,是类的每个成员方法的胶水标记的并.$tokens(C)$是类C的所有数据标记的集合.这样,我们就可以定义一个类的内聚度量.

强数据内聚.一个类C的强数据内聚是强力胶水标记数和类中所有数据标记数的比.一个没有强力胶水标记的类将是零强数据内聚.

$$\mathrm{SDC}(C)=\frac{\#(SG(SA(C)))}{\#(tokens(C))}.$$

弱数据内聚.一个类C的弱数据内聚是胶水标记数和类中所有数据标记数的比.一个没有胶

水标记的类将是零弱数据内聚.

$$\mathrm{WDC}(C)=\frac{\#(G(SA(C)))}{\#(tokens(C))}.$$

数据粘性(data adhesiveness).一个类C的数据粘性定义为包含每个粘性标记切片数的总和与类中数据标记的数目和数据切片数目的乘积的比.

$$\mathrm{DA}(C)=\frac{\sum_{t\in G(SA(C))}\#slices\ containing\ t}{\#(tokens(C))\times\#(SA(C))}.$$

所有这些内聚度量的值是在0和1之间变化.当它们取值为0时,表示此方法有多个输出,并且没有某个特定度量能表示某个内聚属性.没有super glue标记的方法具有零强内聚性——表示没有数据标记对所有输出都有影响.没有glue标记的方法具有零弱内聚性或零粘性——表示没有数据标记对多个输出有影响.如果一个方法的所有数据标记都是super-glue标记,SDC和粘性达到最大值,这时,任何一个数据标记都对所有输出有影响.如果M中所有数据标记都是glue token,则该方法是弱数据内聚(WDC),这时,所有的数据标记都不只影响一个输出.

4 类内聚

4.1 类内切片

4.1.1 类内切片概念

我们把成员方法和成员变量作为类切片中不可分割的单元.这样,如果一个类切片包含某个成员方法,就把该成员方法作为一个整体单元包含进来.成员方法和成员变量是通过该成员方法使用该成员变量相关联的.如果两个成员方法使用一些共同的成员变量,则它们通过这些成员变量而相关.按照这种思路,我们可以定义类内切片如下:

定义11(类内切片). 类C关于某个切片准则(n,V)的类内切片是由C中所有影响V的值(后向切片)或受V的值影响(前向切片)的成员变量和成员方法组成.其中n表示语句号,V表示在第n条语句定义或引用的变量集合.

4.1.2 类依赖图

为了建立类依赖图,必须分析一个类中成员方法和成员变量之间的各种依赖关系.在JAVA的类内部成员函数和成员变量可以通过以下两种情况发生依赖关系:(1) 多个成员方法共享某个(某些)成员变量而形成的直接定义或使用关系,简称MSV(methods sharing variables)关系;(2) 通过调用其他方法来间接使用成员变量而形成的调用关系,简称CRV(call relation vairiables)关系.

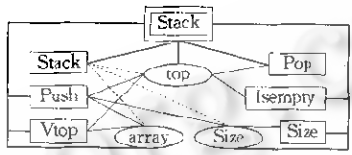
MSV关系是指两个方法通过共享成员变量来实现的通信关系.当两个或多个类方法读或写相同的类成员变量时就会建立MSV关系.CRV关系是指从一个方法到另一个方法的直接(或间接)消息发送.当一个方法通过消息传递调用另一个方法时,服务器方使用的成员变量也可以被客户方间接使用.可以通过MSV关系来反映CRV关系:具有调用关系的两个方法通过被它们使用的成员变量连接起来,一个方法直接使用这些成员变量,另一个方法通过调用关系间接使用这些成员变量.当服务器方法既不读也不写成员变量时,这两个方法之间就不存在MSV关系.

定义12(类依赖图ClassDG). 类依赖图是一个二元组(N,E),其中结点集合N是类C中类入口结点和所有成员变量和成员方法,边集合E表示结点之间所有的MSV和CRV关系以及类成员

关系等.如图 3(b)所示.

```
class Stack
{
    int * array, top, size;
public:
    Stack(int s ){
        size=s ;
        array=new int[size];
        top=0;}
    int Isempty(){
        return top==0;}
    int size(){
        return size;}
    int Vtop(){
        return array[top-1];}
    void push(int item){
        if (top==size)
            printf("Empty stack.\n");
        else
            array[top++]=item;}
        int Pop(){
            if (Isempty())
                printf("Full stack.\n");
            else
                --top;}
}
```

(a) A source program of java class
(a) 一个Java类源程序Stack



(b) Class dependence graph of stack class
(b) Stack类的类依赖图

Fig. 3
图3

图 3 是一个类 Stack 的源程序及各种类成员之间的 MSV 关系.图 3(b)中的矩形表示类成员方法,椭圆形表示类成员变量.矩形和椭圆形之间的一条连接表明了相应矩形的类成员方法使用相应椭圆形的类成员变量.

任何一个 JAVA 类都会由于构造函数和析构函数的存在而在方法之间产生联系.所以,我们在建立模型和度量过程中不包括构造函数和析构函数.图 3 中的虚线表示构造函数与成员变量之间的连接.

4.1.3 计算类内切片的算法

我们基于类依赖图提出一种计算类内切片的算法.

算法.

输入:JAVA 类;

输出:类内切片(成员函数和成员变量的集合).

步骤:

(1) 统计 JAVA 类的类源程序中所有成员变量和成员函数的个数.

(2) 分析成员变量和成员函数之间的所有 MSV 和 CRV 关系.

(3) 建立类依赖图(ClassDG).

(4) 分别找出与每个成员变量相关联的所有成员函数:在 ClassDG 中从某个表示成员变量的结点出发,沿着与之相连的边遍历,标记所有达到的方法结点.在遍历过程中,如果遇到另一个成员变量,则向上回溯一步,继续沿着其他路径遍历,或者返回开始结点,结束遍历过程.

(5) 把遍历过程中标记的所有成员方法和该成员变量放到集合中,此集合就是该类关于某个成员变量的一个类内切片.

性质. 设 $A(1),A(2),\ldots,A(n)$表示类 C 中所有的成员变量,$M(1),M(2),\ldots,M(m)$表示类 C 中所有的成员方法,$Slice(A(i)),i=1,\ldots,n$ 表示类 C 关于成员变量 $A(i)$的类内切片,则有

$C=\bigcup_{i=1}^{n} Slice(A(i)) \bigcup classheadentry(C)$. 其中 $classheadentry(C)$ 表示类 C 的类头入口.

4.2 类内聚

一个类的内聚表示类中可见方法[10]之间的连接程度. 成员变量对类的客户来说通常是不可见的,对象的状态通过类方法来提供. 在可见方法之间,成员变量包含在 MSV 关系之中. 当不可见方法被可见方法间接调用时也被包含进来. 这样,就可以把类内聚模型化为类中可见方法(不包括构造函数和析构函数)之间的 MSV 关系.

类内聚的实际度量是基于成员方法之间的直接和间接连接. 令 $NP(C)$ 表示类 C 中可见方法的总数. NP 表示一个类中直接或间接连接的最大可能数. 如果类 C 中有 N 个方法,则有 $NP(C)$ 为 $N*(N-1)/2$,有如下两种内聚度量:紧密类类聚(tight class cohesion,简称 TCC)和松散类内聚(loose class cohesion,简称 LCC),令 $NDC(C)$ 和 $NIC(C)$ 分别表示类 C 中直接连接和间接连接数,则

紧密类类聚表示直接连接方法的相对数,

$$TCC(C)=NDC*(C)/NP(C);$$

松散类类聚表示直接连接方法或间接连接方法的相对数,

$$LCC(C)=(NDC(C)+NIC(C))/NP(C).$$

LCC 的值总是大于或等于相应的 TCC 的值. TCC 和 LCC 表示了类中可见方法之间的连接程度. 这些可见方法是指在该类中定义或被该类继承的那些方法. 然而,类内聚度量只考虑定义在该类中可见方法也是有用的,因为此度量不受超类的内聚影响. 还可以通过使用一个类中的局部(非继承)方法来定义局部类内聚(local class cohesion)度量.

5 结 论

Bieman 和 Ott 开发了一组基于程序切片的函数内聚度量方法[7~9]. 这些度量方法仅应用到个别的函数,在整个类中的应用却不是很明显. Chidamber 和 Kemerer 为面向对象的软件开发了一种方法中内聚缺乏度(LCOM)的度量[11]. LCOM 在识别最没有内聚的类时是有效的,但它在区分两个部分内聚的类时却是无效的. M. Shumway 讨论了 Java 内聚度量的几个框架,其中提到基于切片的度量,但不够深入[12].

本文首先计算 Java 程序的数据切片和类切片,通过这些切片和数据、方法、类之间的关系来度量数据、方法、类的内聚性问题. 我们认为这种方法起到了很好的效果.

References:

[1] Fenton, N. Software Metrics— a Rigorous Approach. London: Chapman and Hall, 1991.

[2] Li, Bi-xin, Zheng, Guo-liang, Wang, Yun-feng, *et al*. An approach to analyzing and understanding program-program slicing. Journal of Computer Research and Development, 2000,37(3):284~291 (in Chinese).

[3] Weiser, M. Program slicing. IEEE Transactions on Software Engineering, 1984,10(4):352~357.

[4] Longworth, H. Slice based program metrics [MS. Thesis]. Michigan Technological University, 1985.

[5] Thuss, J. An investigation into slice based cohesion metrics [MS. Thesis]. Michigan Technological University, 1988.

[6] Horwitz, S., Reps, T., Binkley, D. Interprocedural slicing using dependence graphs. ACM Transactions on Programming Languages and Systems, 1990,12(1):35~46.

[7] Churcher, N., Shepperd, M. Towards a conceptual framework for object oriented software metrics. ACM Software Engineering Notes, 1995,20(2):69~76.

[8] Bieman, J., Kang, B.-K. Measuring design-level cohesion. IEEE Transactions on Software Engineering, 1998,24(2):111~124.

[9] Bieman, J., Ott, L. Measuring functional cohesion. IEEE Transactions on Software Engineering, 1994,20(8):644~657.

[10] Li, Bi-xin, Liang, Jia, Zheng, Guo-liang, *et al*. A framework for analyzing object-oriented program based on class hierarchy graph. Journal of Software, 2000,11(5):694~700 (in Chinese).

[11] Chidamber, S., Kemerer, C. A metrics suite for object oriented design. IEEE Transactions on Software Engineering, 1994,20(6):476~493.

[12] Shumway, M. Measuring class cohesion in Java [MS. Thesis]. Technical Report, CS-87-113, Department of Computer Science, Colorado State University, 1997. http://www.cs.colostate.edu/~ftppub/TechReports/1997/tr97-113.ps.Z.

附中文参考文献:

[2] 李必信,郑国梁,王云峰,等.一种分析和理解程序的方法—程序切片.计算机研究与发展,2000,37(3):284~291.

[10] 李必信,梁佳,郑国梁,等.一种基于类层次图的分析面向对象程序的框架.软件学报,2000,11(5):694~700.

Measuring JAVA Cohesion Based on Data Slice*

LI Bi-xin, ZHU Ping, TAN Yi, LI Xuan-dong, ZHENG Guo-liang

(*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing* 210093, *China*);
(*Department of Computer Science and Technology, University of Science and Technology of China, Hefei* 230056, *China*)
E-mail: zhenggl@nju.edu.cn; libx@seg.nju.edu.cn
http://www.nju.edu.cn

Abstract: Object-Oriented program slicing technique is widely used in program analysis, program comprehension, software testing, and software maintenance etc. In this paper, the basic concepts of abstract data slice and intra-class slice are created at first. Then, the cohesion existing in JAVA language is discussed based on these slices. Finally, the data cohesion, method cohesion and class cohesion are presented and solved by analyzing the relationship between these slice and data, method or class.

Key words: abstract data slice; inter-class slice; data cohesion; class cohesion; JAVA

* Received April 12, 2000; accepted July 12, 2000
Supported by the Youth Foundation of the National High Technology Development 863 Program of China under Grant No. 863-306-QN2000-2