文章编号:1000-582X(2009)10-1221-05

最大树法的 Aspect 挖掘方法

曾　一[1],洪　媛[1],刘　引[1,2],王　健[1]
(1.重庆大学 计算机学院,重庆 400030; 2.重庆农村商业银行,重庆 400020)

摘　要:提出一种基于最大树的 Aspect 挖掘方法,该方法使用 Aspect 的思想,从动态行为挖掘横切关注点获取运行时方法调用的信息,从而构造方法调用关系数据矩阵,在模糊相似关系的理论基础上,引入相似度构造出对象相似矩阵,并利用最大树方法识别系统中的横切关注点,从而为系统的软件重构和复用提供依据。最后通过实验验证了其有效性,进一步通过与目前具有代表性的挖掘方法进行比较,认为本方法具有实现清晰、效率较高的优点。

关键词:面向对象编程;遗留系统;模式识别;方面挖掘;最大树
中图分类号:TP311　　文献标志码:A

Maximum tree method based aspect mining method

ZENG Yi[1], *HONG Yuan*[1], *LIU Yin*[1,2], *WANG Jian*[1]
(1. College of Computer Science, Chongqing University, Chongqing 4000300, P. R. China;
2. Chongqing Rural Commercial Bank, Chongqing 400020, P. R. China)

Abstract: By means of discovering crosscutting concerns from legacy systems, aspect mining intends to help migrate the systems to an aspect-oriented design. An improved method based on maximum tree method for aspect mining is presented. The method uses aspect ideas to capture the runtime method-call information by mining crosscutting concerns from dynamic behaviors, and then constructs a method-call relationship data matrix. Based on fuzzy similarity relation theory, by introducing the similarity, an object similarity matrix is constructed, and the maximum tree method is used to identify the crosscutting concerns in the system. The method can provide a basis for system's software reconstruction and reusability. An experiment is conducted to verify the validity of the method. Compared with the existing typical mining methods, the method shows the virtue of clear implementation and high efficiency.

Key words: object oriented programming; legacy systems; pattern recognition; aspect mining; maximum tree

传统的面向对象方法将软件的业务逻辑作为软件设计的主线,业务逻辑是软件系统的核心关注点。但除此之外,软件还包括系统级的关注点,如日志、事务完整性、授权、安全性及性能等。这些系统级的关注点,往往混合且散乱于根据核心关注点划分组成软件的结构单元中。一般的程序语言不可能在不同的模块中实现不同的关注,必然导致同一模块实现多种关注。对关注的实现相互交叉的现象称为横切现象(cross cutting)。它将代码混乱(code tangling)和代码分散(code scattering)引入到软件结构中,使得代码的可维护性和可复用性降低。为了解决以上问题,在 20 世纪 90 年代提出了面向

收稿日期:2009-05-10
基金项目:国家自然科学基金面上项目资助(50875268)
作者简介:曾一(1961-),男,重庆大学教授,硕士生导师,主要从事软件工程理论与应用方向研究,(Tel)65102621;
(E-mail)zyjckxx@cqu.edu.cn。

Aspect(aspect oriented programming 简称 AOP)程序设计技术[1]。AOP 是一种能有效地管理、组织、分离横切关注点的程序设计方法学，是对 OOP 的一种补充。通过对横切关注点的模块化处理，AOP 使得避免代码分散和代码混乱成为可能，而且为面向对象程序的静态特性引入了动态元素，提供在已有系统中不改变静态对象模型的前提下加入新的应用程序行为的能力[2]。

目前，国际上已经发布了多种 AOP 程序设计语言，如 AspectJ[3]，AspectC＋＋[4]，AspectCJHJ 等。针对如何在已有软件代码中识别 Aspect 的问题，已经形成了一个 AOSD 的分支研究领域：Aspect 挖掘与重构工程（aspect mining and refactor engineering)[5]。Aspect 挖掘是指将面向对象系统中的横切关注点识别并分离出来，Aspect 重构是指将与横切关注相关的代码拆分出来并组装成 Aspect。

目前，国际上已提出了一些 Aspect 挖掘方法，并对这些方法在性能和实现上作了一定的评价[6-7]，主要趋势是从程序的静态代码和动态行为两方面来进行挖掘[8]。静态挖掘的代表有 D Shepherd 提出的基于的 Aspect 挖掘方法[9]和 Silvia Breu 提出的基于版本修改历史的挖掘方法[10-11]。动态挖掘的代表有使用扇入概念进行 aspect 挖掘方法[12]、Tom Tourwe 等人提出的基于形式概念分析的挖掘方法[13]等。

笔者提出一种基于最大树法 Aspect 挖掘方法。该方法在不修改原有程序代码的情况下，应用 AOP 的思想获取运行时方法调用的信息，使用最大树方法对系统中的横切关注点进行模糊聚类分析，从而为系统的软件重构和复用提供依据。与现有的一些方法相比，该方法具有实现清晰、效率较高的优点。

1 基于最大树法的模糊聚类挖掘方法

1.1 基本思想

AOP 的基本思想是方面高于对象的存在，方面可以在对象自身不知情的情况下对对象的状态和行为进行动态修改。因此，将利用现有的 AOP 机制在不修改遗留系统原有代码的情况下记录方法调用的日志。

模式识别是指运用一种自动技术把待识别的模式归入到相应的模式类中去。但是，现实的识别问题往往具有不同程度的模糊性，要划分的横切关注点常常是未知的。因此，把模糊集的相关理论方法应用于对横切关注点进行识别，既可满足客观情况的要求，也能得到较为理想的结果。通过分析比较，本文使用最大树法进行模糊聚类。

1.2 实现步骤

1.2.1 获取方法调用集合

设方法调用集合 MC 中的每一个元素为三元组构成

MC＝＜caller,target,level＞,

其中，caller 代表方法调用者，target 代表被调用的方法，level 代表方法调用的层数，若记程序中所有方法集合为 M，有 caller∈M，target∈M；若方法为直接调用，记 level＝0。

切入点（pointcut）是用于声明连接点(joinpoint)中关注的 AspectJ 机制，它封装了决策逻辑，通过计算这个逻辑来决定在遇到某个连接点时是否调用一份特定的通知(advice)。

根据 AspectJ 的语法，命名的切入点定义的一般形式如下

＜visibility－modifier＞ pointcut name (parameterList)：PointcutExpression；

特别地，对方法调用的捕获，其语法是

＜visibility－modifier＞ pointcut ＜pointcut name＞(＜any values to be picked up＞)：call(＜optional modifier＞ ＜return type＞ ＜class＞.＜method＞(＜parameter types＞))；

因此可以声明这样一个切入点用于对方法调用的捕获 public pointcut callAllMethods()：call(＊＊.＊(..))；

为获取程序中方法调用的关系，使用 AOP 的思想实现自动记录调用信息的功能。通过使用通配符，无论被调用的方法具有怎样的修饰符、返回类型或任意数量的参数，只要发生了方法调用，都可以捕获这个被调用的方法。

通知(advice)就是方面被调用时所执行的代码。

```
before():callAllMethods() //方法体代码运行前执行
{
...
thisEnclosingJoinPointStaticPart. getSignature();
//使用运行时 API 获取调用者
thisJoinPoint. getSignature();
//使用运行时 API 获取被调用方法
…//具体的记录方法调用日志代码
}
```

由此，在不对原有代码进行修改的情况下，为程序增加了记录运行时调用信息的功能。

1.2.2 构造方法调用关系数据矩阵

在面向对象的遗留系统中，很多横切关注点都以方法的散布形式表现，若存在某些方法在其他系统业务功能的方法调用中频繁出现，则这些方法就

有可能成为一个横切关注点的组成部分。所以,可以将系统中每一个方法作为一个分析对象。该方法是否是一个横切关注点的组成部分,由系统运行时的动态行为所决定,即此方法被哪些其它的方法共同调用了[14]。假设系统中方法的数量为 m,则系统中有 m 个待分析对象,每一个对象都是一个 m 维向量。规定此对象的数据矩阵这样构成:若方法 i 被方法 j 调用,则 i 的第 j 维标为 1,否则为 0。

1.2.3　使用模糊相似关系进行分类

定义 1　模糊集与隶属函数

设 U 是论域,u_A 是把任意 $u \in U$ 映射为[0,1]上某个值的函数,即

$u_A: U \rightarrow [0,1]$,

$u \rightarrow u_A(u)$,

则称 u_A 为定义在 U 上的一个隶属函数,由 $u_A(u)(u \in U)$ 所构成的集合 A 称为 U 上的一个模糊集,$u_A(u)$ 称为 u 对 A 的隶属度。

定义 2　模糊关系

在 $U_1 \times U_2 \times \cdots \times U_n$ 上的一个 n 元模糊关系 R 是指以 $U_1 \times U_2 \times \cdots \times U_n$ 为论域的一个模糊集,记为 $R = \int_{U_1 \times U_2 \times \cdots \times U_n} \mu_R(u_1, u_2, \cdots u_n)/(u_1, u_2, \cdots u_n)$。

定义 3　模糊关系合成

设 R_1 与 R_2 分别是 $U \times V$ 与 $V \times W$ 上的 2 个模糊关系,则 R_1 与 R_2 的合成是指从 U 到 W 的一个模糊关系,记为 $R_1 \circ R_2$,其隶属函数为 $\mu_{R_1 \cdot R_{12}}(u,w) = \vee \{\mu_{R_1}(u,v) \wedge \mu_{R_{12}}(v,w)\}$

定义 4　模糊相似关系

设 $U = \{u_1, u_2, \cdots, u_n\}$ 是论域,R 是定义在 $U \times U$ 上的一个模糊关系,它对应的模糊矩阵为 $\boldsymbol{A} = (a_{ij})$,如果该矩阵满足如下条件:

1)具有自反性,即 $a_{ii} = 1, i = 1,2,\cdots,n$;

2)具有对称性,即 $a_{ij} = a_{ji}, i,j = 1,2,\cdots,n$。

则称矩阵 $\boldsymbol{A}$ 是一个模糊相似矩阵,它所对应的模糊关系 R 是一个模糊相似关系[15]。

实际上,构造论域 $U = \{u_1, u_2, \cdots, u_n\}$ 上的模糊关系 R,$u_R(u_i, u_j)$ 表示 u_i 与 u_j 的相似程度,其值可由专家评分得到,或者通过计算相似度得到。在这里,可以利用方法调用的关系作为相似度的度量。

对上个步骤所得方法调用矩阵(参见表 1)引入相似度进行重新计算得到对象相似矩阵(参见表 2)。若方法 i 和方法 j 同时被方法 k 调用,则在列 k 上同时为 1,表示两方法相似;若均未被调用,则在列 k 上同时为 0,不能确定两方法是否相似或相异;若一个被调用而另一个未被调用,则一个为 0,一个为 1,表示在列 k 上相异,这种相异的列占所有相同列和相异列总和的比例越小,表示 i 和 j 越相似。

由此可得计算相似度的公式

$$u(i,j) = \frac{s}{s+m+n},$$

$u(i,j)$ 表示方法 i 和 j 的相似程度,s 是 i,j 同时为 1 的列的数目,m 是 i 为 1、j 为 0 的列的数目,n 是 i 为 0,j 为 1 的列的数目。

1.2.4　利用最大树法对对象相似矩阵进行聚类分析

在模糊分类时,若使用基于模糊等价关系的分类,为了得到满足传递性的等价矩阵,则需要对矩阵进行多次合成运算,特别是当矩阵的维数较大时,将占用的十分可观的机器时间。所以在这里利用最大树法直接对模糊相似矩阵进行分类。其具体步骤如下

1)以调用矩阵的第 0 列中的方法序列为顶点,记为 $u_1, u_2, \cdots, u_n$;

2)将 $u(i,j)$ $(1 \leqslant i,j \leqslant n)$ 按从大到小的顺序排序:$a_1 > a_2 > \cdots > a_k$,其中 a_k $(k = 1,2,\cdots,l)$ 为某 $u(i,j)$;

3)将相似度为 a_1 的顶点连接,并在相应的线段上注明 a_1,若连边时出现回路,则去掉此边;

4)依次对 $a_2, a_3, \cdots, a_k$ $(k \leqslant l)$ 重复步骤 3 直至全部顶点连通为止,这样就得到了一棵最大树(可以不唯一);

5)取 $\lambda \in [0,1]$,并在最大树中砍去权重小于 λ 的边,这就得到互不连通的几棵子树,每一棵子树中的节点则在水平 λ 下归为一类。

2　挖掘示例

为了验证方法的可行性和正确性,下面用一个实例来详述挖掘示例的整个过程。以下为一段代码示例。类 Account 主要用于提取存款,辅助功能包括访问数据库、记录日志等。

```
public class Account {
  private float amount = 0;
  public static void main(String argv[]) {
  new Account().debit(100);
log("operation success,exit");
  }
  public void debit(float debitAmount) {
  float amount = getAmount();
  if (amount > =debitAmount) {
  amount -= debitAmount;
  update(amount);
  log("debit" + debitAmount);
  }
  }
  private float getAmount() {
  conn();
//get amount here with sql
  log("read amount");
close();
```

```
    return amount;
    }
    private void update(float amount) {
    conn();
  //store amount here
     log("update" + amount);
    close();
    }
    private void conn() {
  //do db open here
    }
    private void close() {
  // do db close here
    }
    private void log(String str) {
  //do log here
    }
  }
```

运行代码,获取的直接方法调用集合 MC 如下:

<main(), debit(float)>, <main(), log(String)>, <debit(), getAmount()>, <getAmount(), conn()>, <getAmount(), log(String)>, <getAmount(), close()>, <debit(), update(float)>, <update(), conn()>, <update(), log()>, <update(), close()>, <debit(), log()>。

得到的方法调用关系数据矩阵如表 1 所示。

表 1 方法调用关系数据矩阵

参数	O_1	O_2	O_3	O_4	O_5	O_6	O_7
O_1(main)	0	0	0	0	0	0	0
O_2(debit)	1	0	0	0	0	0	0
O_3(getAmount)	0	1	0	0	0	0	0
O_4(update)	0	1	0	0	0	0	0
O_5(conn)	0	0	1	1	0	0	0
O_6(close)	0	0	1	1	0	0	0
O_7(log)	1	1	1	1	0	0	0

根据前面提出的相似度计算方法 $u(i,j)=\frac{s}{s+m+n}$,计算得到的相似矩阵如表 2 所示。

表 2 对象相似矩阵

参数	O_1	O_2	O_3	O_4	O_5	O_6	O_7
O_1	1	0	0	0	0	0	0
O_2	0	1	0	0	0	0	0.25
O_3	0	0	1	1	0	0	0.25
O_4	0	0	1	1	0	0	0.25
O_5	0	0	0	0	1	1	0.50
O_6	0	0	0	0	1	1	0.50
O_7	0	0.25	0.25	0.25	0.5	0.5	1.00

$u(i,j)$ 是一个非负的值,有 $0 \leqslant u(i,j) \leqslant 1$。根据相似度的计算方法可知,当方法 i 和 j 相似的程度越高,$u(i,j)$ 越接近 1;

根据对象相似矩阵构造的最大树如图 1 所示:

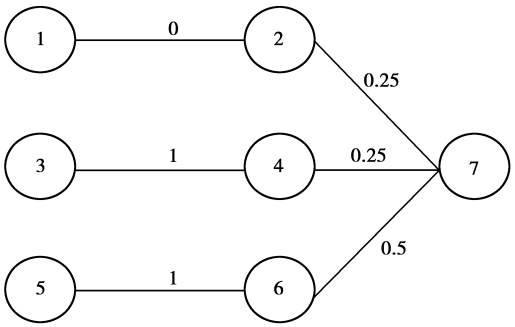

图 1 7 个节点的最大树

用 λ 对最大树去边,若 λ 取值趋近于 1,则关注点内的方法相似度较高,划分出的关注点越多;若 λ 取值趋近于 0,则关注点内的方法相似度较低,划分出的关注点越少。综合以上考虑,不妨设 λ=0.5,即在最大树中去掉权重小于 0.5 的边,可得切割后的子树集如图 2 所示

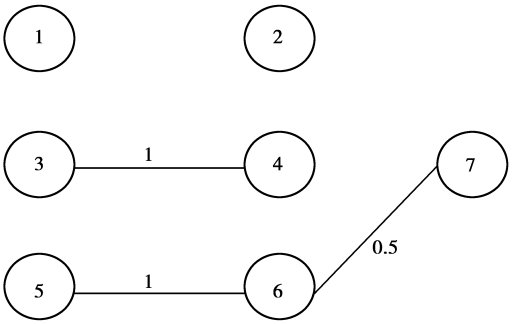

图 2 用 λ=0.5 切割后的子树集

由此可得到相应的分类为

$$\begin{cases} w_1 = \{\text{main}\}; \\ w_2 = \{\text{debit}\}; \\ w_3 = \{\text{getAmount}, \text{update}\}; \\ w_4 = \{\text{conn}, \text{close}, \text{log}\}。 \end{cases}$$

用得到的分类对 Account 类进行分析,可以得到以下结论:main 和 debit 是相对独立的功能;conn 和 close 是一对同时出现的功能,它们共同实现了对数据库访问的封装,并且与 getAmount、update 中的具体业务逻辑是相互独立的,所以这 2 个方法可以提取出来封装成一个横切关注点;同理可知,log 被调用的方式也是与 conn、close 相似的,同样它与具体业务逻辑实现也是相互独立的,也可以封装为一个横切关注点。

3 结　语

提出一种基于最大树法的 Aspect 方法，通过分析代码运行时调用的关系,使用最大数法通过模糊相似关系将代码中具有一定相似调用模式的方法聚集在一个分类中,从而为实现面向对象程序向面向方面程序的转换提供了一定的帮助。最后通过一个实际系统中的例子进行实验比较,说明了该方法的可行性和有效性。

采用的方法基于代码运行时调用的关系,与基于文本的和基于类型的 Aspect 搜索方法相比,不要求相似的行为必须采用相似的命名规则,而是从动态行为挖掘横切关注点;与基于版本修改历史的挖掘方法相比,不要求原系统代码具有良好的版本配置管理工具和详细的版本历史记录。与使用扇入概念进行 Aspect 挖掘方法相比,本方法不需要通过反射原理对目标系统进行植入,而是直接使用 AOP 的思想为系统中每一个方法调用动态增加日志记录功能。与使用模糊等价关系进行 Aspect 挖掘的方法相比,本方法占用的机器时间较少,效率较高。

参考文献:

[1] HURSCH W, VIDEIRA L C. Separation of concerns [M]. Boston :Northeastern University , 1995.

[2] MONNOX A. Rapid J2EE development[M]. Pearson Education, 2005.

[3] KICZALES G, HILSDALE E, HUGUNIN J, et al. An overview of AspectJ[C]// In: Knudsen JL, ed. Proc. of the European Conf. on Object Oriented Programming. Berlin: Springer-Verlag, 2001: 327-254.

[4] SPINCZYK O, LOHMANN D, URBAN M. Advances in AOP with AspectC++[C]// In: Hamido F, eds. Proc. of the Software Methodologies, Tools and Techniques (SoMeT 2005). Tokyo: IOS Press, 2005: 33-53.

[5] 陈向群, 杨芙清. 面向 Aspect 的操作系统研究[J]. 软件学报,2006,17(3):620-627.
CHEN XIANG-QUN, YANG FU-QING. Research on aspect oriented operating systems [J]. Journal of Software, 2006,17(3):620-627.

[6] CHANCHAL K R , MOHAMMAD G U , BANANI R ,et al. Evaluating aspect mining techniques: A case study[C]// 15th IEEE International Conference on Program Comprehension. [S. L.]: IEEE,2007:167-176.

[7] COJOCAR S, SERBAN G. On evaluating aspect mining techniques [C]// Intelligent Computer Communication and Processing, 2007 IEEE International Conference on 6-8 Sept. [S. L.]:IEEE, 2007 :217-224.

[8] MARIUS M , LEON M, ARIE V D . A common framework for aspect mining based on crosscutting concern sorts[C]// Reverse Engineering, 2006. WCRE 06. 13th Working Conference [S. L.]: IEEE, 2006 (s): 29-38.

[9] SHEPHERD D , TOURWE T, POLLOCK L. Using language clues to discover crosscutting concerns [C]. Workshop on the Modeling and Analysis of Concerns, 2005.

[10] BREU S, ZIMMERMANN T. Aspect mining for large systems[C]// Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA, 2006. [S. L.]:IEEE, 2006: 641-642.

[11] BREU S, ZIMMERMANN T. Mining aspects from version history[C]// Automated Software Engineering. ASE '06. 21st IEEE/ACM International Conference. [S. L]:IEEE, 2006, 9:221-230.

[12] MARIN M , VAN DEURSEN A , MOONEN L. Identifying aspects using fan in analysis[R]. Technical Report SENR0413, 2004: 132-141.

[13] TOURWE T, MENS K. Mining aspectual views using formal concept analysis[C]// In Proceedings of the 4th International Workshop on Source Code Analysis and Manipulation (SCAM 2004). [S. L.]: IEEE Computer Society, 2004:97-106.

[14] 何丽莉,白洪涛.用聚类分析方法挖掘 Aspect[J].计算机集成制造系统,2006,12(1): 149-153.
HE LI-LI , BAI HONG-TAO. Research on aspect mining using clustering analysis [J]. Computer Integrated Manufacturing Systems, 2006, 12 (1): 149-153.

[15] 王永庆.人工智能原理与方法[M].西安交通大学出版社,2003.
WANG YONG-QING. The principle and method of aI [M]. Xi'an:Xi'an Jiaotong University Press, 2003.

(编辑　侯　湘)