14—18 持久性 对象标识，人工智能 维普资讯 http://www.cqvip.com

计算机科学1992Vol.19No.5

基于对象标识的对象持久性语义

TP18

陈 睿 蔡希尧 陈 平 （710071 西安电子科技大学）

摘 要

本文在Khoshafian的对象标识模型基础上引伸出一种分析和处理对象持久性语义的工具——引用可达图，讨论了在持久对象的处理过程中引用可达图的变化情况，并讨论了引用可达图应用于持久存贮空间管理的方法。

所谓持久性，是指对象的时态特性，一个对象的存活期如果超过了一次程序运行时间，则该对象就是一个持久的对象。传统的数据库系统是支持持久性的，数据的存活期长于一个数据库事务，但大多数程序设计语言不支持持久性。现在，越来越多的研究工作集中于数据库程序设计语言这个课题上，试图把数据库系统和程序设计语言有机地结合起来，以提供更先进的集成化软件开发工具[1]。而程序设计语言中引入持久性则是程序设计语言中加入数据库功能的首要条件[2]。PS-Algol语言是一种最经典的支持持久性的程序设计语言[3]，这种语言方案中第一次提出了保持类型完全性（type completeness）的持久性原则，这一原则被普遍接受，后来提出的支持持久性的OOPL方案，例如CO_2[4]、O++[5]等也大都坚持这一原则。在现有的支持持久性的面向对象系统中，对对象持久性语义的解释不尽相同，无论是从用户观点出发，还是从系统实现的角度出发，都需要找到一种处理持久性语义的统一方法，本文的目的，就是阐述一种对象标识的持久性语义分析和表示工具——引用可达图。

一、对象标识（Object identity）

引入对象标识是为了区分对象和对象的值。在OOPL中，对象是独立于值而存在的，一个对象的值可以在程序运行的不同时刻变化，因此不能根据值来唯一地识别对象，而需要有一种独立于值的对象标识机制。Khoshafian等人提出了一种基于对象标识的对象模型，用以表示独立于值的对象[6]。每个对象都是一个三元组：

O=（标识符，类型，值）

其中，

1．标识符能够唯一地识别对象，所有对象的标识符构成了一个标识符集合I；

2．类型是{atom, tuple, set}中的一个元素，atom指系统定义的内部数据类型，如int、boolean等，tuple指record、struct等元组构造类型，而set指集合构造类型，例如数组等；

3．对象值是下列情况之一种：

1）若对象类型为atom，则对象值是某个系统内部数据类型值域中的一个元素；

2）若对象类型为tuple，则其值是I的一个子集；

3）若对象类型为tuple，则对象值具有以下形式：$[A_1:I_1, \cdots, A_n:I_n]$，其中$A_i$是元组对象的属性名，$I_i$是I的元素，作为$A_i$属性值的对象$I_i$可以用$O.A_i$表示，即$I_i=O.A_i$。

上述对象模型已经被用于FAD中。这个模型非常简单，表达力也很强。每个对象可

维普资讯 http://www.cqvip.com

以用这种模型唯一地、完整地表示出来，例如

(obj1, tuple, [name:obj2,age:obj3, children:obj4])

(obj2, atom, "Marry")

(obj3, atom, 27)

(obj4, set, { })

从上面表示中，我们已经看到，这一模型还有可以改进的余地。为了简化表示方法，也为了使下一节讨论的引用可达图更加清楚，我们对这一模型作两处改动：

1. 对一个atom类型对象，可以根据值来唯一地识别它，因此可用值作为atom类型对象的标识符。这将使得：1) 若set类型对象S的元素类型为atom，则S的值是一个atom类型值集，而非I的子集；2) 若tuple类型对象T的A_i属性$T.A_i$是一个atom类型对象，则$T.A_i$的值是一个atom类型值而非I的某个元素。

2. 引入一个特殊对象标识符NIL，NIL指无定义的对象，即一个未定义的atom值、或一个空集、或一个空元组。任意一个对象在创建后和未初始化或赋值前，其值都是NIL。

于是，obj1可以被表示为

(obj1, tuple, [name:"Marry", age:27, children:NIL])

每个对象还可以用唯一的图完整地表示出来，作图的方法是：

1. 一个对象用一个结点表示，每个结点都标记以对象标识符；

2. 若一个结点对象是set类型，则从该结点引出一条无标记的边到每一个元素对象结点；

3. 若一个结点对象是tuple类型，则从该结点引出一条标记A_i的边到作为其A_i属性值的那个结点。

用这种方法，可以将obj1表示为图1。

显然，从一个tuple类型对象结点引出的所有边中，不能有两条边的标记是相同的；

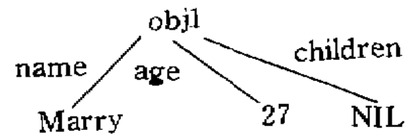

图1

从NIL和atom类型对象结点不能引出任何边。

二、引用可达图

我们已经看到，上面的图可以表示对象的逻辑构成。如果将其改为有向图，则可以充分地表达对象间的引用关系。我们称这种图为引用可达图。任意给一个对象，设其标识符为oid，我们可作一个以oid为根的引用可达图，作图步骤如下：

1. 画结点oid；

2. 对任一个结点obj，若obj是一个set类型对象，则将其值集合中的每个元素作为一个结点，并从obj结点引一条指向元素对象结点的有向边；

3. 若obj是一个tuple类型对象，则将$obj.A_i$作为一个结点，并从obj引一条指向$obj.A_i$的有向边，以A_i标记此边，其中A_i是obj的任一属性名；

4. 重复第2、3步，直到所有出度为零的结点都是atom类型对象或NIL。

值得一提的是，引用可达图具有以下特点：

1. 只有一个结点入度为零，称为根结点；

2. 所有atom类型对象和NIL结点入度为1，出度为零，称为终止结点。

引用可达图是分析对象持久性语义的直观工具。在支持持久性的面向对象系统中，有些系统认为，所有对象都是持久的，例如Orion[8]，而其它系统则允许暂态对象和持久对象同时存在，这似乎更合理一些。在后一类系统中，允许用户显示地创建持久对象。持久对象被创建后，必须被保存在持久存贮空间中（二级存贮器），那么，为了保存一

维普资讯 http://www.cqvip.com

个持久对象，必须存贮哪些内容呢？为了解决这个问题，大多数系统都采用以下的持久性语义：

1. 若持久对象obj是集合，则obj的所有元素对象都是持久的；

2. 若持久对象obj是元组，则obj的所有属性对象都是持久的。

显然，这种语义刚好可以用引用可达图来表示。若对象obj是持久的，则以obj为根的引用可达图中的所有结点对象都是持久的。因此，可以用引用可达图来分析对象的持久性语义。

支持持久性的面向对象系统，如果要区分持久对象和暂态对象，就总要在用户界面中提供某种手段，以便用户能够定义一个持久对象，系统则以此对象为根，找出所有引用可达的对象，把它们存贮到持久存贮空间中。FAD系统语言中的关键字database[7]、O_2的命名对象[4]、Avalon/C++中的recoverable类[8]都是为这个目的服务的。用户定义的持久对象，其存活期超过一次程序运行时间，我们称这种对象是“系统可引用的持久对象”。假设有一个系统可引用的持久对象obj1，引用了另一个暂态对象obj2，则obj2的存活期将被延长为obj1的存活期。但是，当一次程序运行结束时，只在obj1中保存了obj2的值，而obj2本身已经消亡。也就是说，被持久对象引用的暂态对象，虽然具有持久的存活期，却不能够被系统当作持久对象来使用。为了区分这种情况，我们在引用可达图中加入一个特殊的结点System，从System结点引一条无标记的有向边到每一个系统可引用的持久对象结点。从逻辑上看，System是一个集合的名，由系统管理的所有持久对象都是System的元素。这样，System就成了引用可达图的唯一的根。在系统运行的任意时刻，所有持久对象的状态都能通过以System为根的整体引用可达图表现出来。

我们来看一个稍微复杂一些的例子。用户定义了两个持久的Set类型对象S1，S2。在S1中插入了三个暂态对象person1、person2和person3，在S2中插入了两个暂态对象person4和person5，其中person1和person2是夫妇，它们的children属性值都是S2，而person3没有孩子。这种状态可以用图2来表示。

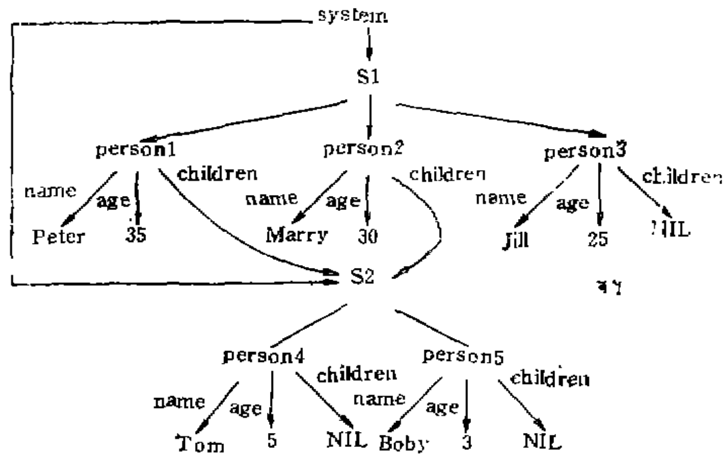

图2

三、持久对象上的操作：创建、更新和删除

引用可达图可以表示运行时刻持久对象状态的变化。一旦持久对象被创建，就被作为结点加入到引用可达图中。如果程序中出现了对持久对象赋值、更新和删除的操作，都将引起引用可达图的变化。我们以O_2为例来说明。

先定义一个类：

```
add class person
  type tuple (name:string,
        age:integer,
        children:set (person)).
```

定义持久对象S1和S2：

```
  add name S1:set(person).
  add name S2:set(person).
```

这时S1、S2作为结点插入到引用可达图中。

```
  S1=S1+set (tuple (name:"Peter",
age:35, children:S2)).
```

维普资讯 http://www.cqvip.com

$S_1=S_1+$set (tuple (name:"Marry", age:30, children:S_2)).

$S_1=S_1+$set (tuple (name:"Jill", age: 25)).

以上三个语句分别插入了person1, person2和person3三个结点，以及终止结点Peter, Marry、Jill、35、30和25。

$S_2=S_2+$set (tuple (name:"Tom", age:5), tuple(name:"Boby", age:3)).

这时又插入了person4和person5等结点，使引用可达图变成图2。

由于引用可达图以对象标识为基础，故能够充分地表达共享的情形。例如，person1·children=person2·children=S_2，从person1和person2引出的children边都指向S_2。这对于对象更新也是很有利的，例如，Tom长了一岁，只需把person4.age改为6就行了，而person1和person2的children属性值都取得了更新后的S_2值。这样，既减少了更新操作的次数，又减少了出错的机会。

在删除一个持久对象时，引用可达图是很有用的。一个持久对象被删除，将使得一系列保存在持久存贮空间中的对象失去继续存在的意义，因为它们已经不被其它对象引用，系统无法存取到它们。进一步讲，如果结点obj被删除，则从obj可达，而且仅从obj可达的结点不再有用，可以被一同删除。利用引用可达图删除持久对象的过程如下：(设要删除obj)

1. 构造一个集合DEL，用以记录可删除的结点，将obj插入DEL中；

2. 从DEL中取任意结点d；

3. 如果从d到d_1有一条有向边，并且通过一条有向边可达d_1的所有其它结点都在DEL中，则将d_1插入DEL；

4. 重复第2、4步，直到集合DEL不再增大；

5. 删除DEL中的所有结点及与其关联的边。

如果在删除任意持久对象时都坚持使用以上算法，就能保证引用可达图是弱连通的。这时，持久空间的利用率最高，既保证了数据完整性，又保持了最小的数据冗余度。O_2系统就是采用这种方法来管理存贮空间的，它采用废址收集技术删除所有从持久对象（即命名对象）不可达的对象。为了保证删除时的速度，也可以仅删除一个结点，而对持久存贮空间进行定期整理。这时，引用可达图可能成为许多互不连通的子图，则所有不含system结点的子图中的结点都可以被删除。

四、引用可达图的实现方法

如前所述，基于对象标识的引用可达图是分析和管理对象持久性语义的有效逻辑工具。从实现的角度看，在系统运行的过程中维持一个引用可达图既不困难，也没有多大的开销，只要保存一个对象引用其它对象的关系和被其它对象引用次数就可以了，这与引用计数的废址收集技术非常相似。

我们让系统在创建一个持久对象的同时生成一个对象标识符UID (Unique IDentifier)，系统可以通过UID存取持久对象。显然，具有UID的持久对象是系统可引用的持久对象。当一个暂态对象由于被某一个持久对象引用而被保存到持久存贮空间时，系统为它生成一个对象标识符PID (Persistent IDentifier)，具有PID而保存在持久存贮空间中的对象不能被系统直接引用，PID实际上是一种持久存贮空间中的逻辑指针或物理指针。同时，为每一个具有PID和UID的对象设置一个引用计数器count，记录它被引用的次数，实际上，count的值就是引用可达图中结点的入度。在持久存贮空间中，我们存放除system外的所有结点，存贮方法是，为每一个结点分配一块持久存贮空间，在其中依次存放从该结点通过一条有向边可达的结点的PID、UID或值，最后存放该结点的入度。例如，图2可以按如下格式保存在持久存贮空间中：

S1	person1	person2	person3	1
S2	person4	person5	3	
person1	Peter	35	S2	1
person2	Marry	30	S2	1
person3	Jill	[illegible]	NIL	1
person4	Tom	5	NIL	1
person5	Boby	3	NIL	1

其中,S1和S2是 UID,person1,…,person5则是PID。在系统运行的不同时刻，持久存贮空间中的内容会不断变化（假设对持久对象的操作是在持久存贮空间中直接进行的),若一个具有PID的对象的引用计数值为零时,就可以被删除;UID对象的引用计数值永远是大于或等于1的，除非它被显式地删除。根据这种方法，可以将废址收集的技术有效地运用于持久存贮空间管理中。

支持持久性的面向对象系统，现在已经有许多种，但大多还是原型，对对象持久性语义的理解也莫衷一是，缺乏总结和提高。本文提出了引用可达图应用于持久对象处理的一些方法，还提出了实现引用可达图的一种方案。利用引用可达图方法来考察现有系统的持久性语义，会得出清楚的理解。引用可达图方法对于支持持久性的面向对象系统的实现，也是很有意义的。

最后，值得说明的是，引用可达图可能是一个有回路的有向图，我们的模型并未排除这种可能性。若引用可达图是一个DAG,在分析对象持久性语义时不会引起二义性，现在的大多数系统都采用了一些限制手段，以避免递归引用的情形。当引用可达图中存在回路时，需要采取必要的措施以防止死循环，这并不困难。但是，在这种情形下持久对象的物理组织中会有一些麻烦，在此不再赘述。

参考文献

〔1〕M.P.Atkinson et al., Types and Persistence in Database Programming Languages, ACM Computing Surveys, Vol.19, No.2, June, 1987

〔2〕P.Wegner, 面向对象的语言谱系，计算机科学，1990，2

〔3〕M.P.Atkinson et al., An Approach to Persistent Programming, Computer Journal, Vol.26, No.4, 1983

〔4〕O.Deux et al., The Story of O_2, IEEE Trans. on Knowledge and Data Engineering, Vol.2, No.1, March, 1990

〔5〕A.Agrawal et al.,ODE(Object Database and Environment), the Language and the Data Model, Proc. ACM SIGMOD, 1989

〔6〕S.N.Khoshafian et al., Object Identity, OOPSLA'86 Proceedings

〔7〕F.Bancilhon et al., Advances in Database Programming Languages, ACM Press, 1990

〔8〕D.D.Detlefs et al., Inheritance of Synchronization and Recovery Properties in Avalon/C++, IEEE Computer, Vol.21, No.12, Dec., 1988

维普资讯 http://www.cqvip.com