迭代算子及其在可重用软件研究中的应用*

薛锦云 吴云峰 万剑怡

(江西师范大学计算机软件研究所 南昌 330027)

摘要 迭代算子在循环控制机制抽象和可重用软件研究中有十分重要的作用,已有10多年研究历史,然而,至今对什么是迭代算子尚无确切和统一的定义,严重影响了对这一概念的理解,也阻碍了它的广泛应用,本文在分析研究现有各种迭代算子概念局限性的基础上,将迭代算子定义为包含于组合数据类型内部的抽象数据类型,得到了一种通用迭代算子模式,并给出了分立迭代算子和集成迭代算子的概念和模式,最后以实例说明了它们在可重用软件开发中的作用和使用方法.

关键词 迭代算子,抽象数据类型,可重用软件,循环机制,软件开发.

在软件开发和算法程序设计过程中,语言中的循环机制起着十分重要的作用.它被广泛应用于迭代计算和对组合数据类型中所有组成元素的访问.这样,深入研究循环控制机制,向用户提供结构清晰、可重用性好、便于使用的循环控制机制和模板就成为软件开发方法研究的重要课题.70年代后期,Pratt提出了循环控制流(Loop Control Flow)和循环控制计算(Loop Control Computation)等概念[1],进而发展成为迭代算子(Iterator)和生成算子(Generator).B. Liskov和J. Guttag在他们设计的CLU语言[2],M. Show设计的Alphard语言中[3]都提供了迭代算子;G. Booch在Ada可重用部件库中也提供了迭代算子[4];近几年M. Bishop[5],D. Lamb[6],J. Beidler[7]还对迭代算子的作用和迭代算子的规范构成方法进行了新的研究.但遗憾的是在这些文献中出现的迭代算子的概念和形式各不相同,缺乏规范统一的定义,严重阻碍了它的广泛应用.本文在分析研究现有各种迭代算子概念局限性的基础上,提出了一种新的通用迭代算子的定义和模式,并以实例说明了它在软件开发中的作用和使用方法.

1 迭代算子及其不同模式

10多年以来,人们为了提高循环程序的构成效率,有效地实现循环控制抽象和循环计算抽象,对迭代算子进行了深入研究,提出了多种迭代算子的定义和模式.

* 本文研究得到国家自然科学基金、国家863高科技项目基金和国家军用共性软件预研计划基金资助.作者薛锦云,1947年生,教授,主要研究领域为算法程序理论,软件工程,容错计算.吴云峰,1969年生,研究生,主要研究领域为算法程序理论,软件工程.万剑怡,女,1974年生,研究生,主要研究领域为算法程序理论,软件工程.

本文通讯联系人:薛锦云,南昌 330027,江西师范大学计算机软件研究所

本文1996-03-06收到修改稿

Pratt 和 J. Bishop 认为:循环程序可以由循环控制流(Loop Control Flow),循环控制计算(Loop Control Computation)和循环体(Loop Body)3 部分构成,迭代算子提供了可用于各种不同数据类型的循环控制结构框架(Template).

B. Liskov 和 J. Guttag 的定义是:迭代算子是大多数程序设计语言中所用的迭代方法的拓广,它允许用户以方便和有效的方式在任意数据类型上进行迭代计算.在 CLU 语言中,迭代算子可以用一个模块来表示.

D. Lamb 定义的迭代算子是一个模块,这种模块允许客户访问某一组合数据类型的每一个元素而隐藏了这种结构的具体表示.

G. Booch 认为:迭代算子是一种能够访问对象中所有组成元素的操作,它有主动迭代算子(Active Iterator)和被动迭代算子(Passive Iterator)这 2 种形式.

J. Beidler 对迭代算子进行过系统的研究[7~9],他认为 Booch 把迭代算子作为不同于可重用部件中构造算子(Constructor)和选择算子(Selector)的另一种算子是不妥当的,他赞成把迭代算子看成一个过渡阶段的算法结构,基于这种算法结构可以构成构造算子和选择算子.然而在最近出版的"Ada 95 原理"[10]一书中,仍然以 Booch 提出的主动迭代算子和被动迭代算子模式作为实例.

这些形式各异的定义给循环抽象机制的设计者造成了极大混乱,给理解和使用迭代算子带来了很大的困难.

综合考察上述各种迭代算子的定义,我们认为:设计和使用迭代算子的主要目的是通过对循环控制结构的抽象,有效地实现信息隐藏,提高循环程序的可靠性和可重用性,使得对组合数据类型对象中每个元素的访问可以在不了解该数据类型具体表示的情况下进行,同时又没有破坏该数据类型的内部结构.

为达到这一目标,我们给出迭代算子一种新的统一定义.

定义 1.1. 迭代算子是为访问组合数据类型对象中所有组成元素而设置的一种抽象数据类型,其中数据集中的数据用以标识引用该组合类型对象中的各个元素,操作集由构成循环控制结构的有关操作构成.

我们可以用 Ada 这种具有数据封装机制的语言实现迭代算子.下面是一个任意数据类型 any_type 的迭代算子的实例.

设 any_type 是任意数据类型,变量 t 是数据类型 any_type 的一个对象,可以用 Ada 的程序包定义一个抽象数据类型 iterator 如下:

```
Package any_type_iterator is
    Type iterator is limited private;
    procedure initialize (The_iterator: in out iterator;the_type: in any_type);
    procedure get_next (The_iterator: in out iterator);
    function value_of (The_iterator: in iterator) return item;
    function is_done (The_iterator: in iterator) return Boolean;
Private
    ... ...
end any_type_iterator;
```

其中过程 initialize 的作用是把迭代算子和变量 t 联系起来,即将 t 的首元素的索引赋予迭代算子;过程 get_next 的作用是将迭代算子指向 t 的下一个元素;函数 value_of 的值是 t 的迭代算子所指的当前元素;函数 is_done 的值为真,表示迭代算子已经访问了 t 中所有元

素. 这些操作(过程和函数)是对外可见的，可以供用户去定义一访问 t 中每一个元素的循环程序框架，不必考虑迭代算子的实际表示形式.

事实上如果我们用数组表示变量 t，则迭代算子是数组的下标值；如果用链表表示 t，则迭代算子可用链表的指针表示.

如果我们构造抽象数据类型堆栈的可重用程序部件，则堆栈的迭代算子可以看成堆栈部件内部包含的抽象数据类型.

在堆栈迭代算子中，向外界提供了 initialize，get_next，value_of，is_done 等 4 种操作，它们确实是实现循环控制所必须的. 用它们恰巧可以构成一个循环程序框架. 我们称这种迭代算子为分立式迭代算子，它和 Booch 的主动迭代算子(Active Iterator)形式上类似.

也可以用 1 个操作代替上述 4 个操作，这时候抽象数据类型 Iterator 中只有迭代算子 iterator 和迭代操作 iterate，我们可以用 Ada 的程序包实现这种类型的堆栈迭代算子.

```
Generic
    with procedure process(过程形参);
Package any_type_iterator is
    type iterator is limited private;
    procedure iterate (t:in any_type, i:in out iterator);
Private
    ... ...
end any_type_iterator;
Package body any_type_iterator is
    Procedure iterate (t:in any_type, i:in out iterator) is
    begin
        i:=t 的首元素的索引;
        while not (i=null) loop
            process (过程形参);
        end loop;
    end iterate;
end any_type_iterator;
```

其中类属过程 Process 出现在循环体中. 可以定义不同的实际过程 visit 来代替类属过程 process. 过程 visit 用来描述访问 t 中各元素的方式以及用以描述访问了 t 中各元素以后，需对这些元素所作的处理. 我们称这种迭代算子为集成式迭代算子. 可定义各种不同的 visit 过程，例如若上述类型 any_type 是二叉树，则至少可定义 3 个不同的 visit 过程分别表示对二叉树的先序、中序和后序遍历以及对应于用各种遍历方法访问元素后所作的不同处理.

仔细分析这种迭代算子的结构和功能，它和 Booch 所指的被动迭代算子(Passive iterator)在功能上相似，在形式和概念上都有很大的不同.

2　用迭代算子开发算法程序

我们分别用分立迭代算子和集成迭代算子的模式去开发快速排序的算法程序. 开发过程试图体现数据抽象、信息隐藏、自顶向下和自底向上相结合等现代软件开发方法，为此我们选用 Ada 语言作为最终可执行的算法描述语言. 为节省篇幅，我们直接引用 Gries 在文献[11]中给出的快速排序算法的抽象程序.

2.1　问　题

给定数组 $b[0:n-1]$，$n>=0$，b 中元素可为任意类型，通过定义元素间的比较运算，用

Hoare 快速排序法将 b 中元素进行升序排列.

2.2 有关符号和过程的说明

算法中引进集合类型 set，表示元素形式为二元组(i,j)的集合类型，二元组中的 i 和 j 分别代表待排序数组段的左右端点元素的下标.

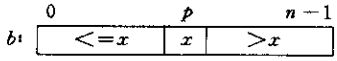

图1

定义过程 $Partition(b,i,j,p)$，它的功能是将数组$b[i:j]$分划成如图 1 的 3 段. 其中第 1 段的每个元素的值都不大于 x；第 2 段只含 1 个元素，其值为 x；第 3 段中每个元素的值都大于 x；

定义函数 $elem_of(s)$表示从集合 s 中得到一元素；运算符“−”和“∪”分别表示集合的差和并运算；“←”表示二元组的赋值运算；b 为算法的输入参数，它的类型可说明为：$b:array(0..n-1)\ of\ basetype$，其中的 $basetype$ 为待排序的数组元素类型，在具体算法中必须在头部形参表部分给出具体类型说明.

2.3 抽象算法程序

基于以上说明，Hoare 快速排序算法可用 dijkstra 的卫式命令语言描述如下：

```
s:set;i,j,p:integer;
s:=(0,n−1);
do s≠ →
    (i,j)←elem_of(s); s:=s−(i,j);
    if j−i<2 →对b[i,j]直接排序;
    □ j−i>=2 → Partition(b,i,j,p);
            s:=s∪(i,p−1),(p+1,j);
    fi
od
```

2.4 用数组实现集合(采用分立式迭代算子)

考察上述抽象算法程序，其中含有集合类型以及对集合中所有元素的访问(循环语句)，因此可以构造一个含有集合迭代算子的程序包实现上述抽象程序. 首先使用分立迭代算子模式，并用数组来表示集合. 以下是基于数组表示的集合类型说明和迭代算子说明. 不难看出这里的迭代算子实际是数组元素下标：

```
type items is array(positive range<>) of elem;
type set(the_size:positive) is
  record
    the_back:natural:=0;
    the_items:items(1..the_size);
  end record;
type iterator is new natural;
```

其中的 elem 为类属类型，表示集合元素的类型. 在本例中它是一个二元组，其中第 1 个元素表示数组段的头元素下标，第 2 个元素表示数组段的尾元素下标.

抽象算法中的集合操作与集合包中具体操作的对应关系表 1，同时，表中还说明了分立迭代算子的操作集.

表 1

	抽象算法中的操作	集合包中的操作

迭代算子操作集	elem_of(s)
	s={}
		procedure initialize(s:in out set;the_iterator:in out iterator);
		function elem_of(the_iterator:in iterator;the_set:in set)return elem;
		procedure get_next(the_iterator:in out iterator);
		function is_empty(the_iterator:in out iterator);

其它操作	s1−s2 s1∪s2	function "−"(s1:in set;s2:in set) return set; function "+" (s1:in set;s2:in set) return set;

利用 set 程序包，上述快速排序算法的可执行程序如下所示：

```
procedure absort(k:in integer;b:in out data) is
      ......
      the_iterator:iterator;
begin
      initialize(s,the_iterator);
      while not is_empty(the_iterator) loop
        i:=elem_of(the_iterator).h;
        j:=elem_of(the_iterator).t;
        s:=s−elem_of(the_iterator);
        get_next(the_iterator);
        if j−i<2 then sort(b(i),b(j));
        else
          partition(i,j,point,b);
        end if;
          if point−i>0 then
            element.h:=i;
            element.t:=point−1;
            s:=s+element;
          end if;
          if j−point−1>0 then
            element.h:=point+1;
            element.t:=j;
            s:=s+element;
          end if;
        end loop;
      end absort;
```

2.5　用链表实现集合(采用集成迭代算子)

我们还可以用链表来实现集合，并构造集合的集成迭代算子. 集成迭代算子的结构及抽象操作和集合包中具体操作的对应关系见表 2.

集合包中的有关类型说明如下：

```
type node is
      record
          the_elem:elem;
          next:iterator;
      end record;
type set is access node;
type iterator is access node;
```

表2

	抽象算法中的操作	集合包中的操作
集合的有关操作	s={} elem_of(s) s1−s2 s1∪s2	function is_empty (s:in set) return boolean; function elem_of(s:in set) return elem; function "−"(s1:in set;s2:in set)return set; function "+" (s1:in set;s2:in set) return set;
迭代算子构成成分		type iterator is limited private; generic with procedure process(the_elem:in out elem;i:in out iterator); procedure iterate(s:in set;i:in out iterator)

同样地,elem 为类属类型,表示集合元素的类型. 在这里它也是一个二元组,其中第1个元素表示链表段头元素的指针,第2个元素表示链表段尾元素的指针. 迭代算子实际上是链表元素的指针. 迭代操作过程 iterate 可定义如下:

```
procedure iterate(s:in set;i:in out iterator) is
  x:elem;
begin
  i:=s;
  while not is_empty(i) loop
    x:=elem_of(i);
    process(x,i);
    i:=i.next;
  end loop;
end iterate;
```

其中 *process*(x,i)是一个类属过程参数,可以定义一个实际过程 deal 替换 process,deal 过程完成对一个集合元素的实际处理:

```
procedure deal(the_elem:in out sortseg;s:in out iterator) is
    ......
    begin
        i:=the_elem.from;
        j:=the_elem.to;
        s:=s−the_elem;
        if j−i<2 then sort(b(i),B(j));
        else
          partition(b,i,j,p);
          if i<p then
              e.from:=i;e.to:=p−1;s:=s+e;
            end if;
            if p<j then
              e.from:=p+1;e.to:=j;s:=s+e;
            end if;
        end if;
    end deal;
```

其中类型 sortseg 表示二元组,说明如下:

```
type sortseg is
    record
        from:integer;
        to:integer;
end record;
```

这样，可执行的快速排序过程可以描述如下：

```
procedure sortall is new iterate(deal);
......
begin
    x0.from:=0;
    x0.to:=n-1;
    s0:=s0+x0;
    sortall(s0,i0);
end absort;
```

使用 Ada 的示例语句，可以得到各种不同类型的数据的排序算法程序.

具体算法程序的正确性可以用我们在文献[13～15]中给出的技术进行证明，限于篇幅，这里不再赘述.

3　总结和讨论

由上面的论述可以看出，我们定义的迭代算子，使以往众多迭代算子的概念统一起来，得到了一种通用迭代算子的模式，进而得到了分立迭代算子和集成迭代算子的概念，使得文献[1～9]中给出的各类迭代算子概念均成为它的一个特例. 首先它实现了 Pratt 和 Bishop 最初提出的思想，即循环程序可以由循环控制流、循环控制计算和循环体 3 部分构成，迭代算子提供了可用于各种不同数据类型的循环控制结构框架. 其次我们定义的迭代算子是一个模块，允许用户以方便和有效的方式在任意数据类型上进行迭代计算，它包含了 Liskov 和 Guttag 的思想. 另外，我们的迭代算子模式允许客户访问某一组合数据类型的每一个元素而隐藏了这种结构的具体表示，这是 Lamb 所希望的. 本文提出的分立迭代算子和集成迭代算子的模式实现了 Booch 定义的主动迭代算子和被动迭代算子的功能，同时也实现了 Beidler 把迭代算子看成一个过渡阶段的算法结构，基于这种算法结构可以构成构造算子和选择算子的思想. 特别是本文将迭代算子定义为抽象数据类型，使得抽象数据类型的全部理论和结果都适用于迭代算子研究，包括迭代算子的规范也可借用抽象数据类型的规范构成方法. 我们正将上述通用迭代算子的概念和模式应用于 Ada 可重用部件库的研究.

参考文献

1　Pratt T W. Control computations and the design of loop control structures. IEEE Trans. on Software Eng., 1978, **4(2)**.

2　Liskov B, Snyder A *et al*. Abstraction mechanisms in CLU. CACM, 1977, **20(8)**: 564～576.

3　Shaw M *et al*. Abstraction and verification in alphard-defining an specifying iterators and generators. CACM, 1977, **20(8)**.

4　Booch G. Software components with Ada. Benjamin Cummings, 1987.

5　Bishop J. The effect of data abstraction on loop programming techniques. IEEE Trans. on Software Eng., 1990, **16(4)**.

6 Lamb D. Specification of iterators. IEEE Trans. on Software Eng., 1990,16(12).

7 Beidler J. Structuring iterators to encourage reuse. Ada-Europe, 1993.

8 Beidler J. A role for iterators as a tool for software reuse. Proceeding of the Washington Ada Symposium, Mclean. VA, July 1992. 14~16.

9 Beidler J. Building on the booch components? What can be learned when modifying real world software tools for educational use. Proceeding of Tri-Ada'92 Orlando, FLA, 1992,11(18).

10 Ada 95. Mapping/revision team. Ada 95 Rational, Intermetrics, Inc., 1995.

11 Gries D. The science of programming. Springer-Verlag, 1981.

12 薛锦云等. 循环控制抽象和迭代算子研究. 中国抗恶劣环境专委会软件学组成立及第1届学术讨论会论文集, 1994.

13 Xue Jinyun, Gries D. Developing a linear algorithm for cubing a cyclic permutation. Science of Computer Programming, 1988,11(12):161~165.

14 Gries D, Xue Jinyun. The hopcroft-tarjan planarity algorithm. Presentations and improvements, TR88-906, CS Dept. of Cornell University, 1988.

15 Xue Jinyun. Two new strategies for developing loop invariants and its applications. Journal of Computer Science and Technologies, 1993,8(2):147~154.

ITERATORS AND ITS APPLICATION IN REUSABLE SOFTWARE STUDY

XUE Jinyun WU Yunfeng WAN Jianyi

(*Computer Software Institute Jiangxi Normal University Nanchang* 330027)

Abstract Iterators act on a very important role in the abstraction of loop control mechanism and research on reusable software. Its research history is over ten years. However until now there is no precise and unified definition about it yet, which affects one to understand its meaning heavily and hinder its widespread application. Based on analyzing and studying the limitations of a variety of concepts of iterators in this paper, the authors define iterator as a abstract data type inside a combined data type and obtain a universal iterator pattern, then present the concepts and patterns of separate iterators and integrated iterators. Finally they illustrate its roles and usage by examples.

Key words Iterator, abstract data type, reusable software, loop mechanism, software development.