杭州电子工业学院学报
第24卷第3期 JOURNAL OF HANGZHOU INSTITUTE OF Vol.24,No.3
2004年6月 ELECTRONIC ENGINEERING Jun.2004

STL和迭代器模式

李卫明,陈大金
(杭州电子科技大学软件学院,浙江 杭州 310018)

摘要:C++融合了泛型程序设计、面向对象程序设计和设计模式思想,提出了STL标准模板库。文章探讨了STL内在机制,指出迭代器是STL算法和容器的粘合剂,是STL的核心。在STL迭代器概念模型基础上,利用特性萃取机制和编译时代码分派技术,泛型算法达到了最佳运行效率。使STL成为一个具有开放架构的、强大、高效的泛型程序库。

关键词:标准模板库; 泛型编程; 迭代器; 容器; 算法
中图分类号:TP312 **文献标识码**:A **文章编号**:1001-9146(2004)03-0035-04

0 引 言

代码重用是软件工程追求的重要目标,人们在工程实践和科学研究中积累了大量成熟的数据结构和优秀算法。早期结构化程序设计支持库函数级代码重用,作为软件工程领域最受欢迎的程序设计语言之一,C++不断汲取新思想,支持包括面向对象程序设计和泛型编程在内的多种程序设计思维模型。C++底层的C内存模型保证了最原始的运行效率,对多态的支持成就了面向对象技术,templates及相关特性则展现为一种强大的泛型编程能力。标准模板库STL即是一种具有开放架构的、强大、高效的泛型数据结构和算法库。

1 迭代器模式

算法往往和相应数据结构(存储结构)联系在一起,从而形成紧密的偶合关系。假设有N个不同的算法,M种不同的数据结构,可能需要N×M个不同的算法实现代码,给泛型程序库的实现者和使用者带来极大的负担;同时,亦使库中已有的数据结构和算法难以扩充。因此,长期以来未有合适的数据结构和算法库。

软件工程中的许多设计问题可通过增加间接层方法来加以解决。Gang Of Four归纳了经常遇到的软件设计问题,形成了可再现的解决方案,提出了23种设计模式。其中有一种行为型模式称为迭代器模式,容许算法使用一个标准接口,即迭代器,依序访问一个数据集。

STL应用迭代器模式,提供了vector、list、deque等序列式容器和set、map、multiset、multimap等关联式

收稿日期:2002-04-06
作者简介:李卫明(1965-),男,浙江杭州人,讲师,面向对象技术和图像处理.

容器,实现了各种常用数据结构和泛型算法;同时,每一种 STL 泛型容器都提供了各自相应的迭代器,作为容器的接口,算法通过迭代器访问容器。STL 算法和容器(数据结构)得到了分离,打破了算法和数据结构的偶合关系,迭代器成为泛型容器和泛型算法的粘合剂。STL 实现者无须再为不同的容器分别编写不同的算法实现代码,STL 使用者可以自行增加新泛型容器、新泛型算法,使 STL 具有了开放构架。

下述代码段展示了 STL 中泛型容器、泛型算法和迭代器的强大威力,它先从标准输入流中读入系列名字,存入向量容器中,然后,应用 STL 算法将向量容器内容拷贝到集合容器,再排序向量容器中名字,将集合容器和向量容器内容拷贝到标准输出流,得到排序好的名字序列。

```
vector < string > V; //向量容器
set < string > S; //集合容器
string name;
while (getline (cin, name)) //标准输入流中读入名字
V.push-back (name); //存入向量容器
copy (V.begin(), V.end(), inserter (S, S.begin()));//向量容器拷贝到集合容器
sort (V.begin(), V.end ());//排序向量容器名字
//将集合容器和向量容器内容拷贝到标准输出流
copy (S.begin(), S.end (), ostream.iterator < string > (cout, "/t"));
copy (V.begin(), V.end (), ostream.iterator < string > (cout, "/t"));
```

2 迭代器概念模型

事实上,C/C++指针类型正是这样一个内建迭代器。C/C++程序可以通过指针访问 C/C++内建容器——数组。C/C++指针可以进行复制、提领、提领赋值、++、--、+/-整数、[]、比较等多种运算。

作为算法和容器的粘合剂,STL 迭代器是一个泛型指针,是具有某种类型的指向其它类型对象的对象。并非所有迭代器都需要支持所有的指针运算,这既无必要,也无可能。STL 使用抽象概念描述算法需求和容器接口,只要某个类型满足某组条件,就说此类型符合某个概念,是它的一个模特。定义抽象的概念,并根据抽象的概念来撰写算法与数据结构,正是泛型编程的本质。泛型算法对于参数类型的大部分假设,都可以由符合某概念来描述,迭代器也不例外。依据用途不同,STL 将迭代器概念分为 5 类:输入迭代器,用于刻划输入;输出迭代器,用于指示输出;前向迭代器,可以单向前移;双向迭代器,可以双向移动;随机存取迭代器,支持随机存取。STL 详细规定了各类迭代器概念需要支持的运算,C/C++指针类型正是一个随机存取迭代器概念的模特。各类迭代器概念间存在纯抽象的强化关系,它不同于面向对象程序设计中的继承关系。STL 迭代器概念强化关系阶层模型如图 1 所示。

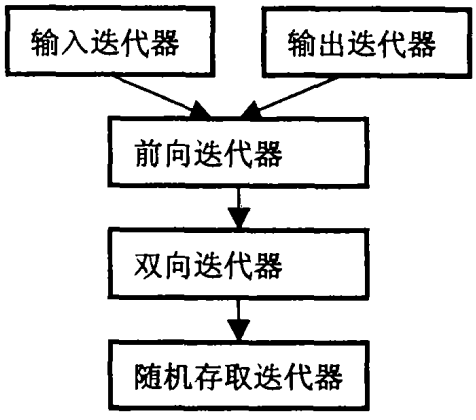

图 1 STL 迭代器概念强化关系阶层模型

STL 各容器根据自身特点,负责提供某种相应类型迭代器。如基于单链表容器提供前向迭代器;基于双向链表容器提供双向迭代器;vector 和 deque 则提供随机存取迭代器。算法则依据自身需要决定使用何种类型迭代器,处于强化关系阶层模型中的下层迭代器可以代替上层迭代器使用。

3 特性萃取机制

作为第一个标准规格的泛型程序库,STL 不仅具有开放的架构,强大的能力,同时也是一个高效可重用的代码生成器;使用 STL 生成的代码运行效率可以完全不逊色于手工精制的代码。每一个 STL 泛型算法、每一个 STL 泛型容器的操作行为,其复杂程度都有明确规范,通

常是最佳效率或极佳效率。

STL 规定各类迭代器支持的所有本身运算的时间复杂性都在可分摊的常数时间内;此外,各泛型算法可利用 C++ templates 特化、偏特化规则,根据使用的迭代器类型,特化、偏特化算法代码,达成最佳效率。任何迭代器类型,包括自定义迭代器类型都具有多种特性,STL 建立了下列特性萃取模板类 iterator_traits,可以萃取出各类迭代器(包括内建指针类型)自身各种特性。

```
template < class iterator >
struct iterator_traits {
    typedef iterator::iterator_category iterator_category ; //迭代器标记特性类型
    typedef iterator::value_type value_type ; //迭代器值类型
…};
template < class T> // iterator_traits 指针偏特化版本
struct iterator_traits<T* > {
    typedef random_access_iterator_tag iterator_category;
    typedef T value_type;
…};
```

这样,泛型算法就可以根据需要,萃取出迭代器类型(包括指针)标记特性,分派算法,使算法既具有泛型特点,又具有最佳效率的实现。下面以 advance 算法为例进行说明 :

```
template<class InIt, class Dist>
inline void advance(InIt& it, Dist n) { //移动迭代器
    advance (it, n, typename iterator_traits<InIt>::iterator_category()); //编译期分派
}
```

与迭代器概念强化关系阶层模型相对应,迭代器标记特性类间存在继承关系。所有迭代器 iterator_category 标记特性类型,是下列 5 个 STL 空标记类之一:

```
struct input_iterator_tag {};
struct output_iterator_tag {};
struct forward_iterator_tag : public input_iterator_tag {};
struct bidirectional_iterator_tag : public forward_iterator_tag {};
struct random_access_iterator_tag : public bidirectional_iterator_tag {};
```

上述各标记类并无任何数据成员,不占用任何存储空间,纯粹用来规范行为。

根据 C++ 模板函数重载分派规则,使用下层迭代器的算法可以分派至使用上层迭代器的函数;本例中,使用前向迭代器的 advance 算法将分派至使用输入型迭代器的 advance 函数。同时,可以根据各类迭代器特点,提供对应下层迭代器算法的最佳实现。

```
template < class InputIterator, class Dist> //同样适用前向迭代器 advance 算法
inline void advance(InputIterator it, Dist n, input-iterator-tag) {
    for ( ; n > 0; --n, ++i) {} //只能前进
}
template < class BidirectionalIterator, class Dist>
inline void advance(BidirectionalIterator it, Dist n, bidirectional_iterator_tag) {
    if (n> = 0)
      for ( ; n > 0; --n, ++it) {} //前进
    else
      for ( ; n < 0; ++n, --it) {} //倒退
}
template < class RandomAccessIterator, class Dist>
inline void advance(RandomAccessIterator it, Dist n, random_access_iterator_tag) {
    it += n; //发挥随机存取迭代器的最佳效率
}
```

advance 算法具有统一接口,它利用特性萃取机制萃取出各类迭代器的标记类特性,然后利用 C+

+模板偏特化机制分派恰当处理代码。这一切都在编译期完成,丝毫不影响运行效率。

4 结束语

迭代器是STL的核心。利用迭代器,STL构成了一个开放的架构,C++程序员不仅可以将标准泛型算法和泛型容器搭配使用,而且可以自行设计泛型容器、泛型算法,配合标准算法、容器使用。此外,利用配接器和函数对象,可以大大增强STL算法、迭代器、容器能力,形成一个具有威力强大的泛型库。

参考文献

[1] Bjarne Stroustrup. The C++ Programming Language[M].北京:高等教育出版社,2001.429-579.
[2] Matthew H Austern.侯捷译. 泛型编程与STL[M]. 北京:中国电力出版社,2003.81-166.
[3] Andrei Alexandrescu.侯捷,於春景译. C++设计新思维[M]. 武汉:华中科技大学出版社,2003.3-20.
[4] Nicolai M Josuttis.侯捷,孟岩译. C++标准程序库[M]. 武汉:华中科技大学出版社,2002.143-435.
[5] 侯捷. STL源码剖析[M]. 武汉:华中科技大学出版社,2002.79-113.
[6] James W Cooper.张志华,刘云鹏译. C#设计模式[M]. 北京:电子工业出版社,2003.197-203.

STL and Iterator Design Pattern

LI Wei-ming, CHEN Da-jin

(*School of Software, Hangzhou Dianzi University, Hangzhou Zhejiang* 310018, *China*)

Abstract: Combined generic programming, object oriented programming and design pattern theories, C++ brings standard template library(STL). This article probes the mechanism involved in the STL, points out iterator, the glue of STL algorithms and containers, is kernel of STL. Based on iterator concepts model, traits extraction mechanism and compile time dispatch technic, STL generic algorithms achieve best efficiency. So, STL becomes a open frame, high performance and powerful generic library.

Key words: STL; generic programming; iterator; container; algorithm