STL的基本概念和常用容器

张阜东
2007.10

STL基本概念

- 什么是STL
- STL历史
- STL版本
- 六大组件

2

什么是STL

- STL（Standard Template Library），即标准模板库，是一个高效的C++程序库。
  - 被容纳于C++标准程序库（C++ Standard Library）中，是ANSI/ISO C++标准中最新的也是极具革命性的一部分。
  - 包含了诸多在计算机科学领域里常用的**基本数据结构**和**基本算法**。为广大C++程序员们提供了一个可扩展的应用框架，高度体现了软件的可复用性。

3

STL历史

- 1971 : David R. Musser 开始倡导 Generic Programming 概念。
- 1979 : Alexander Stepanov 创造 STL
- 1987 : Alex 和 Musser 开发出一套 Ada library
- ???? : Alex 先后在 AT&T 及 HP实验室以 C 及 C++实验大量的体系结构和算法形式。
- 1992 : Meng Lee 加入称为另一位主要贡献者
- 1993/11 : Alex 于 ANSI/ISO C++ 会议展示
- 1994 夏 : STL 被纳入 C++标准

4

STL的不同实现版本

- STL是一个标准，各商家根据这个标准开发了各自的STL版：
  - **HP STL**：全世界所有的STL 实品，都源于 Alexander Stepanov 和 Meng Lee 完成的原始版本，现在已经很少用了
  - **SGI STL** ：STL之父Alexander Stepanov离开HP之后就去了SGI,SGI STL设计者和编写者包括 Alexander Stepanov和Matt Austern。它属于开放源码，被GCC所采用。由于GCC对C++语言标准的支持很好，SGI STL在linux平台上的性能相当出色。此外，其源代码的可读性也很好。
    在学习或实用中，SGI STL应是首选。

5

STL版本cont.

- **STL**port：旨在将SGI STL的基本代码移植到各种主流编译环境中，使各种编译器的用户都能够享受到SGI STL的先进之处。最新的STLport 可以从www.stlport.org免费下载, 可以支持向各种主流C++编译环境的移植。在C++ Builder中使用的就是STLport。
- **P.J. STL** : VC6.0里的STL，作者P.J. Plauger，所以一般也说P.J. STL。
- **Rouge Wave STL** : Rogue Wave Software, Inc.
  可读性较好。是HP STL的一个继承版本。 是C++ Builder 5.0及以前版本采用的STL实现 。

6

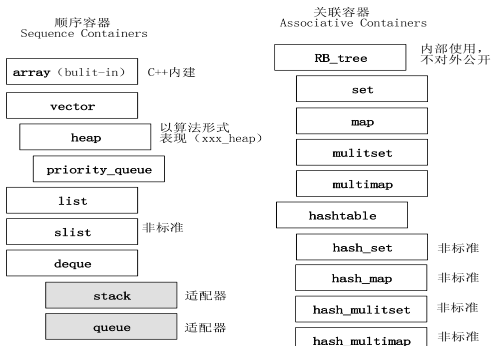
STL组件

- Container(容器) 各种基本数据结构
- Adapter(适配器) 可改变containers或function object接口的一种组件
- Algorithm(算法) 各种泛型算法如sort、search…等
- Iterator(迭代器)* 连接containers和algorithms
- Function object(函数对象) *
- Allocator(分配器)*

STL中的容器

- 顺序容器
  vector、deque、list
  非标准的顺序容器：slist、rope
- 关联容器
  set、multiset、map、multimap
  非标准的关联容器：
  hash_set、hash_multiset、hash_map、hash_multimap

容器谱系图



vector

实际上就是个动态数组
  自动分配内存
随机存取能在常数时间完成
  像数组一样可以使用下标访问元素
  小心，不要数组越界
在尾端增删元素具有较佳的性能
  其他位置的增删操作和插入操作都不好
  需要把待插入元素右边的每个元素都拷贝一遍

vector如何分配内存

- 分配新的内存块，它有容器目前容量的几倍。在大部分实现中，vector和string的容量每次以2为因数增长。也就是说，当容器必须扩展时，它们的容量每次翻倍。（也有使用别的策略的）
- 把所有元素从容器的旧内存拷贝到它的新内存。
- 销毁旧内存中的对象。
- 回收旧内存。

vector如何分配内存

- 使用reserve()

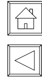
```
vector<int> v;
v.reserve(1000);
for (int i = 1; i <= 1000; ++i)
    v.push_back(i);
```

  这样就不会频繁的重新分配内存
- 交换技巧缩减过剩的容量

```
vector<int>(v).swap(v);
```

  这个操作是高效的

list



- 双向链表。

1. 任意位置插入和删除元素的效率都很高

   指针必须被重新赋值，但是，不需要用拷贝元素来实现移动
2. 对随机访问的支持不好

   访问一个元素需要遍历中间的元素
3. 每个元素还有两个指针的额外空间开销

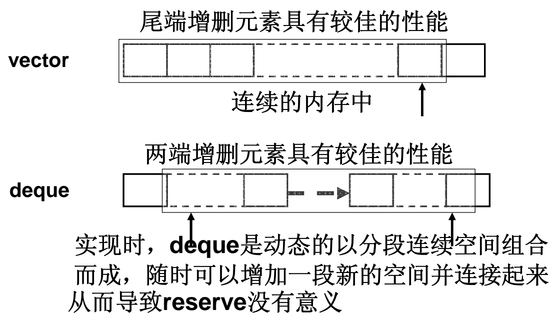
deque

- 也是个动态数组
- 随机存取任何元素都能在常数时间完成(但次于vector)
- 在两端增删元素具有较佳的性能
- 和vector的区别

  多了push_front() pop_front()

  少了capacity() reserve()

deque



实现时，**deque**是动态的以分段连续空间组合而成，随时可以增加一段新的空间并连接起来从而导致**reserve**没有意义

deque的用途

- 可以把deque看作vector和list的折中
- 大部分的时候还是使用vector和list
- 大部分插入和删除发生在序列的头或尾时可以选择deque这种数据结构
- 默认实现stack ,queue

  还是因为需要折中

  lish每次push，pop太慢

  vector数组增量的时候太慢

如何选择顺序容器

- 需要频繁在序列内部位置上进行插入和/或删除操作且不需要过多地在序列内部进行长距离跳转，应该选择？ 链表
- 仅仅出于可动态改变大小的原因，应该选择？

  向量和双端队列

如何选择顺序容器

- 三种成员中的任何一种都无法在所有操作的性能上优于其他两种。
- 在何种情况下其中的一种容器要优于其他两种？

关联容器

- 通过保存在数据项中的索引项，尽可能快速检索数据项。
- 实现关系检索的一般方法
  - 平衡二叉查找树（对数时间 性能可靠）
  - 散列（常数时间）
- STL标准库中只包含关联容器set、multiset、map、multimap
  实现：RB-tree(红黑树)
  - http://dev.csdn.net/develop/article/19/19794.shtm
- 非标准的hash_set、hash_map、hash_multiset、hash_multimap

rb-tree系列关联容器

- set, multiset
  数据项就是索引项。
  多集允许出现重复的索引项。
- map, multimap
  数据项是由索引项和其他某种类型的数据组成的一对数据。
- set<T> T必须包含偏序关系
  也就是说，必须定义元素间的比较函数
  less(t1,t2)

map的插入和更新

map<string, string> m;

string key1, key2, v1, v2;

- 插入
  m.insert(make_pair(key1,v1));
- 更新
  m[key2] = v2;
  如果key2不存在则插入一个(key2,v2)

集合（删除）

set<char>::iterator i=set1.find(‘e’);

set1.erase(i);

i=set1.find(‘1’);

j=set1.find(‘9’);

set1.erase(i, j);

集合（访问）

- 集合和多集的find成员函数的时间复杂度为O(logN)
  算法find的时间复杂度为O(N)。
  - 成员函数利用索引的有序排列进行对分查找
  - find算法线性查找
  - 所以尽量使用成员函数

散列基本思想

- 一个确定的函数关系h
- 以结点的关键码K为自变量
- 函数值h(K)作为结点的存储地址
- 检索时也是根据这个函数计算其存储位置
  - 通常散列表的存储空间是一个一维数据
  - 散列地址是数组的下标

Hash系列关联容器

- 用法和rb-tree系列容器基本相同
- 时间复杂度为O(1)，比O(logn)快，但是更浪费空间
- 不需要比较元素大小的函数
- 但是需要对元素的hash函数
  常用的string，数值有默认的hash函数
- 寻找一个快速的hash函数是hash系列容器效率的关键
  例如对一个文本全文进行hash就不一定能得到好的效率，因为文本比较函数是个很快返回的函数

25

Adapter适配器

- 一种STL组件，作用改变其他组件的接口
- 以模板类的形式定义，并且以另一种组件的类型作为参数
- STL提供了三种适配器组件
  - 容器适配器（栈、队列、优先级队列）
  - 迭代器适配器*
  - 函数适配器*

26

容器适配器

- 只提供非常有限的接口。
  - 比如尽管可以选择list作为栈的实现方式，但如果不对其接口进行限制，则很难保证不会由于疏忽而引入某些栈的基本操作之外的操作。

27

Stack栈

- 在一端的插入操作
- 在同一端的删除操作
- 在同一端获取元素的值
- 测试栈是否为空
- stack容器适配器可以用于vector、list和deque

28

Stack栈

stack<T>是类型为T的栈，默认情况下以deque实现

stack<T, vector<T>>是类型为T的栈，以vector实现

stack<T, list<T>>是类型为T的栈，以list实现

stack<T, deque<T>>是类型为T的栈，以deque实现

29

更多

- 队列queue
- 优先级队列priority_queue

30

Algorithm算法

- STL提供了大量的算法，用来对定义在STL框架内的数据结构进行操作。
- 每个算法都适用于若干种不同的数据结构，而不是仅能用于一种数据结构。
- 算法变体：原地形式、复制形式、判断函数参数的形式
- 算法分类：非可变序列算法、可变序列算法、排序相关算法、通用数值算法。

31

Algorithm算法

- 原地形式，指算法把它对容器操作的结果存放在同一个容器
- 复制形式，指算法将操作结果赋值到另外的容器中，或者复制到被操作容器的另一个不重叠的部分中。
  - STL根据算法的复杂性决定是否包含算法的复制形式。不提供sort_copy，提供reverse_copy。

32

排序类算法

- sort
  给定范围内排序
- stable_sort
  不破坏相等的情况下的原元素的顺序
- partial_sort
  只对最小的前n个元素排序
- nth_element
  只得到最小的n个元素
- partition
  把满足某个条件的元素移到前端
- stable_partition

33

sort,nth_element举例

```
vector<int> v;
for(int I = 10; I > 0; --I)
  v.push_back(i);
vector<int>::iterator start = v.begin();
vector<int>::iterator end = v.end();
//将最小的四个数放到前面，第五个数刚好是第五小的
nth_element(start, start+4, end) ;
//全排序
sort(start,end);
```

34

关于堆的算法

- 堆是满二叉树
- 最大堆：就是每个节点都比它的子节点大，但同层节点相互之间没有偏序关系
- 堆可以用于找出一个序列n个数中找出最大的m个数，仅使用m数量级的空间，时间复杂度只有O(nlogm)
- 因为是满二叉树，堆一般使用数组实现
  父节点下标i ,则子节点下标就是i*2+1,i*2+2
- 主要相关算法
  make_heap() push_heap()
  pop_heap() sort_heap()

35

heap举例

```
vector<int> v;
……
make_heap(v.begin(), v.end());
//make_heap用于将一个vector构造为一个堆
v.push_back(num);
//给数组末端增加一个元素
push_heap(v.begin(), v.end());
//将数组最后一个元素加入堆
pop_heap(v.begin(), v.end());
//弹出堆顶元素，放到数组最后
v.pop_back();
//删掉数组最后一个元素
```

36

函数对象

- 函数对象是一个类，重载了函数调用操作符[operator()]
- 一个简单的比较函数对象的例子

```
struct Posting_less
{
    bool operator() (const Posting* lhs,
                     const   Posting* rhs) const
    {
            return lhs->term < rhs->term;
    }
};
```

- 对于sort，set等需要自定义偏序关系函数都可以写成以下形式
- Hash函数也类似处理

37

函数对象的优点

- 函数对象最重要的功能还是实现比较函数，为什么不直接使用函数指针？
- 相对函数指针的优点：

1. 可以内联编译，性能好

    大部分编译器不会试图去内联通过函数指针调用的函数

2. 函数对象可以包含任意的额外数据

38

STL的效率

- 请相信STL一定是用复杂度最好的算法实现的
- 由于需要兼顾通用，STL会比自己专门为特定需求实现的函数慢一点

    所以ACM题有时候不适合使用STL

39

如何学好STL

- 相信STL，使用STL
- 多多Google Baidu
- 阅读STL源码
- 扩充STL*

40

STL参考数目

- 标准模板库自修教程与参考手册—STL进行C++编程, D.R.Musser, G.J.Derge, A.Saini.
- STL源码剖析, 候捷. //SGI STL
- Effective STL ， Scott Meyers
- C++大学教程, H.M.Deitel, P.J.Deitel.
- C++语言的设计和演化, Bjarne Stroustrup.
- 设计模式：可复用面向对象软件的基础, Erich Gamma 等.

41