几种 STL 容器的基本用法[资料]

常兴龙
天津大学计算机学院
QQ:286259397 MSN:cxl82116@msn.com

一、原型与构造函数

Vector 的原型可定义为
vector<T, allocator <T> >
其构造函数为
vector()　　//空的
vector(al)　　//指定一种 allocator
vector(n)　　//用默认 T()初始化 n 个元素
vector(n, val)　//用 Val 初始化 n 个元素
vector(n,val,al)　　//用 val 初始化 n 个元素，用 al 做分配器
vector(first,last)　　//从已有的 first 到 last 复制生成
vector(first,last,al)　//从已有的 first 到 last 复制生成，用 al 做分配器

二、操作

1.开辟 N 个空间
vecobj.reserve(N);
2.当前（重新分配内存前）得到最大容量
capacity();
3.重新分配内存为 N
resize(N)
如果变小，则删除多余。如果变大，则用 T()添充
4.清空
clear();
注意，clear()和 resize()都不一定使得 vector 变小，若欲释放内存，请使用 vecobj.swap(vector<T, A>())
5.存取首尾元素
front()与 back()操作,取后一个和最前一个元素，注意其返回是引用,其而是左值(l_value),因此可以赋值. 做类似于 vecobj.front() = 3;的操作,但要保证 front 空间有效，否则形为无法预测。
6.取值
[]与 at 可以做此操作，at 会检查，如果越界有会 out_of_range 的异常被 throw
7.push_back, pop_back
要保证不为空

8.使用 assign

assign 可以改变大小和初值，大小是随意的，不受开始时大小的限制，如果设置为 0,则清空。

assign(5,0)把 vector 改为 5 个大小，并用 0 添充

assign(iax+3,iax+5); 从数组第 4 到 5 个填充，注意左闭右开，即可取到 iax[3]与 iax[4]

9.使用 insert

insert(it, x),在 it 前插入一个元素 x

insert(it,first,last),在 it 前插入一个序列[first,last)左闭右开

10.使用 erase

erase(it)删除在 it 处的元素，返回值为下一元素。如果 intVec.erase(intVec.end());并不会报错，

如果删除一个序列[first,last)，使用 erase(first,last)

11.BVector 是 vector<bool>的特化版，具体的用途有待查证

12.flip()把某一元素，求反。如 vecObj[i].flip();

13.swap. vecObj.swap(vecObj[i],vecObj[j]);

若要在容器中装一个对象并且能并检索，需要重载 operator == ,如下：

```
#include <vector>
#include <iostream>
#include <stdlib.h>
#include <time.h>
//#include <getopt.h>
using namespace std;

class Obj
{
public:
	Obj(int x, int y, int z)
	{
		this->x = x;
		this->y = y;
		this->z = z;
	}
	bool operator == (const Obj & obj)
	{
		if(obj.x == x && obj.y == y && obj.z == z)
			return true;
		return false;
	}
	int getX()
	{
		return this -> x;
	}
private:
	int x;
	int y;
	int z;
```

```
};

int main(int argc, char * argv[])
{
	vector<Obj> vecObj;
	Obj obj1(2,3,4);
	Obj obj2(4,5,6);
	vecObj.push_back(obj1);
	vecObj.push_back(obj2);

	vector<Obj>::iterator it =find(vecObj.begin(),vecObj.end(),Obj(2,3,4));
	if(it != vecObj.end())
		cout << (*it).getX() << endl;
	return 0;
}
```

list 的基本用法

与 vector 的用法基本相同，其中需要强调一点的是 splice()函数，是指把指定段的另一个 List 插入到指定位置的前面。

splice(iterator it , list &x)

splice(iterator it, list &x, iterator first)

splice(iterator it,list &x, iterator first, iterator last)

一、原型与构造函数

typdef list<T, allocator<T> > listObj;

构造函数

list() //空

list(al) //指定 allocator 的空表

list(n)//n 个元素，所有元素都是 T()出来的

list(n,val)//n 个元素，所有元素都是 T(val)出来的

list(n,val,al)//同上，并指定 allocator 为 al

list(first, last) //复制构造

list(first,last,al) //指定 allocator 构造

二、操作

1.resize & clear

使用 resize(n)改变大小，使用 resize(n, val)如果需要用 T(val) 来填满空闲值。

2.front ()& back()

如果 listObj 非常量对象,返回是一个左值函数

3.插入操作

insert(iterator it , val)

insert(iterator it, first, last)

insert(iteratot it, n, x)//插入 n 个 x

4.移除

remove(x); //vector.erase(integrator it)

按值删

```
int iax[] ={3,4,5,6,6,7,8};
list<int> lObj;
lObj.insert(lObj.begin(),iax, iax + 7);
lObj.remove(6); //
```

按函数条件删

```
#include <iostream>
#include <list>
using namespace std;
// a predicate implemented as a function:
bool single_digit (const int& value) { return (value<10); }
// a predicate implemented as a class:
class is_odd
{
public:
   bool operator() (const int& value) {return (value%2)==1; }
};
int main ()
{
   int myints[]= {15,36,7,17,20,39,4,1};
   list<int> mylist (myints,myints+8);      // 15 36 7 17 20 39 4 1
   mylist.remove_if (single_digit);           // 15 36 17 20 39
   mylist.remove_if (is_odd());                 // 36 20
   cout << "mylist contains:";
   for (list<int>::iterator it=mylist.begin(); it!=mylist.end(); ++it)
      cout << " " << *it;
   cout << endl;
   return 0;
}
```

当然，对于 class is_odd，也可以写成

```
template <class T>
class is_odd
{
};
```

调用时，则要改成

```
mylist.remove_if(is_odd<int>());
```

5.unique 操作

```
// list::unique
#include <iostream>
#include <cmath>
#include <list>
using namespace std;

// a binary predicate implemented as a function:
bool same_integral_part (double first, double second)
{ return ( int(first)==int(second) ); }

// a binary predicate implemented as a class:
class is_near
{
public:
  bool operator() (double first, double second)
  { return (fabs(first-second)<5.0); }
};

int main ()
{
  double mydoubles[]={ 12.15,  2.72, 73.0,  12.77,  3.14,
                       12.77, 73.35, 72.25, 15.3,  72.25 };
  list<double> mylist (mydoubles,mydoubles+10);
  //UNIQUE 以前必须要 Sort,切记,它的内部实现是 I,i+1 的方式。
  mylist.sort();              //  2.72,  3.14, 12.15, 12.77, 12.77,
                              // 15.3,  72.25, 72.25, 73.0,  73.35

  mylist.unique();            //  2.72,  3.14, 12.15, 12.77
                              // 15.3,  72.25, 73.0,  73.35

  mylist.unique (same_integral_part);  //  2.72,  3.14, 12.15
                                       // 15.3,  72.25, 73.0

  mylist.unique (is_near());           //  2.72, 12.15, 72.25

  cout << "mylist contains:";
  for (list<double>::iterator it=mylist.begin(); it!=mylist.end(); ++it)
    cout << " " << *it;
  cout << endl;

  return 0;
}
```

6.排序操作
sort(); //默认按 operator <排序，从小到大
sort(pr); //pr 为 Functional 函数
7.Merge 操作
在 merge 操作前，需要对两个序列都用 operator <排序，当然，也可以指定 pr 排序函数
merge(s2)
merge(s2,pr);
8.reverse()
翻转操作,把整个 list 翻转

deque 的基本操作

一、原型与构造函数

typedef deque<T, allocator<T> > deqObj;
构造函数
deque();
deque(al);
deque(n);
deque(n,x);
deque(n,x,al);
deque(first,last);
deque(first,last,al);

二、操作

1.resize & clear
使用 resize(n)改变大小，使用 resize(n, val)如果需要用 T(val) 来填满空闲值。
2.clear 操作
在 clear 后调用 deqObj.swap(deque<T,A>())是好习惯，而且也一定要这么做。
3.font(),back(),operator []，(如出边界，形为未定)at()(如出边界，抛异常),push_back(),push_front(),pop_back(),pop_front(),insert(iterator it,x),insert(iterator it,n,x),insert(iterator first,iterator last),(插入后指向刚插入的值)，erase(it),删除在 it 指定位置的值，erase(iterator first,iterator last)删除指定区间的值(左闭右开)。这些操作与上面的操作雷同。

Set 与 multiset 的基本操作

一、原型与构造函数

typedef set<Key, less<Key>, allocator<key> > setObj;
构造函数
set(); //空 set,按 pred()排序
set(pr); //声明一个空的按 pr 排序的 set
set(pr,al); //声明一个按 pr 排序的集合用 al 分配
set(first,last)
set(first,last,pr)
set(first,last,pr,al)
操作
1.clear()
2.erase(it); erase(first, last)
3.insert(key),返回值为 pair<iterator, bool> 类型，没有与插入元素相同的元素时，second 为 true,此时 first 指向新插入的元素。否则为 False，first 仍指向原来的元素
4.find(key)
5.lower_bound(key)
6.upper_bound(key)
7.equal_range(key),返回一个 pair<iterator , iterator >(lower_bound(key), upper_bound(key))
8.count, equal_range 的长度
9.key_comp,如果 k1 排在 k2 的前面，那么 key_comp()(key1,key2)就为 true
10.value_comp,对于 set<key>对象，它与 key_comp 一样。
multiset
1.insert,由于 insert 总能成功，那么它返回的就是新元素的迭代器，而并非 pair<iteraor, bool> 对象.
2.find 返回第一个与 key 相等的迭代器。
3.equal_range 将返回 [0,setObj.size())的任意长度.
4.count()将返回[0,setObj.size())的任意值。