第十章 标准模板库STL

一、标准模板库STL

1.STL中的基本概念

标准模板库(Standard Template Library)就是一些常用的数据结构和算法的模板的集合。有了STL，不必再写大多的标准数据结构和算法，并且获得非常高的性能。

1.容器：可容纳各种数据类型的通用数据结构，是类模板；

2.迭代器：可用于依次存取容器中的元素，类似于指针；

3.算法：用来操作容器中的元素的函数模板

1）sort()来对一个vector中的数据进行排序

2）find()来搜索一个list中的对象

算法本身与他们的操作的数据的类型无关，因此他们可以在从简单数组到高度复杂的任何数据结构上使用。

```
int array[100];
sort(array,array+70);//将前70个元素排序
```

该数组就是容器，而int *类型的指针就可以作为迭代器，sort算法作用于数组，对其进行排序。

2.容器概述

可以用于存放各种类型的数据（基本类型的变量，对象等）的数据结构，都是类模板，分为三种：

1）顺序容器：vector，deque，list；

2）关联容器：set，mutiset，map，multimap；

3）容器适配器：stack，queue，priority_queue；

(1)vector向量

头文件¡vector¿

1.本质：对数组的封装。

2.特点：能够在随机读取数据的时候在常数时间内完成

3.初始化vector对象的方式

```
vector<T> v1;       //vector保存类型为T的对象。默认构造函数v1为空
vector<T> v2(v1);   //v2是v1的一个副本
vector<T> v3(n,i); //v3包含n个值为i的元素
vector<T> v4(n);    //v4包含有值初始化元素的n个副本
```

具体代码：

```
vector<int>ivec1;
vector<int>ivec2(ivec1);
vector<string>svec1;
vector<string>svec2(ivec);
vector<int>ivec4(10,-1);      //10个-1的元素初始化ivec4
vector<string>svec(10,"hi"); //10个hi字符串初始化svec
```

4.vector常用函数

```
empty()          //判断向量是否为空
begin()          //返回想向量迭代器首元素
end()            //返回向量迭代器末元素的下一个元素
clear()          //清空向量
front()          //第一个数据
back()           //最后一个数据
size()           //获得向量数据大小
push_back(elem)//将数据插入向量尾
pop_back()     //删除向量尾部数据
```

实例如下：

```
int main(void)
{
   vector<int>vec;
   vec.push_back(10);       //当前元素尾部插入一个元素10
```

```
    vec>push_pop();          //抹掉
    cout<<vec.size()<<endl; //输出数据大小
    return 0;
}
```

遍历数组:

```
for(int k=0;k<vec.size();k++)
{
  cout<<vec[k]<<endl;
}                          //也可以用迭代器来遍历数组
```

(2)迭代器iterator

1）用于指向顺序容器和关联容器中的元素

2）迭代器的用法和指针类似

3）有const和非const两种

4）通过迭代器可以读取它指向的元素

5）通过非const迭代器还能修改其指向的元素

6）定义方法:

```
容器类名: : iterator   变量名:
容器类名: : const_iterator   变量名
* 迭代器变量名    //访问一个迭代器指向的元素
```

迭代器遍历数组元素:

```
int main(void)
{
   vector vec;
   vec.push_back("hello");
   vector<string>::iterator citer=vec.begin();
   for(;citer!=vec.end();citer++)
      {cout<<*citer<<endl;}
   return 0;
}
```

(3)链表list

头文件¡list¿

1.特点：数据插入的速度快,双向链表，元素在内存不连续存放，在任何位置增删元素都能在常数时间内完成。不支持随机存取。

2.使用方法：与vector使用方法相似

(4)关联容器

1.元素是排序的

2.插入任何元素，都按照相应的规则来确定其位置

3.在查找时具有非常好的性能

4.通常以平衡二叉树方式实现，插入和检索的时间都是0(log(N))

(5)映射map

头文件¡map¿

通过map的键key找到它的值value

程序示例1：

```
map<int,string>m;
pair<int,string>p1(10,"shanghai");//键为10，值为shanghai
pair<int,string>p2(20,"beijing");
m.insert(p1);
m.insert(p2);
cout<<m[10]<<endl;
cout<<m[20]<<endl;
```

程序示例2：

```
map<string,string>m;
pair<string,string>p1("S","shanghai");//键为"S"，值为shanghai
pair<string,string>p2("B","beijing");
m.insert(p1);//将p1插入m里面
m.insert(p2);
cout<<m["S"]<<endl;
cout<<m["B"]<<endl;
```

代码实例一（vector用法）：

```
#include<iostream>
#include<stdlib.h>
#include<vector>
#include<list>
#include<map>
//通过使用标准模板库，学习vector用法
using namespace std;
int main()
{
  vector<int>vec;
  vec.push_back(3);//从当前向量的尾部插入的
  vec.push_back(4);
  vec.push_back(6);
  //vec.pop_back();//将尾部的元素删除了
  //cout<<vec.size()<<endl;
/*for(int i=0;i<vec.size();i++)//遍历数组
{
  cout<<vec[i]<<endl;
}*/
/*vector<int>::iterator itor=vec.begin();
//cout<<*itor<<endl;//通过指针打印
for(;itor!=vec.end();itor++)//初始条件就是 =vec.begin()//最后一个元素的下一个位置
{
  cout<<*itor<<endl;
}*/
cout<<vec.front()<<endl;//第一个数据
cout<<vec.back()<<endl;//最后一个数据
  return 0;
}
```

代码实例二（list用法）：

```
#include<iostream>
#include<stdlib.h>
#include<vector>
#include<list>
#include<map>
//通过使用标准模板库，学习list用法
using namespace std;
int main()
{
  list<int>list1;
  list1.push_back(4);
  list1.push_back(7);
```

```
  list1.push_back(10);
  /*for(int i=0;i<list1.size();i++)
  {
    cout<<list1[i]<<endl;
  }*/    //有误，必须使用迭代器进行访问
  list<int>::iterator itor=list1.begin();
  for(;itor!=list1.end();itor++)
  {
    cout<<*itor<<endl;//遍历list数组
  }
  return 0;
}
```

代码实例三（map用法）：

```
#include<iostream>
#include<stdlib.h>
#include<vector>
#include<list>
#include<map>
#include<string>
//通过使用标准模板库，学习map其用法
using namespace std;
int main()
{
  /*map<int,string> m;
  pair<int,string>p1(3,"hello");
  pair<int,string>p2(6,"world");
  m.insert(p1);
  m.insert(p2);
  //cout<<m[3]<<endl;
  //cout<<m[6]<<endl;//通过key找到相应的value
  map<int,string>::iterator itor=m.begin();//迭代器
  for(;itor!=m.end();itor++)
  {
    cout<<itor->first<<endl;
    cout<<itor->second<<endl;
    cout<<endl;
  }*/
  map<string,string> m;
  pair<string,string>p1("H","hello");
  pair<string,string>p2("W","world");
  pair<string,string>p3("B","beijing");
  m.insert(p1);
  m.insert(p2);
  m.insert(p3);
  //cout<<m["H"]<<endl;
```

```
  //cout<<m["B"]<<endl;//通过key找到相应的value
  map<string,string>::iterator itor=m.begin();//迭代器
  for(;itor!=m.end();itor++)
  {
    cout<<itor->first<<endl;
    cout<<itor->second<<endl;
    cout<<endl;
  }
    return 0;
}
```