**对map函数的一些整理**

**C++ map的基本操作和使用**

Map是c++的一个标准容器，她提供了很好一对一的关系，在一些程序中建立一个map可以起到事半功倍的效果，总结了一些map基本简单实用的操作！

**1、map简介**

map是一类关联式容器。它的特点是增加和删除节点对迭代器的影响很小，除了那个操作节点，对其他的节点都没有什么影响。对于迭代器来说，可以修改实值，而不能修改key。

**2、map的功能**

自动建立Key －value的对应。key 和value可以是任意你需要的类型。

根据key值快速查找记录，查找的复杂度基本是Log(N)，如果有1000个记录，最多查找10次，1,000,000个记录，最多查找20次。

快速插入Key - Value 记录。

快速删除记录 根据Key 修改value记录。

遍历所有记录。

**3、使用map**

使用map得包含map类所在的头文件

#include <map> **//注意，STL头文件没有扩展名.h**

map对象是模板类，需要关键字和存储对象两个模板参数：

std:map<int, string> personnel;

这样就定义了一个用int作为索引,并拥有相关联的指向string的指针.

为了使用方便，可以对模板类进行一下类型定义，

typedef map<int, CString> UDT\_MAP\_INT\_CSTRING;

UDT\_MAP\_INT\_CSTRING enumMap;

**4、在map中插入元素**

改变map中的条目非常简单，因为map类已经对[]操作符进行了重载

enumMap[1] = "One";

enumMap[2] = "Two";

.....

这样非常直观，但存在一个性能的问题。插入2时，先在enumMap中查找主键为2的项，没发现，然后将一个新的对象插入enumMap，键是2， 值是一个空字符串，插入完成后将字符串赋为"Two"; 该方法会将每个值都赋为缺省值，然后再赋为显示的值，如果元素是类对象，则开销比较大。我们可以用以下方法来避免开销：

enumMap.insert(map<int, CString> :: value\_type(2, "Two"))

**5、查找并获取**

map中的元素下标操作符给出了获得一个值的最简单方法：

CString tmp = enumMap[2];

但是,只有当map中有这个键的实例时才对，否则会自动插入一个实例，值为初始化值。 我们可以使用Find()和Count()方法来发现一个键是否存在。 查找map中是否包含某个关键字条目用find()方法，传入的参数是要查找的key，在这里需要提到的是begin()和end()两个成员，分别代表map对象中第一个条目和最后一个条目，这两个数据的类型是iterator.

int nFindKey = 2; //要查找的Key //

定义一个条目变量(实际是指针)

enumMap.find(nFindKey)返回类型也是iterator，可直接与begin()和end()比较

UDT\_MAP\_INT\_CSTRING::iterator it= **enumMap.find(nFindKey);**

if(it == enumMap.end()) {

    //没找到

}

else {

    //找到

}

通过map对象的方法获取的iterator数据类型是一个std::pair对象，包括两个数据 iterator->first 和 iterator->second 分别代表关键字和存储的数据

**6、从map中删除元素**

移除某个map中某个条目用erase()

该成员方法的定义如下

iterator erase(iterator it); //

通过一个条目对象删除

iterator erase(iterator first, iterator last); //

删除一个范围

size\_type erase(const Key& key); //

通过关键字删除clear()

就相当于

enumMap.erase(enumMap.begin(), enumMap.end());

**1. map最基本的构造函数；**

map<string , int >mapstring;

map<int ,string >mapint;

map<sring, char>mapstring;

map< char ,string>mapchar;

map<char ,int>mapchar;

map<int ,char >mapint;

**2. map添加数据；**

map<int ,string> maplive;

1.maplive.insert(pair<int,string>(102,"aclive"));

2.maplive.insert(map<int,string>::value\_type(321,"hai"));

3, maplive[112]="April";//map中最简单最常用的插入添加！

**3.map中元素的查找：**

find()函数返回一个迭代器指向键值为key的元素，如果没找到就返回指向map尾部的迭代器。

map<int ,string >::iterator l\_it;;

l\_it=maplive.find(112);

if(l\_it==maplive.end())

cout<<"we do not find 112"<<endl;

else cout<<"wo find 112"<<endl;

**4,map中元素的删除：**

如果删除112；

map<int ,string >::iterator l\_it;

l\_it=maplive.find(112);

if(l\_it==maplive.end())

    cout<<"we do not find 112"<<endl;

else

    maplive.erase(l\_it);

    //delete 112;

**5,map中swap的用法：**

Map中的swap不是一个容器中的元素交换，而是两个容器交换；

For example:

#include <map>

#include <iostream>

using namespace std;

int main( )

{

    map <int, int> m1, m2, m3;

    map <int, int>::iterator m1\_Iter;

    m1.insert ( pair <int, int> ( 1, 10 ) );

    m1.insert ( pair <int, int> ( 2, 20 ) );

    m1.insert ( pair <int, int> ( 3, 30 ) );

    m2.insert ( pair <int, int> ( 10, 100 ) );

    m2.insert ( pair <int, int> ( 20, 200 ) );

    m3.insert ( pair <int, int> ( 30, 300 ) );

    cout << "The original map m1 is:";

    for ( m1\_Iter = m1.begin( ); m1\_Iter != m1.end( ); m1\_Iter++ )

        cout << " " << m1\_Iter->second;

        cout << "." << endl;

       // This is the member function version of swap

      //m2 is said to be the argument map; m1 the target map

    m1.swap( m2 );

    cout << "After swapping with m2, map m1 is:";

    for ( m1\_Iter = m1.begin( ); m1\_Iter != m1.end( ); m1\_Iter++ )

         cout << " " << m1\_Iter -> second;

    cout << "." << endl;

    cout << "After swapping with m2, map m2 is:";

    for ( m1\_Iter = m2.begin( ); m1\_Iter != m2.end( ); m1\_Iter++ )

         cout << " " << m1\_Iter -> second;

    cout << "." << endl;

    // This is the specialized template version of swap

    swap( m1, m3 );

    cout << "After swapping with m3, map m1 is:";

    for ( m1\_Iter = m1.begin( ); m1\_Iter != m1.end( ); m1\_Iter++ )

         cout << " " << m1\_Iter -> second;

    cout  << "." << endl;

}

**6.map的sort问题：**

**Map中的元素是自动按key升序排序,所以不能对map用sort函数：**

 For example：

 #include <map>

 #include <iostream>

 using namespace std;

 int main( )

 {

        map <int, int> m1;

        map <int, int>::iterator m1\_Iter;

        m1.insert ( pair <int, int> ( 1, 20 ) );

        m1.insert ( pair <int, int> ( 4, 40 ) );

        m1.insert ( pair <int, int> ( 3, 60 ) );

        m1.insert ( pair <int, int> ( 2, 50 ) );

        m1.insert ( pair <int, int> ( 6, 40 ) );

        m1.insert ( pair <int, int> ( 7, 30 ) );

        cout << "The original map m1 is:"<<endl;

        for ( m1\_Iter = m1.begin( ); m1\_Iter != m1.end( ); m1\_Iter++ )

        cout << m1\_Iter->first<<" "<<m1\_Iter->second<<endl;

}

 The original map m1 is:

 1 20

 2 50

 3 60

 4 40

 6 40

 7 30

请按任意键继续. . .

**7,map的基本操作函数：**

 C++ Maps是一种关联式容器，包含“关键字/值”对

begin()             返回指向map头部的迭代器

clear(）            删除所有元素

count()            返回指定元素出现的次数

empty()           如果map为空则返回true

end()               返回指向map末尾的迭代器

equal\_range()   返回特殊条目的迭代器对

erase()             删除一个元素

find()               查找一个元素

get\_allocator()  返回map的配置器

insert()            插入元素

key\_comp()      返回比较元素

key的函数 lower\_bound()   返回键值>=给定元素的第一个位置

max\_size()       返回可以容纳的最大元素个数

rbegin()            返回一个指向map尾部的逆向迭代器

rend()              返回一个指向map头部的逆向迭代器

size()               返回map中元素的个数

swap()             交换两个map

upper\_bound() 返回键值>给定元素的第一个位置

value\_comp()   返回比较元素value的函数