[STL中map用法详解](http://blog.csdn.net/bat603/article/details/1456141)

<http://blog.csdn.net/bat603/article/details/1456141>

Map是STL的一个关联容器，它提供一对一（其中第一个可以称为关键字，每个关键字只能在map中出现一次，第二个可能称为该关键字的值）的数据处理能力，由于这个特性，它完成有可能在我们处理一对一数据的时候，在编程上提供快速通道。这里说下map内部数据的组织，map内部自建一颗红黑树(一种非严格意义上的平衡二叉树)，这颗树具有对数据自动排序的功能，所以在map内部所有的数据都是有序的，后边我们会见识到有序的好处。

下面举例说明什么是一对一的数据映射。比如一个班级中，每个学生的学号跟他的姓名就存在着一一映射的关系，这个模型用map可能轻易描述，很明显学号用int描述，姓名用字符串描述(本篇文章中不用char \*来描述字符串，而是采用STL中string来描述),下面给出map描述代码：

Map<int, string> mapStudent;

1.       map的构造函数

map共提供了6个构造函数，这块涉及到内存分配器这些东西，略过不表，在下面我们将接触到一些map的构造方法，这里要说下的就是，我们通常用如下方法构造一个map：

Map<int, string> mapStudent;

2.       数据的插入

在构造map容器后，我们就可以往里面插入数据了。这里讲三种插入数据的方法：

第一种：用insert函数插入pair数据，下面举例说明(以下代码虽然是随手写的，应该可以在VC和GCC下编译通过，大家可以运行下看什么效果，在VC下请加入这条语句，屏蔽4786警告  ＃pragma warning (disable:4786) )

#include <map>

#include <string>

#include <iostream>

Using namespace std;

Int main()

{

       Map<int, string> mapStudent;

       mapStudent.insert(pair<int, string>(1, “student\_one”));

       mapStudent.insert(pair<int, string>(2, “student\_two”));

       mapStudent.insert(pair<int, string>(3, “student\_three”));

       map<int, string>::iterator  iter;

       for(iter = mapStudent.begin(); iter != mapStudent.end(); iter++)

{

       Cout<<iter->first<<”   ”<<iter->second<<end;

}

}

第二种：用insert函数插入value\_type数据，下面举例说明

#include <map>

#include <string>

#include <iostream>

Using namespace std;

Int main()

{

       Map<int, string> mapStudent;

       mapStudent.insert(map<int, string>::value\_type (1, “student\_one”));

       mapStudent.insert(map<int, string>::value\_type (2, “student\_two”));

       mapStudent.insert(map<int, string>::value\_type (3, “student\_three”));

       map<int, string>::iterator  iter;

       for(iter = mapStudent.begin(); iter != mapStudent.end(); iter++)

{

       Cout<<iter->first<<”   ”<<iter->second<<end;

}

}

第三种：用数组方式插入数据，下面举例说明

#include <map>

#include <string>

#include <iostream>

Using namespace std;

Int main()

{

       Map<int, string> mapStudent;

       mapStudent[1] =  “student\_one”;

       mapStudent[2] =  “student\_two”;

       mapStudent[3] =  “student\_three”;

       map<int, string>::iterator  iter;

       for(iter = mapStudent.begin(); iter != mapStudent.end(); iter++)

{

       Cout<<iter->first<<”   ”<<iter->second<<end;

}

}

以上三种用法，虽然都可以实现数据的插入，但是它们是有区别的，当然了第一种和第二种在效果上是完成一样的，用insert函数插入数据，在数据的插入上涉及到集合的唯一性这个概念，即当map中有这个关键字时，insert操作是插入数据不了的，但是用数组方式就不同了，它可以覆盖以前该关键字对应的值，用程序说明

mapStudent.insert(map<int, string>::value\_type (1, “student\_one”));

mapStudent.insert(map<int, string>::value\_type (1, “student\_two”));

上面这两条语句执行后，map中1这个关键字对应的值是“student\_one”，第二条语句并没有生效，那么这就涉及到我们怎么知道insert语句是否插入成功的问题了，可以用pair来获得是否插入成功，程序如下

Pair<map<int, string>::iterator, bool> Insert\_Pair;

Insert\_Pair = mapStudent.insert(map<int, string>::value\_type (1, “student\_one”));

我们通过pair的第二个变量来知道是否插入成功，它的第一个变量返回的是一个map的迭代器，如果插入成功的话Insert\_Pair.second应该是true的，否则为false。

下面给出完成代码，演示插入成功与否问题

#include <map>

#include <string>

#include <iostream>

Using namespace std;

Int main()

{

       Map<int, string> mapStudent;

Pair<map<int, string>::iterator, bool> Insert\_Pair;

       Insert\_Pair ＝ mapStudent.insert(pair<int, string>(1, “student\_one”));

       If(Insert\_Pair.second == true)

       {

              Cout<<”Insert Successfully”<<endl;

       }

       Else

       {

              Cout<<”Insert Failure”<<endl;

       }

       Insert\_Pair ＝ mapStudent.insert(pair<int, string>(1, “student\_two”));

       If(Insert\_Pair.second == true)

       {

              Cout<<”Insert Successfully”<<endl;

       }

       Else

       {

              Cout<<”Insert Failure”<<endl;

       }

       map<int, string>::iterator  iter;

       for(iter = mapStudent.begin(); iter != mapStudent.end(); iter++)

{

       Cout<<iter->first<<”   ”<<iter->second<<end;

}

}

大家可以用如下程序，看下用数组插入在数据覆盖上的效果

#include <map>

#include <string>

#include <iostream>

Using namespace std;

Int main()

{

       Map<int, string> mapStudent;

       mapStudent[1] =  “student\_one”;

       mapStudent[1] =  “student\_two”;

       mapStudent[2] =  “student\_three”;

       map<int, string>::iterator  iter;

       for(iter = mapStudent.begin(); iter != mapStudent.end(); iter++)

{

       Cout<<iter->first<<”   ”<<iter->second<<end;

}

}

3.       map的大小

在往map里面插入了数据，我们怎么知道当前已经插入了多少数据呢，可以用size函数，用法如下：

Int nSize = mapStudent.size();

4.       数据的遍历

这里也提供三种方法，对map进行遍历

第一种：应用前向迭代器，上面举例程序中到处都是了，略过不表

第二种：应用反相迭代器，下面举例说明，要体会效果，请自个动手运行程序

#include <map>

#include <string>

#include <iostream>

Using namespace std;

Int main()

{

       Map<int, string> mapStudent;

       mapStudent.insert(pair<int, string>(1, “student\_one”));

       mapStudent.insert(pair<int, string>(2, “student\_two”));

       mapStudent.insert(pair<int, string>(3, “student\_three”));

       map<int, string>::reverse\_iterator  iter;

       for(iter = mapStudent.rbegin(); iter != mapStudent.rend(); iter++)

{

       Cout<<iter->first<<”   ”<<iter->second<<end;

}

}

第三种：用数组方式，程序说明如下

#include <map>

#include <string>

#include <iostream>

Using namespace std;

Int main()

{

       Map<int, string> mapStudent;

       mapStudent.insert(pair<int, string>(1, “student\_one”));

       mapStudent.insert(pair<int, string>(2, “student\_two”));

       mapStudent.insert(pair<int, string>(3, “student\_three”));

       int nSize = mapStudent.size()

//此处有误，应该是 for(int nIndex = 1; nIndex <= nSize; nIndex++)

//by rainfish

       for(int nIndex = 0; nIndex < nSize; nIndex++)

{

       Cout<<mapStudent[nIndex]<<end;

}

}

5.       数据的查找（包括判定这个关键字是否在map中出现）

在这里我们将体会，map在数据插入时保证有序的好处。

要判定一个数据（关键字）是否在map中出现的方法比较多，这里标题虽然是数据的查找，在这里将穿插着大量的map基本用法。

这里给出三种数据查找方法

第一种：用count函数来判定关键字是否出现，其缺点是无法定位数据出现位置,由于map的特性，一对一的映射关系，就决定了count函数的返回值只有两个，要么是0，要么是1，出现的情况，当然是返回1了

第二种：用find函数来定位数据出现位置，它返回的一个迭代器，当数据出现时，它返回数据所在位置的迭代器，如果map中没有要查找的数据，它返回的迭代器等于end函数返回的迭代器，程序说明

#include <map>

#include <string>

#include <iostream>

Using namespace std;

Int main()

{

       Map<int, string> mapStudent;

       mapStudent.insert(pair<int, string>(1, “student\_one”));

       mapStudent.insert(pair<int, string>(2, “student\_two”));

       mapStudent.insert(pair<int, string>(3, “student\_three”));

       map<int, string>::iterator iter;

       iter = mapStudent.find(1);

if(iter != mapStudent.end())

{

       Cout<<”Find, the value is ”<<iter->second<<endl;

}

Else

{

       Cout<<”Do not Find”<<endl;

}

}

第三种：这个方法用来判定数据是否出现，是显得笨了点，但是，我打算在这里讲解

Lower\_bound函数用法，这个函数用来返回要查找关键字的下界(是一个迭代器)

Upper\_bound函数用法，这个函数用来返回要查找关键字的上界(是一个迭代器)

例如：map中已经插入了1，2，3，4的话，如果lower\_bound(2)的话，返回的2，而upper-bound（2）的话，返回的就是3

Equal\_range函数返回一个pair，pair里面第一个变量是Lower\_bound返回的迭代器，pair里面第二个迭代器是Upper\_bound返回的迭代器，如果这两个迭代器相等的话，则说明map中不出现这个关键字，程序说明

#include <map>

#include <string>

#include <iostream>

Using namespace std;

Int main()

{

       Map<int, string> mapStudent;

       mapStudent[1] =  “student\_one”;

       mapStudent[3] =  “student\_three”;

       mapStudent[5] =  “student\_five”;

       map<int, string>::iterator  iter;

iter = mapStudent.lower\_bound(2);

{

       //返回的是下界3的迭代器

       Cout<<iter->second<<endl;

}

iter = mapStudent.lower\_bound(3);

{

       //返回的是下界3的迭代器

       Cout<<iter->second<<endl;

}

iter = mapStudent.upper\_bound(2);

{

       //返回的是上界3的迭代器

       Cout<<iter->second<<endl;

}

iter = mapStudent.upper\_bound(3);

{

       //返回的是上界5的迭代器

       Cout<<iter->second<<endl;

}

Pair<map<int, string>::iterator, map<int, string>::iterator> mapPair;

mapPair = mapStudent.equal\_range(2);

if(mapPair.first == mapPair.second)  
       {

       cout<<”Do not Find”<<endl;

}

Else

{

Cout<<”Find”<<endl;  
}

mapPair = mapStudent.equal\_range(3);

if(mapPair.first == mapPair.second)  
       {

       cout<<”Do not Find”<<endl;

}

Else

{

Cout<<”Find”<<endl;  
}

}

6.       数据的清空与判空

清空map中的数据可以用clear()函数，判定map中是否有数据可以用empty()函数，它返回true则说明是空map

7.       数据的删除

这里要用到erase函数，它有三个重载了的函数，下面在例子中详细说明它们的用法

#include <map>

#include <string>

#include <iostream>

Using namespace std;

Int main()

{

       Map<int, string> mapStudent;

       mapStudent.insert(pair<int, string>(1, “student\_one”));

       mapStudent.insert(pair<int, string>(2, “student\_two”));

       mapStudent.insert(pair<int, string>(3, “student\_three”));

//如果你要演示输出效果，请选择以下的一种，你看到的效果会比较好

       //如果要删除1,用迭代器删除

       map<int, string>::iterator iter;

       iter = mapStudent.find(1);

       mapStudent.erase(iter);

       //如果要删除1，用关键字删除

       Int n = mapStudent.erase(1);//如果删除了会返回1，否则返回0

       //用迭代器，成片的删除

       //一下代码把整个map清空

       mapStudent.earse(mapStudent.begin(), mapStudent.end());

       //成片删除要注意的是，也是STL的特性，删除区间是一个前闭后开的集合

       //自个加上遍历代码，打印输出吧

}

8.       其他一些函数用法

这里有swap,key\_comp,value\_comp,get\_allocator等函数，感觉到这些函数在编程用的不是很多，略过不表，有兴趣的话可以自个研究

9.       排序

这里要讲的是一点比较高深的用法了,排序问题，STL中默认是采用小于号来排序的，以上代码在排序上是不存在任何问题的，因为上面的关键字是int型，它本身支持小于号运算，在一些特殊情况，比如关键字是一个结构体，涉及到排序就会出现问题，因为它没有小于号操作，insert等函数在编译的时候过不去，下面给出两个方法解决这个问题

第一种：小于号重载，程序举例

#include <map>

#include <string>

Using namespace std;

Typedef struct tagStudentInfo

{

       Int      nID;

       String   strName;

}StudentInfo, \*PStudentInfo;  //学生信息

Int main()

{

    int nSize;

       //用学生信息映射分数

       map<StudentInfo, int>mapStudent;

    map<StudentInfo, int>::iterator iter;

       StudentInfo studentInfo;

       studentInfo.nID = 1;

       studentInfo.strName = “student\_one”;

       mapStudent.insert(pair<StudentInfo, int>(studentInfo, 90));

       studentInfo.nID = 2;

       studentInfo.strName = “student\_two”;

mapStudent.insert(pair<StudentInfo, int>(studentInfo, 80));

for (iter=mapStudent.begin(); iter!=mapStudent.end(); iter++)

    cout<<iter->first.nID<<endl<<iter->first.strName<<endl<<iter->second<<endl;

}

以上程序是无法编译通过的，只要重载小于号，就OK了，如下：

Typedef struct tagStudentInfo

{

       Int      nID;

       String   strName;

       Bool operator < (tagStudentInfo const& \_A) const

       {

              //这个函数指定排序策略，按nID排序，如果nID相等的话，按strName排序

              If(nID < \_A.nID)  return true;

              If(nID == \_A.nID) return strName.compare(\_A.strName) < 0;

              Return false;

       }

}StudentInfo, \*PStudentInfo;  //学生信息

第二种：仿函数的应用，这个时候结构体中没有直接的小于号重载，程序说明

#include <map>

#include <string>

Using namespace std;

Typedef struct tagStudentInfo

{

       Int      nID;

       String   strName;

}StudentInfo, \*PStudentInfo;  //学生信息

Classs sort

{

       Public:

       Bool operator() (StudentInfo const &\_A, StudentInfo const &\_B) const

       {

              If(\_A.nID < \_B.nID) return true;

              If(\_A.nID == \_B.nID) return \_A.strName.compare(\_B.strName) < 0;

              Return false;

       }

};

Int main()

{

       //用学生信息映射分数

       Map<StudentInfo, int, sort>mapStudent;

       StudentInfo studentInfo;

       studentInfo.nID = 1;

       studentInfo.strName = “student\_one”;

       mapStudent.insert(pair<StudentInfo, int>(studentInfo, 90));

       studentInfo.nID = 2;

       studentInfo.strName = “student\_two”;

mapStudent.insert(pair<StudentInfo, int>(studentInfo, 80));

}

10.   另外

由于STL是一个统一的整体，map的很多用法都和STL中其它的东西结合在一起，比如在排序上，这里默认用的是小于号，即less<>，如果要从大到小排序呢，这里涉及到的东西很多，在此无法一一加以说明。

还要说明的是，map中由于它内部有序，由红黑树保证，因此很多函数执行的时间复杂度都是log2N的，如果用map函数可以实现的功能，而STL  Algorithm也可以完成该功能，建议用map自带函数，效率高一些。

下面说下，map在空间上的特性，否则，估计你用起来会有时候表现的比较郁闷，由于map的每个数据对应红黑树上的一个节点，这个节点在不保存你的数据时，是占用16个字节的，一个父节点指针，左右孩子指针，还有一个枚举值（标示红黑的，相当于平衡二叉树中的平衡因子），我想大家应该知道，这些地方很费内存了吧，不说了……