<https://www.cnblogs.com/fnlingnzb-learner/p/5833051.html>

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。

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、       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;

5、     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) )

**[cpp]** [view plain](http://blog.csdn.net/shawn_hou/article/details/38035577) [copy](http://blog.csdn.net/shawn_hou/article/details/38035577)

1. //数据的插入--第一种：用insert函数插入pair数据
2. #include <map>
4. #include <string>
6. #include <iostream>
8. **using namespace std;**
10. int main()
12. {
14. map<int, string> mapStudent;
16. mapStudent.insert(pair<int, string>(1, "student\_one"));
18. mapStudent.insert(pair<int, string>(2, "student\_two"));
20. mapStudent.insert(pair<int, string>(3, "student\_three"));
22. map<int, string>::iterator iter;
24. **for(iter = mapStudent.begin(); iter != mapStudent.end(); iter++)**
26. cout<<iter->first<<' '<<iter->second<<endl;
28. }

第二种：用insert函数插入value\_type数据，下面举例说明

**[cpp]** [view plain](http://blog.csdn.net/shawn_hou/article/details/38035577) [copy](http://blog.csdn.net/shawn_hou/article/details/38035577)

1. //第二种：用insert函数插入value\_type数据，下面举例说明
3. #include <map>
5. #include <string>
7. #include <iostream>
9. **using namespace std;**
11. int main()
13. {
15. map<int, string> mapStudent;
17. mapStudent.insert(map<int, string>::value\_type (1, "student\_one"));
19. mapStudent.insert(map<int, string>::value\_type (2, "student\_two"));
21. mapStudent.insert(map<int, string>::value\_type (3, "student\_three"));
23. map<int, string>::iterator iter;
25. **for(iter = mapStudent.begin(); iter != mapStudent.end(); iter++)**
27. cout<<iter->first<<' '<<iter->second<<endl;
29. }

第三种：用数组方式插入数据，下面举例说明

**[cpp]** [view plain](http://blog.csdn.net/shawn_hou/article/details/38035577) [copy](http://blog.csdn.net/shawn_hou/article/details/38035577)

1. //第三种：用数组方式插入数据，下面举例说明
3. #include <map>
5. #include <string>
7. #include <iostream>
9. **using namespace std;**
11. int main()
13. {
15. map<int, string> mapStudent;
17. mapStudent[1] = "student\_one";
19. mapStudent[2] = "student\_two";
21. mapStudent[3] = "student\_three";
23. map<int, string>::iterator iter;
25. **for(iter = mapStudent.begin(); iter != mapStudent.end(); iter++)**
27. cout<<iter->first<<' '<<iter->second<<endl;
29. }

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

**[cpp]** [view plain](http://blog.csdn.net/shawn_hou/article/details/38035577) [copy](http://blog.csdn.net/shawn_hou/article/details/38035577)

1. //验证插入函数的作用效果
2. #include <map>
4. #include <string>
6. #include <iostream>
8. **using namespace std;**
10. int main()
12. {
14. map<int, string> mapStudent;
16. pair<map<int, string>::iterator, bool> Insert\_Pair;
18. Insert\_Pair = mapStudent.insert(pair<int, string>(1, "student\_one"));
20. **if(Insert\_Pair.second == true)**
22. cout<<"Insert Successfully"<<endl;
24. **else**
26. cout<<"Insert Failure"<<endl;
28. Insert\_Pair = mapStudent.insert(pair<int, string>(1, "student\_two"));
30. **if(Insert\_Pair.second == true)**
32. cout<<"Insert Successfully"<<endl;
34. **else**
36. cout<<"Insert Failure"<<endl;
38. map<int, string>::iterator iter;
40. **for(iter = mapStudent.begin(); iter != mapStudent.end(); iter++)**
42. cout<<iter->first<<' '<<iter->second<<endl;
44. }

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

**[cpp]** [view plain](http://blog.csdn.net/shawn_hou/article/details/38035577) [copy](http://blog.csdn.net/shawn_hou/article/details/38035577)

1. //验证数组形式插入数据的效果
3. #include <map>
5. #include <string>
7. #include <iostream>
9. **using namespace std;**
11. int main()
13. {
15. map<int, string> mapStudent;
17. mapStudent[1] = "student\_one";
19. mapStudent[1] = "student\_two";
21. mapStudent[2] = "student\_three";
23. map<int, string>::iterator iter;
25. **for(iter = mapStudent.begin(); iter != mapStudent.end(); iter++)**
27. cout<<iter->first<<' '<<iter->second<<endl;
28. }

6、      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();

7、     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

**[cpp]** [view plain](http://blog.csdn.net/shawn_hou/article/details/38035577) [copy](http://blog.csdn.net/shawn_hou/article/details/38035577)

1. //第二种，利用反向迭代器
3. #include <map>
5. #include <string>
7. #include <iostream>
9. **using namespace std;**
11. int main()
13. {
15. map<int, string> mapStudent;
17. mapStudent.insert(pair<int, string>(1, "student\_one"));
19. mapStudent.insert(pair<int, string>(2, "student\_two"));
21. mapStudent.insert(pair<int, string>(3, "student\_three"));
23. map<int, string>::reverse\_iterator iter;
25. **for(iter = mapStudent.rbegin(); iter != mapStudent.rend(); iter++)**
27. cout<<iter->first<<"  "<<iter->second<<endl;
29. }

第三种，用数组的形式，程序说明如下：

**[cpp]** [view plain](http://blog.csdn.net/shawn_hou/article/details/38035577) [copy](http://blog.csdn.net/shawn_hou/article/details/38035577)

1. //第三种：用数组方式，程序说明如下
3. #include <map>
5. #include <string>
7. #include <iostream>
9. **using namespace std;**
11. int main()
13. {
15. map<int, string> mapStudent;
17. mapStudent.insert(pair<int, string>(1, "student\_one"));
19. mapStudent.insert(pair<int, string>(2, "student\_two"));
21. mapStudent.insert(pair<int, string>(3, "student\_three"));
23. int nSize = mapStudent.size();
25. //此处应注意，应该是 for(int nindex = 1; nindex <= nSize; nindex++)
26. //而不是 for(int nindex = 0; nindex < nSize; nindex++)
28. **for(int nindex = 1; nindex <= nSize; nindex++)**
30. cout<<mapStudent[nindex]<<endl;
32. }

8、    查找并获取map中的元素（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。

**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.

程序说明

**[cpp]** [view plain](http://blog.csdn.net/shawn_hou/article/details/38035577) [copy](http://blog.csdn.net/shawn_hou/article/details/38035577)

1. #include <map>
3. #include <string>
5. #include <iostream>
7. **using namespace std;**
9. int main()
11. {
13. map<int, string> mapStudent;
15. mapStudent.insert(pair<int, string>(1, "student\_one"));
17. mapStudent.insert(pair<int, string>(2, "student\_two"));
19. mapStudent.insert(pair<int, string>(3, "student\_three"));
21. map<int, string>::iterator iter;
23. iter = mapStudent.find(1);
25. **if(iter != mapStudent.end())**
27. cout<<"Find, the value is "<<iter->second<<endl;
29. **else**
31. cout<<"Do not Find"<<endl;
33. **return 0;**
34. }

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。

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

**[cpp]** [view plain](http://blog.csdn.net/shawn_hou/article/details/38035577) [copy](http://blog.csdn.net/shawn_hou/article/details/38035577)

1. #include <map>
3. #include <string>
5. #include <iostream>
7. **using namespace std;**
9. int main()
11. {
13. map<int, string> mapStudent;
15. mapStudent[1] = "student\_one";
17. mapStudent[3] = "student\_three";
19. mapStudent[5] = "student\_five";
21. map<int, string>::iterator iter;
23. iter = mapStudent.lower\_bound(1);
25. //返回的是下界1的迭代器
27. cout<<iter->second<<endl;
29. iter = mapStudent.lower\_bound(2);
31. //返回的是下界3的迭代器
33. cout<<iter->second<<endl;
35. iter = mapStudent.lower\_bound(3);
37. //返回的是下界3的迭代器
39. cout<<iter->second<<endl;
41. iter = mapStudent.upper\_bound(2);
43. //返回的是上界3的迭代器
45. cout<<iter->second<<endl;
47. iter = mapStudent.upper\_bound(3);
49. //返回的是上界5的迭代器
51. cout<<iter->second<<endl;
53. pair<map<int, string>::iterator, map<int, string>::iterator> mappair;
55. mappair = mapStudent.equal\_range(2);
57. **if(mappair.first == mappair.second)**
59. cout<<"Do not Find"<<endl;
61. **else**
63. cout<<"Find"<<endl;
65. mappair = mapStudent.equal\_range(3);
67. **if(mappair.first == mappair.second)**
69. cout<<"Do not Find"<<endl;
71. **else**
73. cout<<"Find"<<endl;
75. **return 0;**
76. }

9、    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());

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

**[cpp]** [view plain](http://blog.csdn.net/shawn_hou/article/details/38035577) [copy](http://blog.csdn.net/shawn_hou/article/details/38035577)

1. #include <map>
3. #include <string>
5. #include <iostream>
7. **using namespace std;**
9. int main()
11. {
13. map<int, string> mapStudent;
15. mapStudent.insert(pair<int, string>(1, "student\_one"));
17. mapStudent.insert(pair<int, string>(2, "student\_two"));
19. mapStudent.insert(pair<int, string>(3, "student\_three"));
21. //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
23. //如果要删除1,用迭代器删除
25. map<int, string>::iterator iter;
27. iter = mapStudent.find(1);
29. mapStudent.erase(iter);
31. //如果要删除1，用关键字删除
33. int n = mapStudent.erase(1);//如果删除了会返回1，否则返回0
35. //用迭代器，成片的删除
37. //一下代码把整个map清空
39. mapStudent.erase( mapStudent.begin(), mapStudent.end() );
41. //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
43. //自个加上遍历代码，打印输出吧
45. }

10、    map中的swap用法

map中的swap不是一个容器中的元素交换，而是两个容器所有元素的交换。

11、     排序 ·  map中的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；

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。

**[cpp]** [view plain](http://blog.csdn.net/shawn_hou/article/details/38035577) [copy](http://blog.csdn.net/shawn_hou/article/details/38035577)

1. #include <iostream>
2. #include <string>
3. #include <map>
4. **using namespace std;**
6. **typedef struct tagStudentinfo**
8. {
10. int      niD;
12. string   strName;
14. bool operator < (tagStudentinfo **const& \_A) const**
16. {     //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
18. **if(niD < \_A.niD) return true;**
20. **if(niD == \_A.niD)**
22. **return strName.compare(\_A.strName) < 0;**
24. **return false;**
26. }
28. }Studentinfo, \*PStudentinfo; //学生信息
30. int main()
32. {
34. int nSize;   //用学生信息映射分数
36. map<Studentinfo, int>mapStudent;
38. map<Studentinfo, int>::iterator iter;
40. Studentinfo studentinfo;
42. studentinfo.niD = 1;
44. studentinfo.strName = "student\_one";
46. mapStudent.insert(pair<Studentinfo, int>(studentinfo, 90));
48. studentinfo.niD = 2;
50. studentinfo.strName = "student\_two";
52. mapStudent.insert(pair<Studentinfo, int>(studentinfo, 80));
54. **for (iter=mapStudent.begin(); iter!=mapStudent.end(); iter++)**
56. cout<<iter->first.niD<<' '<<iter->first.strName<<' '<<iter->second<<endl;
58. **return 0;**
59. }

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

**[cpp]** [view plain](http://blog.csdn.net/shawn_hou/article/details/38035577) [copy](http://blog.csdn.net/shawn_hou/article/details/38035577)

1. //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
3. #include <iostream>
5. #include <map>
7. #include <string>
9. **using namespace std;**
11. **typedef struct tagStudentinfo**
13. {
15. int      niD;
17. string   strName;
19. }Studentinfo, \*PStudentinfo; //学生信息
21. **class sort**
23. {
25. **public:**
27. bool operator() (Studentinfo **const &\_A, Studentinfo const &\_B) const**
29. {
31. **if(\_A.niD < \_B.niD)**
33. **return true;**
35. **if(\_A.niD == \_B.niD)**
37. **return \_A.strName.compare(\_B.strName) < 0;**
39. **return false;**
41. }
42. };
44. int main()
46. {   //用学生信息映射分数
48. map<Studentinfo, int, sort>mapStudent;
50. map<Studentinfo, int>::iterator iter;
52. Studentinfo studentinfo;
54. studentinfo.niD = 1;
56. studentinfo.strName = "student\_one";
58. mapStudent.insert(pair<Studentinfo, int>(studentinfo, 90));
60. studentinfo.niD = 2;
62. studentinfo.strName = "student\_two";
64. mapStudent.insert(pair<Studentinfo, int>(studentinfo, 80));
66. **for (iter=mapStudent.begin(); iter!=mapStudent.end(); iter++)**
68. cout<<iter->first.niD<<' '<<iter->first.strName<<' '<<iter->second<<endl;
69. }

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……

12、

      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