## STL

### set

**begin()** ,返回set容器的第一个元素

**end()** ,返回set容器的最后一个元素

**clear()** ,删除set容器中的所有的元素

**empty()**,判断set容器是否为空

**max\_size()** ,返回set容器可能包含的元素最大个数

**size()** ,返回当前set容器中的元素个数

**rbegin** ,返回的值和end()相同

**rend()**,返回的值和rbegin()相同

**count()** 用来查找set中某个某个键值出现的次数。

**equal\_range()** ，返回一对定位器，分别表示第一个大于或等于给定关键值的元素和 第一个大于给定关键值的元素，这个返回值是一个pair类型，如果这一对定位器中哪个返回失败，就会等于end()的值。

**erase(iterator)** ,删除定位器iterator指向的值

**erase(first,second)**,删除定位器first和second之间的值

**erase(key\_value)**,删除键值key\_value的值

**find()** ，返回给定值值得定位器，如果没找到则返回end()。

**insert(key\_value);** 将key*value插入到set中 ，返回值是pair<set::iterator,bool>，bool标志着插入是否成功，而iterator代表插入的位置，若key*value已经在set中，则iterator表示的key\_value在set中的位置。

**inset(first,second);**将定位器first到second之间的元素插入到set中，返回值是void.

**lower*bound(key*value)** ，返回第一个大于等于key\_value的定位器

**upper*bound(key*value)，**返回最后一个大于等于key\_value的定位器

### map

**插入操作**

使用[ ]进行单个插入

map<int, string> ID\_Name;  
  
// 如果已经存在键值2015，则会作赋值修改操作，如果没有则插入  
ID\_Name[2015] = "Tom";1234

使用insert进行单个和多个插入 (insert共有4个重载函数：

// 插入单个键值对，并返回插入位置和成功标志，插入位置已经存在值时，插入失败  
pair<iterator,bool> insert (const value\_type& val);  
  
//在指定位置插入，在不同位置插入效率是不一样的，因为涉及到重排  
iterator insert (const\_iterator position, const value\_type& val);  
  
// 插入多个  
void insert (InputIterator first, InputIterator last);  
  
//c++11开始支持，使用列表插入多个   
void insert (initializer\_list<value\_type> il);

**取值**

Map中元素取值主要有at和[ ]两种操作，at会作下标检查，而[]不会。

map<int, string> ID\_Name;  
  
//ID\_Name中没有关键字2016，使用[]取值会导致插入  
//因此，下面语句不会报错，但打印结果为空  
cout<<ID\_Name[2016].c\_str()<<endl;  
  
//使用at会进行关键字检查，因此下面语句会报错  
ID\_Name.at(2016) = "Bob";

**容量查询**

// 查询map是否为空  
bool empty();  
  
// 查询map中键值对的数量  
size\_t size();  
  
// 查询map所能包含的最大键值对数量，和系统和应用库有关。  
// 此外，这并不意味着用户一定可以存这么多，很可能还没达到就已经开辟内存失败了  
size\_t max\_size();  
  
// 查询关键字为key的元素的个数，在map里结果非0即1  
size\_t count( const Key& key ) const; //

**迭代器**

共有八个获取迭代器的函数：**begin, end, rbegin,rend** 以及对应的 **cbegin, cend, crbegin,crend**。

二者的区别在于，后者一定返回 const*iterator，而前者则根据map的类型返回iterator 或者 const*iterator。const情况下，不允许对值进行修改。如下面代码所示：

map<int,int>::iterator it;  
map<int,int> mmap;  
const map<int,int> const\_mmap;  
  
it = mmap.begin(); //iterator  
mmap.cbegin(); //const\_iterator  
  
const\_mmap.begin(); //const\_iterator  
const\_mmap.cbegin(); //const\_iterator123456789

返回的迭代器可以进行加减操作，此外，如果map为空，则 begin = end。

**删除**

// 删除迭代器指向位置的键值对，并返回一个指向下一元素的迭代器  
iterator erase( iterator pos )  
  
// 删除一定范围内的元素，并返回一个指向下一元素的迭代器  
iterator erase( const\_iterator first, const\_iterator last );  
  
// 根据Key来进行删除， 返回删除的元素数量，在map里结果非0即1  
size\_t erase( const key\_type& key );  
  
// 清空map，清空后的size为0  
void clear();

**交换**

// 就是两个map的内容互换  
void swap( map& other );

**顺序比较**

// 比较两个关键字在map中位置的先后  
key\_compare key\_comp() const;

**查找**

// 关键字查询，找到则返回指向该关键字的迭代器，否则返回指向end的迭代器  
// 根据map的类型，返回的迭代器为 iterator 或者 const\_iterator  
iterator find (const key\_type& k);  
const\_iterator find (const key\_type& k) const;

**操作符**

operator: == != < <= > >=  
**注意** 对于==运算符, 只有键值对以及顺序完全相等才算成立。

### unordered\_map

**查找元素是否存在**

若有unordered\_map <int, int> mp;查找x是否在map中  
 方法1: 若存在 mp.find(x)!=mp.end()  
 方法2: 若存在 mp.count(x)!=0123

**插入数据**

mp.insert(Map::value\_type(1,"Raoul"));1

**遍历map**

unordered\_map<key,T>::iterator it;  
 (\*it).first; //the key value  
 (\*it).second //the mapped value  
 for(unordered\_map<key,T>::iterator iter=mp.begin();iter!=mp.end();iter++)  
 cout<<"key value is"<<iter->first<<" the mapped value is "<< iter->second;  
  
 //也可以这样  
 for(auto& v : mp)  
 print v.first and v.second