关联容器map/set

有序	
map	关联数组，保存关键字——值对
set	只保存关键字的容器
multimap	允许重复的map
multiset	允许重复的set
无序	
unordered_map	用hash函数组织的map
unordered_set	用hash函数组织的set
unordered_multimap	
unordered_multiset	

使用关键字类型的比较函数（用自己的比较函数）

```
bool myCompare(const Nice& a, const Nice& b)
{
  return a.age <= b.age;
}
set<Nice,decltype(myCompare)*> testNice(myCompare);
```

map成员类型为pair

```
pair<string,int> a;  // 初始化一个pair 类型 a
pair<string,string> author{"James","nice"};
pair<T1,T2> p(v1,v2);
pair<T1,T2> p = {v1,v2};
make_pair(v1,v2);   // 通过推断返回一个pair
p.first
p.second
```

关联容器的操作

关联容器额外类型别名	
key_type	此容器类型的关键字类型
mapped_type	适用于map
value_type	通用型

```
set<string>::value_type v1; // v1是一个string
set<string>::key_type v2; //v2 是一个string
map<string,int>::value_type v3; v3 是个 pair<const string,int>
map<string,int>::key_type v4; v4 是个 string
map<string,int>::mapped_type v5; v5 是个 int
```

关联容器的迭代器

map中的pair中的first是const类型，set中所有元素都是const类型，只能读，不能改。

遍历迭代器用begin和end。

添加元素

```
vector<int> vect = {1,2,3,4};
set<int> set1;
set1.insert(vect.begin(),vect.end());
set1.insert(5);
set1.insert({6});
set1.insert(set1.begin(),7);     // 不知道有什么用，后期再会
set1.emplace(8);  // 初始化加入那种
set1.emplace(set1.begin(),9);    //报错，不知道什么原因
// map也一样，但单个插入有返回值
map<string,int> map1;
auto ret = map1.insert({"nice",1}); // 返回一个pair，其first指向刚插入的那个pair，已存在，则指向已存在那个，其second的值为是否插入成功。所以可以通过ret访问刚插入的那个值或者已存在那个值。对于multimap直接返回一个迭代器，无需返回bool值。
```

删除元素

```
c.erase(k); // 删除关键字为k的元素，返回删除元素的个数
c.erase(p); // 删除指针p指向的元素，p指向真实元素，则返回一个指向p之后的迭代器，反之，返回c.end();
c.erase(b,e);    // 删除迭代器b和e范围之内的元素，返回e。
```

map的下标操作

```
c[k]; // 返回关键字为k的元素：如果k不存在，则添加一个关键字为k的元素。
c.at(k); // 只是访问，并返回存在值，不存在则跑出out_of_range异常

c.find(k);  //返回一个迭代器，指向第一个为k的元素
c.count(k); // 返回关键字等于k的元素个数

c.lower_bound(k);    // 返回一个迭代器，且其值 >= k
c.upper_bound(k);    //                       >k
c.equal_range(k);    // 返回一个迭代器pair，表示关键字等于k的元素范围，若k不存在，pair两个成员都等于c.end();
```

无序容器的一些操作

```
// 桶接口
c.bucket_count()    // 正在使用的桶的数目
c.max_bucket_count()    // 能容纳最大桶数
c.bucket_size(n)    // 第n个桶有几个元素
c.bucket(k)     //关键字为k的元素在那个桶中
// 桶迭代
local_itearor ..... // 跟其他迭代一样
// 哈希策略
c.load_factor()   // 每个桶平均元素数量，返回float
c.rehash(n)     // 重新存储，使得bucket_count >= n
c.reserve(n)    // 重新存储，使得c可以存储n个元素，且不必rehash。
```