|  |  |
| --- | --- |
| 1. 访问元素 2. 对map使用find代替下标操作 3. 在multimap或multiset中查找元素 4. 一种不同的，面向迭代器的解决方法 5. Equal\_range函数 | set1.count(4)  set1.find(4)  map1.find(4) == map1.end()  lower\_bound upper\_bound  queal\_range |
| 1. 访问元素 2. 有多种查找一个指定元素 3. Find与count差不多，但在可重复的容器中，不想计数 就用find 4. 对map使用find代替下标操作 5. 下标会有额外工作，所以用find，找到就找到指向关键字为k的迭代器，没有就为end尾后迭代器 6. 在multimap或multiset中查找元素 7. 允许重复关键字中，查找所有关键字所对应的元素。 8. 因为可重复容器的元素会相邻存储，所以可以find与count配合使用，count控制循环，find找到的迭代器前进多少 9. 一种不同的，面向迭代器的解决方法 10. lower\_bound upper\_bound得到迭代器范围 11. 若给定关键字是容器的关键字最大的，则upper\_bound返回尾后迭代器，若关键字不存在，且大于容器中任何关键字，则lower\_bound返回的也是尾后迭代器   Note:若关键字不在容器中，low\_bound返回关键字的第一个安全插入点-不影响容器中元素顺序的插入位置。  Note：lower\_bound与upper\_bound返回相同的迭代器，则给定的关键字不在容器中   1. Equal\_range函数 2. 接受一个关键字，返回一个pair，first为第一个与关键字匹配的元素，第二个迭代器指向最后一个匹配元素之后的位置。未找到，两个迭代器都指向关键字可以插入的位置 3. Pair<map<string,size\_t>,map<string,size\_t>>;访问pair.fisrt->second;   代码:  multimap<string, string> mulmap1({ {"liu","c"} ,{ "liu2","c" } ,{ "liu3","c" } ,{ "liu","a" } ,{ "liu","b" } });  string search\_item("liu");  auto entries = mulmap1.count("liu");  auto iter1 = mulmap1.find(search\_item);  while (entries) {  cout << (\*iter1).second << endl;  ++iter1;  --entries;  }  auto beg = mulmap1.lower\_bound("liu");  auto end = mulmap1.upper\_bound("liu");  while (beg != end) {  cout << (\*beg).second << endl;  ++beg;  }  auto range1 = mulmap1.equal\_range("liu");  while (range1.first != range1.second) {  cout << range1.first->second << endl;  cout << (\*range1.first).second << endl;  ++range1.first;  } | |

