08.unordered_map和unordered_set的使用

1. unordered_set系列的使用

1.1 unordered_set和unordered_multiset参考文档

https://legacy.cplusplus.com/reference/unordered_set/

1.2 unordered_set类的介绍

- unordered_set的声明如下，Key就是unordered_set底层关键字的类型
- unordered_set默认要求Key支持转换为整形，如果不支持或者想按自己的需求走可以自行实现支持将Key转成整形的仿函数传给第二个模板参数
- unordered_set默认要求Key支持比较相等，如果不支持或者想按自己的需求走可以自行实现支持将Key比较相等的仿函数传给第三个模板参数
- unordered_set底层存储数据的内存是从空间配置器申请的，如果需要可以自己实现内存池，传给第四个参数。
- 一般情况下，我们都不需要传后三个模板参数
- unordered_set底层是用哈希桶实现，增删查平均效率是 $O(1)$ ，迭代器遍历不再有序，为了跟set区分，所以取名unordered_set。
- 前面部分我们已经学习了set容器的使用，set和unordered_set的功能高度相似，只是底层结构不同，有一些性能和使用的差异，这里我们只讲他们的差异部分。

```
template < class Key,                          //
unordered_set::key_type/value_type
           class Hash = hash<Key>,             // unordered_set::hasher
           class Pred = equal_to<Key>,         // unordered_set::key_equal
           class Alloc = allocator<Key>        // unordered_set::allocator_type
           > class unordered_set;
```

1.3 unordered_set和set的使用差异

- 查看文档我们会发现unordered_set的支持增删查且跟set的使用一模一样，关于使用我们这里就不再赘述和演示了。
- unordered_set和set的第一个差异是对key的要求不同，set要求Key支持小于比较，而unordered_set要求Key支持转成整形且支持等于比较，要理解unordered_set的这个两点要求得后续我们结合哈希表底层实现才能真正理解，也就是说这本质是哈希表的要求。

- unordered_set和set的第二个差异是迭代器的差异，set的iterator是双向迭代器，unordered_set是单向迭代器，其次set底层是红黑树，红黑树是二叉搜索树，走中序遍历是有序的，所以set迭代器遍历是有序+去重。而unordered_set底层是哈希表，迭代器遍历是无序+去重。
- unordered_set和set的第三个差异是性能的差异，整体而言大多数场景下，unordered_set的增删查改更快一些，因为红黑树增删查改效率是 $O(logN)$ ，而哈希表增删查平均效率是 $O(1)$ ，具体可以参看下面代码的演示的对比差异。

```
pair<iterator,bool> insert ( const value_type& val );
size_type erase ( const key_type& k );
iterator find ( const key_type& k );
```

```
#include<unordered_set>
#include<unordered_map>
#include<set>
#include<iostream>
using namespace std;

int test_set2()
{
    const size_t N = 1000000;

    unordered_set<int> us;
    set<int> s;

    vector<int> v;
    v.reserve(N);
    srand(time(0));
    for (size_t i = 0; i < N; ++i)
    {
            //v.push_back(rand()); // N比较大时，重复值比较多
            v.push_back(rand()+i); // 重复值相对少
            //v.push_back(i); // 没有重复，有序
    }

    // 21:15
    size_t begin1 = clock();
    for (auto e : v)
    {
            s.insert(e);
    }
    size_t end1 = clock();
    cout << "set insert:" << end1 - begin1 << endl;

```

```
33      size_t begin2 = clock();
34      us.reserve(N);
35      for (auto e : v)
36      {
37              us.insert(e);
38      }
39      size_t end2 = clock();
40      cout << "unordered_set insert:" << end2 - begin2 << endl;
41
42      int m1 = 0;
43      size_t begin3 = clock();
44      for (auto e : v)
45      {
46              auto ret = s.find(e);
47              if (ret != s.end())
48              {
49                      ++m1;
50              }
51      }
52      size_t end3 = clock();
53      cout << "set find:" << end3 - begin3 << "->" << m1 << endl;
54
55      int m2 = 0;
56      size_t begin4 = clock();
57      for (auto e : v)
58      {
59              auto ret = us.find(e);
60              if (ret != us.end())
61              {
62                      ++m2;
63              }
64      }
65      size_t end4 = clock();
66      cout << "unorered_set find:" << end4 - begin4 << "->" << m2 << endl;
67
68      cout << "插入数据个数: " << s.size() << endl;
69      cout << "插入数据个数: " << us.size() << endl << endl;
70
71      size_t begin5 = clock();
72      for (auto e : v)
73      {
74              s.erase(e);
75      }
76      size_t end5 = clock();
77      cout << "set erase:" << end5 - begin5 << endl;
78
79      size_t begin6 = clock();
```

```
    for (auto e : v)
    {
            us.erase(e);
    }
    size_t end6 = clock();
    cout << "unordered_set erase:" << end6 - begin6 << endl << endl;

    return 0;
}

int main()
{
    test_set2();

    return 0;
}
```

1.4 unordered_map和map的使用差异

- 查看文档我们会发现unordered_map的支持增删查改且跟map的使用一模一样，关于使用我们这里就不再赘述和演示了。
- unordered_map和map的第一个差异是对key的要求不同，map要求Key支持小于比较，而unordered_map要求Key支持转成整形且支持等于比较，要理解unordered_map的这个两点要求得后续我们结合哈希表底层实现才能真正理解，也就是说这本质是哈希表的要求。
- unordered_map和map的第二个差异是迭代器的差异，map的iterator是双向迭代器，unordered_map是单向迭代器，其次map底层是红黑树，红黑树是二叉搜索树，走中序遍历是有序的，所以map迭代器遍历是Key有序+去重。而unordered_map底层是哈希表，迭代器遍历是Key无序+去重。
- unordered_map和map的第三个差异是性能的差异，整体而言大多数场景下，unordered_map的增删查改更快一些，因为红黑树增删查改效率是 $O(logN)$ ，而哈希表增删查平均效率是 $O(1)$ ，具体可以参看下面代码的演示的对比差异。

```
pair<iterator,bool> insert ( const value_type& val );
size_type erase ( const key_type& k );
iterator find ( const key_type& k );
mapped_type& operator[] ( const key_type& k );
```

1.5 unordered_multimap/unordered_multiset

- unordered_multimap/unordered_multiset跟multimap/multiset功能完全类似，支持Key冗余。

- unordered_multimap/unordered_multiset跟multimap/multiset的差异也是三个方面的差异，key的要求的差异，iterator及遍历顺序的差异，性能的差异。

1.6 unordered_xxx的哈希相关接口

Buckets和Hash policy系列的接口分别是跟哈希桶和负载因子相关的接口，日常使用的角度我们不需要太关注，后面学习了哈希表底层，我们再来看这个系列的接口，一目了然。