关联容器

- C++ Primer第五版
- 第11章

1
1

@阿西拜-南昌

关联容器：元素是按关键字来保存和访问的
顺序容器：元素是按他们在容器中的位置来顺序保存和访问的

关联容器类型	
按关键字有序保存元素	
map	关联数组；保存关键字-值对
set	关键字即值，即只保存关键字的容器
multimap	关键字可重复出现的 map
multiset	关键字可重复出现的 set
无序集合	
unordered_map	用哈希函数组织的 map
unordered_set	用哈希函数组织的 set
unordered_multimap	哈希组织的 map；关键字可以重复出现
unordered_multiset	哈希组织的 set；关键字可以重复出现

map和multimap 头文件map
u_map u_mumap 头文件unordered_map中
set同理

使用关联容器

使用map　　first 下标（关键字）　second 对应的值 value

```
//统计每个单词在输入中出现的次数
map<string, size_t> word_cout ; //string到size_t的空map
string word;
while (cin >> word)
      ++word_cout[word]; //提取word的计算器并将其加1
for (const auto& w : word_cout) //对map中的每个元素
      cout << w.first << " occurs " << w.second
           << ((w.second > 1) ? "times" : "time") << endl;
```

```
Microsoft Visual Studio 调试控制台
hello
hello
hi
^Z
hello occurs 2times
hi occurs 1time
```

使用set

```
//统计每个单词在输入中出现的次数
map<string, size_t> word_cout ; //string到size_t的空map
set<string> exclude = { "The","But","And","Or","An","A",
                        "the","but","and","or","an","a" };
string word;
while (cin >> word)
      if(exclude.find(word)==exclude.end())//只统计不在exclude中的单词
            ++word_cout[word]; //提取word的计算器并将其加1
for (const auto& w : word_cout) //对map中的每个元素
      cout << w.first << " occurs " << w.second
           << ((w.second > 1) ? "times" : "time") << endl;
```

```
Microsoft Visual Studio 调试控制台
hello the world
^Z
hello occurs 1time
world occurs 1time
```

find 调用返回 一个迭代器 如果关键字在set中 则迭代器指向该关键字
否则find 返回尾后迭代器

关联容器概述

@阿西拜-南昌

map或set中的关键字必须是唯一的

```
//统计每个单词在输入中出现的次数
map<string, size_t> word_cout ; //空map
//列表初始化
set<string> exclude = { "The","But","And","Or","An","A",
                        "the","but","and","or","an","a" };
//三个元素；authors将姓映射为名
map<string, string> authors = { {"Joyce","James"},
                        {"Austen","Jane"},{"Dickens","Charles"} };
```

初始化multimap或multiset

```
//定义一个有20个元素的vector，保存0到9每个整数的两个拷贝
vector<int> ivec;
for(vector<int>::size_type i=0;i!=10;++i){
      ivec.push_back(i);
      ivec.push_back(i);//每个数重复保存一次
}
//iset包含来自ivec的不重复的元素；miset包含所有20个元素
set<int> iset(ivec.cbegin(),ivec.cend());
multiset<int> miset(ivec.cbegin(),ivec.cend());
cout<<ivec.size()<<endl;//打印出20
cout<<iset.size()<<endl;//打印出10
cout<<miset.size()<<endl;//打印出20
```

关键字类型要求：

根据关键字比较

有序容器：map、multimap、set、multiset关键字类型必须定义元素的比较的方法。

所提供的的操作必须在关键字类型上定义一个严格弱序:

- 两个关键字不能同时“小于等于”对方；
- 如果k1“小于等于”k2，且k2“小于等于”k3，那么k1必须“小于等于”k3；
- 如果存在两个关键字，任何一个都不“小于等于”另一个，那么我们称这两个关键字是“等价”的；

如果一个类型定义了“行为正常”的<运算符，则可以用作关键字类型

```
bool compareIsbn(const Sales_data &lhs,const Sales_data &rhs)
{
      return lhs.isbn()<rhs.isbn();
}

//bookstore中多条记录可以有相同的ISBn
//bookstore中的元素以ISBN的顺序进行排列
multiset<Sales_data,decltype(compareIsbn)*> bookstore(compareIsbn);
//&compareIsbn效果一样      seconde 也必须是一个类型
```

pair类型：定义在头文件utility中

```
pair<string, string> anon;
pair<string, size_t> word_count;
pair<string, vector<int>> line;

pair<string, string> author{"James","Joyce"};
```

```
//统计每个单词在输入中出现的次数
map<string, size_t> word_cout ; //string到size_t的空map
string word;
while (cin >> word)
        ++word_cout[word]; //提取word的计算器并将其加1

for (const auto& w : word_cout) //对map中的每个元素
        cout << w.first << " occurs " << w.second
                << ((w.second > 1) ? "times" : "time") << endl;
```

pair的数据成语是public的

创建pair对象的函数

```
pair<string, int> process(vector<string> &v){
        //处理v
        if(!v.empty())
                return{ v.back(),v.back().size()};//列表初始化
                //return pair<string, int>(v.back(),v.back().size());//显式构造
                //return make_pair(v.back(),v.back().size());//推断类型
        else
                return pair<string,int>();//隐式构造返回值
}
```

关联容器操作

@阿西拜-南昌

关联容器额外的类型别名	
key_type	此容器类型的关键字类型
mapped_type	每个关键字关联的类型；只适用于 map　　map的 value
value_type	对于 set，与 key_type 相同
	对于 map，为 pair<const key_type, mapped_type>

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

关联容器迭代器　　尽量使用关联容器的迭代器　而不是泛型的

```
//获得指向word_count中一个元素的迭代器
auto map_it = word_count.begin();
//*map_it是指向pair<const string,size_t>对象的引用
cout<<map_it->first;                    //打印此元素的关键字
cout<<" "<<map_it->second;              //打印此元素的值
map_it->first="new key";                //错误：关键字是const的
++map_it->second;                       //正确：我们可以通过迭代器改变元素
```

一个map的value_type是一个pair，我们可以改变pair的值，但不能改变关键字成员的值

```
set<int> iset = {0,1,2,3,4,5,6,7,8,9};
set<int>::iterator set_it = iset.begin();
if(set_it != iset.end()){
    *set_it = 42; //错误：set中的关键字是只读的
    cout<<*set_it<<endl;//正确：可以读取关键字
}
```

set的迭代器是const的

```
//获得一个指向首元素的迭代器
auto map_it = word_count.cbegin();
//比较当前迭代器和尾后迭代器
while(map_it != word_count.cend()){
    //解引用迭代器，打印关键字-值对
    cout<<map_it->first<<" occurs"<<map_it->second<<" times" <<endl;
    ++map_it;//递增迭代器，移动到下一个元素
}
```

map、set、multimap，multiset迭代器按关键字升序遍历元素

通常不对关联容器使用泛型算法，

添加元素

```
//统计每一个单词在输入中出现次数
map<string,size_t> word_cout;//从string到size_t的空map
string word;
while(cin>>word){
      //插入一个元素，关键字等于word，值为1；
      //若word已在word_count中，insert什么也不做
      auto ret = word_count.insert({word,1});
      if(!ret.second)//word已经在word_count中
          ++ret.first->second;//递增计数器
}
```

first：指向具体给定关键字的元素
second：支出元素是否插入成功

ret 的类型为 pair<map<,>, bool>
所以返回second是~~true~~表示是否插入成功 如果已有就插入失败(单词重复了)会执行对val++

```
++((ret.first)->second);//等价表达式
//ret保存insert返回的值，是一个pair。
//ret.first是一个map迭代器：map<string,size_t>::iterator
// ret.first->解引用迭代器，提取map中的元素，是pair。
//ret.first->second是map中元素的值部分
```

删除元素

从关联容器删除元素	
c.erase(k)	从 c 中删除每个关键字为 k 的元素。返回一个 size_type 值，指出删除的元素的数量
c.erase(p)	从 c 中删除迭代器 p 指定的元素。p 必须指向 c 中一个真实元素，不能等于 c.end()。返回一个指向 p 之后元素的迭代器，若 p 指向 c 中的尾元素，则返回 c.end()
c.erase(b, e)	删除迭代器对 b 和 e 所表示的范围中的元素。返回 e

```
//删除一个关键字，返回删除的元素数量
if(word_count.erase(removal_word))
      cout<<"ok: "<<removal_word<<" removed\n";
else cout<<"oops: "<<removal_word<<" not found!\n";
//对于不能重复关键字的容器，返回0或1
```

map的下标操作

@阿西拜-南昌

map和unordered_map的下标操作	
c[k]	返回关键字为 k 的元素；如果 k 不在 c 中，添加一个关键字为 k 的元素，对其进行值初始化
c.at(k)	访问关键字为 k 的元素，带参数检查；若 k 不在 c 中，抛出一个 out_of_range 异常

```
map<string,size_t> word_count;//empty map
//插入一个关键字为Anna的元素，关联值进行初始化；然后将1赋予它
word_count["Anna"] = 1;
```

操作下标对象：mapped_type
对下标解引用 是 value_type

- 在 word_count 中搜索关键字为 Anna 的元素，未找到。
- 将一个新的关键字-值对插入到 word_count 中。关键字是一个 const string，保存 Anna。值进行值初始化，在本例中意味着值为 0。
- 提取出新插入的元素，并将值 1 赋予它。

下标运算符可能插入一个新元素，我们只可以对非const的map使用下标操作

访问元素

在一个关联容器中查找元素的操作	
lower_bound 和 upper_bound 不适用于无序容器。	
下标和 at 操作只适用于非 const 的 map 和 unordered_map。	
c.find(k)	返回一个迭代器，指向第一个关键字为 k 的元素，若 k 不在容器中，则返回尾后迭代器
c.count(k)	返回关键字等于 k 的元素的数量。对于不允许重复关键字的容器，返回值永远是 0 或 1
c.lower_bound(k)	返回一个迭代器，指向第一个关键字不小于 k 的元素
c.upper_bound(k)	返回一个迭代器，指向第一个关键字大于 k 的元素
c.equal_range(k)	返回一个迭代器 pair，表示关键字等于 k 的元素的范围。若 k 不存在，pair 的两个成员均等于 c.end()

```
set<int> iset = {0,1,2,3,4,5,6,7,8,9};
iset.find(1); //返回一个迭代器，指向key==1的元素
iset.find(11);//返回一个迭代器，其值等于iset.end()
iset.count(1);//返回1
iset.count(11);//返回0
```

在multimap或multimap中查找元素

```
string search_item("Alain de Botton"); //要查找的作者
auto entries = authors.count(search_item); //元素的数量
auto iter = authors.find(search_item); //此作者的第一本书
//用一个循环查找此作者的所有著作
while(entries){
        cout<<iter->second<<endl;//打印书名
        ++iter;//前进到下一本书
        --entries;//记录以及打印了多少本书
}
```

具有相同关键字的这些元素在容器中会相邻存储

```
//authors和search_item的定义，与前面的程序一样
//beg和end表示对应此作者的元素的范围
for(auto beg = authors.lower_bound(search_item),
        end = authors.upper_bound(search_item);
        beg != end; ++beg)
        cout << beg->second << endl; //打印书名
```

如果lower_bound和upper_bound返回相同的迭代器，则给定关键字不在容器中

```
//authors和search_item的定义，与前面的程序一样
//pos保存迭代器对，表示与关键字匹配的元素范围
for(auto pos = authors.equoal_range(search_item);
        pos.first != pos.second; ++pos.first)
        cout<<pos.first->second<<endl; //打印书名
```

一个单词转换的map：

- 给定一个string，将它转换为另一个string
- 程序的输入是两个文件：第一个保存的是一些规则，第二个是需要转换的文本。

```
brb be right back
k okay?
y why
r are
u you
pick picture
thk thanks
18r later
```

```
where r u
y dont u send me a pic
k thk l8r
```

输入

```
where are you
why dont you send me a picture
okay? thanks! later
```

输出

@阿西拜-南昌

```
void word_transform(ifstream &map_file, ifstream &input){
    auto trans_map = buildMap(map_file); //保存转换规则
    string text;
    while( getline(input,text)){//读取一行输入
        istringstream stream(text);//读取每个单词
        string word;
        bool firstword = true; //控制是否打印空格
        while( stream >> word){
            if(firstword)
                firstword = false;
            else
                cout << " ";
                cout << transform(word,trans_map);//打印输出
        }
        cout << endl; //完成一行的转换
    }
}
```

```
//读入给定文件，建立起转换映射
map<string, string> buildMap(ifstream &map_file){
    map<string, string> trans_map;//保存转换规则
    string key; //要转换的单词
    string value; //替换后的内容
    //读取第一个单词存入key中，行中剩余内容存入value
    while(map_file >> key && getline(map_file,value))
        if(value.size()>1)//检查是否有转换规则
            trans_map[key] = value.substr(1); //跳过前导空格
        else
            throw runtime_error("no rule for " + key);
    return trans_map;
}
```

```
//生成转换文本
const string & transform(const string &s, const map<string, string> &m)
{
    //实际的转换工作；此部分是程序的核心
    auto map_it = m.find(s);
    //如果单词在转换规则map中
    if(map_it != m.cend())
        return map_it->second; //使用替换短语
    else
        return s; //否则返回string
}
```

无序容器

不使用比较运算符来组织元素，而是使用哈希函数和关键字类型的==运算符

```
//统计出现次数，但单词不按字典排序
unordered_map<string, size_t> word_count;
string word;
while(cin>>word)
        ++word_count[word];
for(const auto &w:word_count)
        cout<<w.first<<" occurs " <<w.second
             <<((w.second>1)?" times":" time")<<endl;
```

管理桶：无序容器的性能依赖于哈希函数的质量和桶的数量大小

无序容器管理操作

桶接口	
c.bucket_count()	正在使用的桶的数目
c.max_bucket_count()	容器能容纳的最多的桶的数量
c.bucket_size(n)	第 n 个桶中有多少个元素
c.bucket(k)	关键字为 k 的元素在哪个桶中
桶迭代	
local_iterator	可以用来访问桶中元素的迭代器类型
const_local_iterator	桶迭代器的 const 版本
c.begin(n), c.end(n)	桶 n 的首元素迭代器和尾后迭代器
c.cbegin(n), c.cend(n)	与前两个函数类似，但返回 const_local_iterator
哈希策略	
c.load_factor()	每个桶的平均元素数量，返回 float 值
c.max_load_factor()	c 试图维护的平均桶大小，返回 float 值。c 会在需要时添加新的桶，以使得 load_factor<=max_load_ factor
c.rehash(n)	重组存储，使得 bucket_count>=n 且 bucket_count>size/max_load_factor
c.reserve(n)	重组存储，使得 c 可以保存 n 个元素且不必 rehash

无序容器默认情况下：

@阿西拜-南昌

- 使用关键字类型的==运算符比较元素
- 使用一个hash<key_type>类型的对象来生成每个元素的哈希值
  - 标准库为内置类型（包括指针），string，智能指针提供了hash模板

不能直接定义关键字类型为自定义类类型的无序容器

```
//为了将Sale_data用作关键字，我们需要提供：
//1、函数来替代==运算符
//2、哈希函数
size_t haser(const Sales_data &sd)
{
	return hash<string>()(sd.isbn());//利用标准库hash类型对象来计算
}

bool eqOp(const Sales_data &lhs, const Sales_data &rhs)
{
	return lhs.isbn() == rhs.isbn();
}

using SD_multiset = unordered_multiset<Sales_data,decltype(haser)*,decltype(
eqOp)*>;
//参数是（桶大小、哈希函数指针、相等性判断运算符指针
SD_multiset bookstore(42,hasher,eqOp);
//假设Foo有==运算符
unordered_set<Foo,decltype(FooHash)*> foodSet(10,FooHash);
```