第11章 关联容器

> 关联容器中的元素是按关键字来保存和访问的

- 按关键字有序保存元素
  - map
  - set
  - multimap
  - multiset
- 无序集合
  - unordered_map
  - unordered_set
  - unordered_multimap
  - unordered_multiset

> 无序的容器有其特有的头文件；
>
> 无序容器使用哈希函数组织元素

关联容器概述

- 关联容器不支持位置的相关操作，只能用关键字访问；

  关联容器有其特有操作和**类型别名**
- 关联容器的迭代器其都是双向的

定义关联容器

```
/*
支持的形式
1. 空容器
2. 列表初始化
3. 值初始化/值范围初始化   （只能这些值可以转换为容器所需要都类型）
*/
map<string, int> mp1;
map<string, int> mp2={{"1",1}};
map<string, int> mp2{{"1",1}};
set<int> st1(v.cbegin(), v.cend());     // v是vector<int>
```

关键字类型的要求

关联容器对其关键字类型有一些限制

- 无序关联容器：**后续介绍**
- 有序关联容器：关键字类型**内**必须定义元素比较的方法(默认`<运算符`比较)
    - 自定义排序器
        1. 如果容器中直接存储对象的话，那么我们可以在对象类中重载`<`即可;
        2. 提供比较器
            - 类内重载函数调用运算符的方法

              > 需要额外定义一个类

            - 以函数的方式提供比较器

              > 需要用函数初始化关联容器

```
struct Compare{
    //override the operator ()
    bool operator()(const Student &ls, const Student &rs)const{
        return ls.getId() < rs.getId();
    }
};

bool compare(const Student &ls, const Student &rs){
    return ls.getId() < rs.getId();
}

int main(){
    /*the first type-----define a class as the comparator*/
    set<Student, Compare> ms;
    ...
    /*the second type----define a function as the comparator*/
    set<Student, decltype(compare)*> ms(compare);   // 第二个函数需要是对应的函数指针
    ...
    return 0;
}
```

pari类型

`<utility>`

- 一个pair保存两个数据成员，默认进行值初始化。
- pair的数据成员是public的，分别命名为 `first` 和 `second`

表 11.2：pair 上的操作

`pair<T1, T2> p;`	p 是一个 pair，两个类型分别为 T1 和 T2 的成员都进行了值初始化（参见 3.3.1 节，第 88 页）
`pair<T1, T2> p(v1, v2)`	p 是一个成员类型为 T1 和 T2 的 pair；first 和 second 成员分别用 v1 和 v2 进行初始化
`pair<T1,T2>p = {v1,v2};`	等价于 p（v1,v2）
`make_pair(v1, v2)`	返回一个用 v1 和 v2 初始化的 pair。pair 的类型从 v1 和 v2 的类型推断出来
`p.first`	返回 p 的名为 first 的（公有）数据成员
`p.second`	返回 p 的名为 second 的（公有）数据成员
`p1 relop p2`	关系运算符（<、>、<=、>=）按字典序定义：例如，当 p1.first < p2.first 或!(p2.first < p1.first) && p1.second < p2.second 成立时，p1 < p2 为 true。关系运算利用元素的< 运算符来实现
`p1 == p2`	当 first 和 second 成员分别相等时，两个 pair 相等。相等性判断利用元素的==运算符实现
`p1 != p2`	

> **C++11**，支持使用花括号包围的初始器来返回pair对象。

关联容器操作

关联容器的额外类型别名

除了顺序容器介绍的类型别名外，还支持额外的类型别名

- `key_type`：容器的关键字类型
- `mapped_type`：每个关键字关联的类型

  > 只适用于map

- `value_type`：对于set，key_type相同；对于map，为 `pair<const key_type, mapped _type>`、

  > 由这个类型可知，我们不能改变关联容器元素的关键字

关联容器的迭代器

- 解引用得到的是`value_type`类型
- 通常不对关联容器使用泛型算法
  - 不支持修改或重排关键字
  - 可用于只读取元素的算法

    > 但是不支持快速搜索，使用泛型算法效率低

  > 如果真要使用，要么将其当作一个源序列，要么当作一个目标位置

添加元素

```
c.insert(v);    c.emplace(args);
c.insert(b, e); c.insert({x,x,x,x});
c.insert(p,v);  c.empalce(p, args);          // ? 迭代器p作为一个提示，指出从哪里开始搜索新元素应该出现的位置
```

- insert的返回值
  - 对于不包含重复关键字的容器，返回一个pair；
    - first成员是一个迭代器，指出具有给定关键字的元素
    - second成员是一个bool， 指出本次插入是否成功还是该元素已经在容器中
  - 对于允许包含重复关键字的容器，返回一个指向新元素的迭代器

    > 总会插入一个元素

删除元素

```
c.erase(k);         // 删除每个关键字为k的元素，返回size_type，指出删除元素的数量
c.erase(p);
c.erase(b, 3);
```

map的下标操作

关联容器中，下标操作是获取与一个关键字相关联的值，**只有`map & unordered_map`支持**

> set 不存在关键字对应的值；multimap可能有多个值与一个关键字对应

```
map[k];          // 返回关键字k对应的元素，如果不存在，则添加该关键字，并进行值初始化
map.at(k)        // 访问关键字为k的值，带参数检查。如果不存在，则抛出out_of_range
```

> k不一定需要整数类型，支持任何类型

查找元素

```
c.find(k);       // 返回一个迭代器，指向第一个关键字为k的元素。不在则返回尾后迭代器
c.count(k);      // 返回关键子等于k的元素的数量
c.lower_bound(k);    // 返回一个迭代器，指向第一个关键字为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_bound(k);    // 返回一个迭代器pair，分别是lower_bound，upper_bound
```

- lower_bound & upper_bound 不支持无序容器；

  当两者返回相同迭代器时，则给定关键字不存在，但是**指向的位置是一个不影响排序的该关键字插入位置**

无序容器

> 在没有要求关键有序的情况下，推荐使用无序容器

- 除了有序容器特别说明的操作，无序容器与有序容器操作相同

管理桶

- 无序容器在存储上组织为一组桶；

  每个桶保存零个或多个元素
- 无序容器使用一个哈希函数将元素映射到桶；

  为了访问一个元素，容器首先计算元素的哈希值，它指出应该搜索哪个桶；

  容器将具有特定哈希值的所有元素都保存在一个桶中
- 无序容器的性能依赖与哈希函数的质量和桶的数量和大小

表 11.8：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

无序容器对关键字类型的要求

默认情况下，

- 无序容器使用关键字类型 `==运算符` 来比较元素，
- 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和智能指针类型（12章）定义了hash

如果要使用其他类型，必须

- 定义对应类型的哈希模版（16章）
- 重载 `==运算符`

```
unordered_multiset<自定义类，哈希函数指针，重定义==替代函数> t;
// 类中重载了==
unordered_multiset<自定义类，哈希函数指针>
```

> 是否可将函数指针，代替为函数可调用对象？？？ **测试**

扩展阅读

- 有序容器自主定义排序器