C++ Primer第十一章

关联容器类型

类型 map 和 **multimap** 定义在头文件 map 中；set 和 **multiset** 定义在头文件 set 中；无序容器则定义在头文件 unordered_map 和 unordered_set 中。

表 11.1：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

使用关联容器

使用map

从map提取元素，会得到一个pair类型。
pair类型是一个模板，保存first second（共有）数据成员
first保存关键字，second保存对应的值

使用set

必须指定容器类型

可以对一个关联容器进行列表初始化

概述

定义关联容器

不支持顺序容器的位置相关操作

初始化multimap或multiset

map、set关键字必须是唯一的，给定一个关键字只能有一个元素等于它

multimap、multiset则不然，允许多个元素有相同关键字

关键字类型的要求

Note

传递给排序算法的可调用对象（参见 10.3.1 节，第 344 页）必须满足与关联容器中关键字一样的类型要求。

子主题

有序容器关键字类型

严格弱序：小于等于

实际编程中，如果一个类型定义了“行为”正常的<运算符，它可以用作关键字类型

使用关键字类型比较函数

为了使用自己定义的操作，定义multiset时
我们必须提供两个类型

关键字类型，可以是一个自己定义的类

比较操作类型——函数指针类型

使用decltype来获得一个函数指针类型时，必须加上*指出
我们要使用一个给定函数类型的指针

pair类型

默认构造函数是对数据成员进行值初始化，
同时也提供初始化器

数据成员是public，成员命名为first 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。关系运算符用元素的<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	

创建pair对象的函数

新标准下，可以对返回值进行列表初始化

可以使用make_pair函数生成pair对象

关联容器操作

关联容器迭代器

只有map类型才定义了mapped_type

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

set迭代器是const的

遍历关联容器　支持begin() end()操作，获得迭代器来遍历容器

关联容器和算法

关联容器的元素不能通过他们关键字快速查找，因此对其直接使用泛化搜索算法不太恰当

如要使用关联容器算法：
要么作为源序列（比如拷贝关联容器到另一个序列）
要么作为一个目的位置（比如inserter绑定）

表 11.3：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
value_type	对于 set，与 key_type 相同 对于 map，为 pair<const key_type, mapped_type>

添加元素

map set包含不重复关键字，因此插入一个已经存在的元素对容器没有如何影响

map添加元素

表 11.4. 关联容器 insert 操作

c.insert(v)	v 是 value_type 类型的对象；args 用来构造一个元素
c.emplace(args)	对于 map 和 set，只有当元素的关键字不在 c 中时才插入（或构造）元素。函数返回一个 pair，包含一个迭代器，指向具有指定关键字的元素，以及一个指示插入是否成功的 bool 值。 对于 multimap 和 multiset，总会插入（或构造）给定元素，并返回一个指向新元素的迭代器
c.insert(b, e)	b 和 e 是迭代器，表示一个 c::value_type 类型值的范围；il 是这种值的花括号列表。函数返回 void
c.insert(il)	对于 map 和 set，只插入关键字不在 c 中的元素。对于 multimap 和 multiset，则会插入范围中的每个元素
c.insert(p, v) c.emplace(p, args)	类似 insert(v)（或 emplace(args)），但将迭代器 p 作为一个提示，指出从哪里开始搜索新元素应该存储的位置。返回一个迭代器，指向具有给定关键字的元素

检测insert的返回值

添加单一元素的insert emplace版本返回一个pair
pair的first成员是一个迭代器，指向具有给定关键字的元素
pair的second是bool值，指出插入是否成功还是存在容器中

向multiset multimap添加元素　接受单个元素的insert操作返回一个指向新元素的迭代器

删除元素

和顺序容器一样，可以通过传递给erase一个迭代器和
一个迭代器对来删除一个元素或者一个元素访问

额外操作——接受一个key_type参数，返回实际删除的元素数量

表 11.5：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

map下标操作

map unordered_map容器提供下标运算符和一个对应的at函数
set类型不支持下标

不能对multimap 或 unordered_multimap进行下标操作

Note

对一个 map 使用下标操作，其行为与数组或 vector 上的下标操作很不相同：使用一个不在容器中的关键字作为下标，会添加一个具有此关键字的元素到 map 中。

表 11.6：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（参见 5.6 节，第 173 页）

使用下标操作的返回值

通常解引用一个迭代器返回类型和下标运算符返回类型一样，但map不同

对map进行下标操作时，会获得一个mapped_type对象
解引用会map迭代器会得到value_type对象

表 11.7. 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。

11.3 关联容器操作

续表

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()

访问元素

对map使用find代替下标

在multimap或multiset中查找元素　与map set不一样的地方，它们具有可以存储相同的元素，相邻排列

另一种方法：一种不同的，面向迭代器的解决办法

lower_bound,upper_bound划定范围

Note

lower_bound 返回的迭代器可能指向一个具有给定关键字的元素，但也可能不指向。如果关键字不在容器中，则 lower_bound 会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

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。

第三种方法：equal_range函数

接受一个关键字，返回一个迭代器pair
如果关键字存在，第一个迭代器指向第一个与关键字匹配的元素
第二个迭代器指向最后一个与关键字匹配的元素

一个元素转换的map

这里涉及到文件简约词汇的替换

无序容器

关键字类型的元素没有明显的序关系情况下，无序容器是十分有用的

Tip

如果关键字类型固有就是无序的，或者性能测试发现问题可以用哈希技术解决，就可以使用无序容器。

使用无序容器

哈希管理操作 + 有序容器相同的操作

map和set的操作也能用于unordered_map 和 unordered_set

管理桶

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

默认情况下，无序容器使用关键字类型==运算符来比较元素，还使用一个hash<key_type>类型的对象来生成每个元素的哈希值

无序容器对关键字类型的要求

标准库为内置类型（包括指针）提供了hash模板

与容器不同，不能直接使用哈希模板，而必须提供我们自己的hash模板版本