第11章 关联容器

1. 标准库提供8个关联容器：`map`、`set`、`multimap`、`multiset`、`unordered_map`、`unordered_set`、`unordered_multimap`、`unordered_multiset`。
2. 8个容器间的不同体现在三个维度上：
   （1）每个容器要么是一个`set`、要么是一个`map`；
   （2）要么要求不重复的关键字、要么允许重复关键字（`multi`）；
   （3）要么按顺序保存元素、要么无序保存（`unordered`）。

11.2 关联容器概述

11.2.1 定义关联容器

1. 可以将关联容器初始化为另一个同类型容器的拷贝。
2. 新标准下可以对关联容器进行值初始化

```
vector<int> ivec = { 0, 1, 2 };
set<int> iset(ivec.begin(), ivec.end());
map<int, int> imap = { {0, 1}, {2, 3} };
```

11.2.2 关键字类型的要求

1. 对于有序容器，关键字类型必须定义元素比较的方法。默认情况下，标准库使用关键字类型的 < 运算符来比较两个关键字。
2. 传递给排序算法的可调用对象必须满足与关联容器中关键字一样的类型要求。
3. 可以向一个算法提供我们自己定义的比较操作，与之类似，也可以提供自己定义的操作来代替关键字上的 < 运算符。所提供的操作必须在关键字类型上定义一个 *严格弱序（strict weak ordering）*。
4. 实例化无<运算符的类型作为关键字类型的关联容器：

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

multiset<Sales_data, decltype(compareIsbn)*> bookstore(compareIsbn);
```

5. 使用 `decltype` 来指出自定义操作的类型，必须加上一个*来指出要使用一个给定函数类型的指针
6. 使用 `compareIsbn` 来初始化 `bookstore` 对象，表示当向容器添加元素时，通过调用 `compareIsbn` 来为这些元素排序
7. 可以使用 `compareIsbn` 代替 `&compareIsbn` 作为构造函数的参数，因为当我们使用一个函数名字的时候，在需要的情况下它会自动转化为一个指针。

11.3 关联容器操作

1. 关联容器额外的类型别名： `key_type` 、 `mapped_type` 和 `value_type` 。

11.3.1 关联容器迭代器

1. 虽然 `set` 类型同时定义了 `iterator` 和 `const_iterator` 类型，但两种类型都只允许 *只读访问* `set` 中的元素。
2. 在实际编程中，如果真要对一个关联容器使用算法，要么是将它当作一个源序列，要么当作一个目的位置。

11.3.2 添加元素

1. 由于 `map` 和 `set` （以及对应的无序类型）包含不重复的关键字，因此插入一个已存在的元素对容器没有任何影响。
2. `insert` （或 `emplace` ）返回的值依赖于容器类型和参数。对于不包含重复关键字的容器，添加单一元素的 `insert` 和 `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； `second` 成员是一个 `bool` 值，指出元素是插入成功还是已经存在与容器中。
3. 对于允许重复关键字的容器，接受单个元素的 `insert` 操作返回一个指向新元素的迭代器。无须返回一个 `bool` 值。

11.3.3 删除元素

1. 关联容器提供一个额外的 `erase` 操作，它接受一个 `key_type` 参数。此版本删除所有匹配给定关键字的元素（如果存在的话），返回实际删除的元素的数量。

11.3.4 `map` 的下标操作

1. `set` 类型不支持下标，因为 `set` 中没有与关键字相关联的 “值”。
2. 不能对一个 `multimap` 或一个 `unordered_map` 进行下标操作，因为这些容器可能有多个值与一个关键字相关联。
3. 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（同理：调用容器的 `at()` 操作，会抛出一个 `out_of_range` 异常）。

11.3.5 访问元素

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。
2. 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。
3. `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。
4. 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。
5. `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()`。
6. 如果一个 `multimap` 或 `multiset` 中有多个元素具有相同的给定关键字，则这些元素在容器中会相邻存储。

11.4 无序容器

1. 新标准定义了4个 *无序关联容器（unordered associative container）*。这些容器不是使用比较运算符来组织元素，而是使用一个 *哈希函数（hash function）* 和关键字类型的 `==` 运算符。
2. 无序容器在存储上组织为一个 *桶（bucket）*。
3. 容器将具有一个特定哈希值的所有元素都保存在相同的桶中。如果容器允许重复关键字，所有具有相同关键字的元素也都会在同一个桶中。
4. 无序容器的性能依赖于哈希函数的质量和桶的数量和大小。
5. 可以直接定义关键字是 *内置类型（包括指针类型）*、`string` 还有 *智能指针类型* 的无序容器。
6. 不能直接定义关键字类型为自定义类型的无序容器，必须提供特定的 hash 模板版本:

```
// 为了能将 Sales_data 作为关键字，需提供函数来替代==运算符和哈希值计算函数。
size_t hasher(const Sales_data &sd)
{
    return hash<string>() (sd.isbn());
}

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

// 使用类型别名简化类型
using SD_multiset = unordered_multiset<Sales_data,
                        decltype(hasher)*, decltype(eqOp)*>;

// 参数是：桶大小、哈希函数指针和相等性判断运算符指针
SD_multiset bookstore(42, hasher, eqOp);
```