C++ STL unordered_map容器用法详解

C++ STL 标准库中提供有 4 种无序关联式容器，本节先讲解 unordered_map 容器。

unordered_map 容器，直译过来就是"无序 map 容器"的意思。所谓“无序”，指的是 unordered_map 容器不会像 map 容器那样对存储的数据进行排序。换句话说，unordered_map 容器和 map 容器仅有一点不同，即 map 容器中存储的数据是有序的，而 unordered_map 容器中是无序的。

> 对于已经学过 map 容器的读者，可以将 unordered_map 容器等价为无序的 map 容器。

具体来讲，unordered_map 容器和 map 容器一样，以键值对（pair类型）的形式存储数据，存储的各个键值对的键互不相同且不允许被修改。但由于 unordered_map 容器底层采用的是哈希表存储结构，该结构本身不具有对数据的排序功能，所以此容器内部不会自行对存储的键值对进行排序。

值得一提的是，unordered_map 容器在 `<unordered_map>` 头文件中，并位于 std 命名空间中。因此，如果想使用该容器，代码中应包含如下语句：

```
#include <unordered_map>
using namespace std;
```

> 注意，第二行代码不是必需的，但如果不用，则后续程序中在使用此容器时，需手动注明 std 命名空间（强烈建议初学者使用）。

unordered_map 容器模板的定义如下所示：

```
template < class Key,                                //键值对中键的类型
           class T,                                  //键值对中值的类型
           class Hash = hash<Key>,                   //容器内部存储键值对所用的哈希函数
           class Pred = equal_to<Key>,               //判断各个键值对键相同的规则
           class Alloc = allocator< pair<const Key,T> >  // 指定分配器对象的类型
           > class unordered_map;
```

以上 5 个参数中，必须显式给前 2 个参数传值，并且除特殊情况外，最多只需要使用前 4 个参数，各自的含义和功能如表 1 所示。

表 1 unordered_map 容器模板类的常用参数

参数	含义
<key,T>	前 2 个参数分别用于确定键值对中键和值的类型，也就是存储键值对的类型。

Hash = hash<Key>	用于指明容器在存储各个键值对时要使用的哈希函数，默认使用 STL 标准库提供的 hash<key> 哈希函数。注意，默认哈希函数只适用于基本数据类型（包括 string 类型），而不适用于自定义的结构体或者类。
Pred = equal_to<Key>	要知道，unordered_map 容器中存储的各个键值对的键是不能相等的，而判断是否相等的规则，就由此参数指定。默认情况下，使用 STL 标准库中提供的 equal_to<key> 规则，该规则仅支持可直接用 == 运算符做比较的数据类型。

> 总的来说，当无序容器中存储键值对的键为自定义类型时，默认的哈希函数 hash 以及比较函数 equal_to 将不再适用，只能自己设计适用该类型的哈希函数和比较函数，并显式传递给 Hash 参数和 Pred 参数。至于如何实现自定义，后续章节会做详细讲解。

创建C++ unordered_map容器的方法

常见的创建 unordered_map 容器的方法有以下几种。

1) 通过调用 unordered_map 模板类的默认构造函数，可以创建空的 unordered_map 容器。比如：

```
01.  std::unordered_map<std::string, std::string> umap;
```

由此，就创建好了一个可存储 <string,string> 类型键值对的 unordered_map 容器。

2) 当然，在创建 unordered_map 容器的同时，可以完成初始化操作。比如：

```
01.  std::unordered_map<std::string, std::string> umap{
02.      {"Python教程","http://c.biancheng.net/python/"},
03.      {"Java教程","http://c.biancheng.net/java/"},
04.      {"Linux教程","http://c.biancheng.net/linux/"} };
```

通过此方法创建的 umap 容器中，就包含有 3 个键值对元素。

3) 另外，还可以调用 unordered_map 模板中提供的复制（拷贝）构造函数，将现有 unordered_map 容器中存储的键值对，复制给新建 unordered_map 容器。

例如，在第二种方式创建好 umap 容器的基础上，再创建并初始化一个 umap2 容器：

```
01.  std::unordered_map<std::string, std::string> umap2(umap);
```

由此，umap2 容器中就包含有 umap 容器中所有的键值对。

除此之外，C++ 11 标准中还向 unordered_map 模板类增加了移动构造函数，即以右值引用的方式将临时 unordered_map 容器中存储的所有键值对，全部复制给新建容器。例如：

```
//返回临时 unordered_map 容器的函数
std::unordered_map <std::string, std::string > retUmap(){
    std::unordered_map<std::string, std::string>tempUmap{
        {"Python教程","http://c.biancheng.net/python/"},
        {"Java教程","http://c.biancheng.net/java/"},
        {"Linux教程","http://c.biancheng.net/linux/"} };
    return tempUmap;
}
//调用移动构造函数，创建 umap2 容器
std::unordered_map<std::string, std::string> umap2(retUmap());
```

注意，无论是调用复制构造函数还是拷贝构造函数，必须保证 2 个容器的类型完全相同。

4) 当然，如果不想全部拷贝，可以使用 unordered_map 类模板提供的迭代器，在现有 unordered_map 容器中选择部分区域内的键值对，为新建 unordered_map 容器初始化。例如：

```
//传入 2 个迭代器，
std::unordered_map<std::string, std::string> umap2(++umap.begin(),umap.end());
```

通过此方式创建的 umap2 容器，其内部就包含 umap 容器中除第 1 个键值对外的所有其它键值对。

C++ unordered_map容器的成员方法

unordered_map 既可以看做是关联式容器，更属于自成一脉的无序容器。因此在该容器模板类中，既包含一些在学习关联式容器时常见的成员方法，还有一些属于无序容器特有的成员方法。

表 2 列出了 unordered_map 类模板提供的所有常用的成员方法以及各自的功能。

表 2 unordered_map类模板成员方法

成员方法	功能
begin()	返回指向容器中第一个键值对的正向迭代器。
end()	返回指向容器中最后一个键值对之后位置的正向迭代器。
cbegin()	和 begin() 功能相同，只不过在其基础上增加了 const 属性，即该方法返回的迭代器不能用于修改容器内存储的键值对。
cend()	和 end() 功能相同，只不过在其基础上，增加了 const 属性，即该方法返回的迭代器不能用于修改容器内存储的键值对。

empty()	若容器为空，则返回 true；否则 false。
size()	返回当前容器中存有键值对的个数。
max_size()	返回容器所能容纳键值对的最大个数，不同的操作系统，其返回值亦不相同。
operator[key]	该模板类中重载了 [] 运算符，其功能是可以向访问数组中元素那样，只要给定某个键值对的键 key，就可以获取该键对应的值。注意，如果当前容器中没有以 key 为键的键值对，则其会使用该键向当前容器中插入一个新键值对。
at(key)	返回容器中存储的键 key 对应的值，如果 key 不存在，则会抛出 out_of_range 异常。
find(key)	查找以 key 为键的键值对，如果找到，则返回一个指向该键值对的正向迭代器；反之，则返回一个指向容器中最后一个键值对之后位置的迭代器（如果 end() 方法返回的迭代器）。
count(key)	在容器中查找以 key 键的键值对的个数。
equal_range(key)	返回一个 pair 对象，其包含 2 个迭代器，用于表明当前容器中键为 key 的键值对所在的范围。
emplace()	向容器中添加新键值对，效率比 insert() 方法高。
emplace_hint()	向容器中添加新键值对，效率比 insert() 方法高。
insert()	向容器中添加新键值对。
erase()	删除指定键值对。
clear()	清空容器，即删除容器中存储的所有键值对。
swap()	交换 2 个 unordered_map 容器存储的键值对，前提是必须保证这 2 个容器的类型完全相等。
bucket_count()	返回当前容器底层存储键值对时，使用桶（一个线性链表代表一个桶）的数量。
max_bucket_count()	返回当前系统中，unordered_map 容器底层最多可以使用多少桶。
bucket_size(n)	返回第 n 个桶中存储键值对的数量。
bucket(key)	返回以 key 为键的键值对所在桶的编号。
load_factor()	返回 unordered_map 容器中当前的负载因子。负载因子，指的是的当前容器中存储键值对的数量（size()）和使用桶数（bucket_count()）的比值，即 load_factor() = size() / bucket_count()。
max_load_factor()	返回或者设置当前 unordered_map 容器的负载因子。

rehash(n)	将当前容器底层使用桶的数量设置为 n。
reserve()	将存储桶的数量（也就是 bucket_count() 方法的返回值）设置为至少容纳 count个元（不超过最大负载因子）所需的数量，并重新整理容器。
hash_function()	返回当前容器使用的哈希函数对象。

> 注意，对于实现互换 2 个相同类型 unordered_map 容器的键值对，除了可以调用该容器模板类中提供的 swap() 成员方法外，STL 标准库还提供了同名的 swap() 非成员函数。

下面的样例演示了表 2 中部分成员方法的用法：

```
#include <iostream>
#include <string>
#include <unordered_map>
using namespace std;
int main()
{
    //创建空 umap 容器
    unordered_map<string, string> umap;
    //向 umap 容器添加新键值对
    umap.emplace("Python教程", "http://c.biancheng.net/python/");
    umap.emplace("Java教程", "http://c.biancheng.net/java/");
    umap.emplace("Linux教程", "http://c.biancheng.net/linux/");

    //输出 umap 存储键值对的数量
    cout << "umap size = " << umap.size() << endl;
    //使用迭代器输出 umap 容器存储的所有键值对
    for (auto iter = umap.begin(); iter != umap.end(); ++iter) {
        cout << iter->first << " " << iter->second << endl;
    }
    return 0;
}
```

程序执行结果为：

```
umap size = 3
Python教程 http://c.biancheng.net/python/
Linux教程 http://c.biancheng.net/linux/
Java教程 http://c.biancheng.net/java/
```

> 有关表 2 中其它成员方法的用法，后续章节会做详细讲解。当然，读者也可以自行查询 C++ STL标准库手册。