C++ STL unordered_map获取元素的4种方法（超级详细）

通过前面的学习我们知道，unordered_map 容器以键值对的方式存储数据。为了方便用户快速地从该类型容器提取出目标元素（也就是某个键值对的值），unordered_map 容器类模板中提供了以下几种方法。

1) unordered_map 容器类模板中，实现了对 [ ] 运算符的重载，使得我们可以像“利用下标访问普通数组中元素”那样，通过目标键值对的键获取到该键对应的值。

举个例子：

```
#include <iostream>
#include <string>
#include <unordered_map>
using namespace std;
int main()
{
    //创建 umap 容器
    unordered_map<string, string> umap{
        {"Python教程","http://c.biancheng.net/python/"},
        {"Java教程","http://c.biancheng.net/java/"},
        {"Linux教程","http://c.biancheng.net/linux/"} };
    //获取 "Java教程" 对应的值
    string str = umap["Java教程"];
    cout << str << endl;
    return 0;
}
```

程序输出结果为：

```
http://c.biancheng.net/java/
```

需要注意的是，如果当前容器中并没有存储以 [ ] 运算符内指定的元素作为键的键值对，则此时 [ ] 运算符的功能将转变为：向当前容器中添加以目标元素为键的键值对。举个例子：

```
#include <iostream>
#include <string>
#include <unordered_map>
using namespace std;
int main()
{
    //创建空 umap 容器
```

```
    unordered_map<string, string> umap;
    //[] 运算符在 = 右侧
    string str = umap["STL教程"];
    //[] 运算符在 = 左侧
    umap["C教程"] = "http://c.biancheng.net/c/";

    for (auto iter = umap.begin(); iter != umap.end(); ++iter) {
        cout << iter->first << " " << iter->second << endl;
    }
    return 0;
}
```

程序执行结果为：

```
C教程 http://c.biancheng.net/c/
STL教程
```

可以看到，当使用 [ ] 运算符向 unordered_map 容器中添加键值对时，分为 2 种情况：

1. 当 [ ] 运算符位于赋值号（=）右侧时，则新添加键值对的键为 [ ] 运算符内的元素，其值为键值对要求的值类型的默认值（string 类型默认值为空字符串）；
2. 当 [ ] 运算符位于赋值号（=）左侧时，则新添加键值对的键为 [ ] 运算符内的元素，其值为赋值号右侧的元素。

2) unordered_map 类模板中，还提供有 at() 成员方法，和使用 [ ] 运算符一样，at() 成员方法也需要根据指定的键，才能从容器中找到该键对应的值；不同之处在于，如果在当前容器中查找失败，该方法不会向容器中添加新的键值对，而是直接抛出 `out_of_range` 异常。

举个例子：

```
#include <iostream>
#include <string>
#include <unordered_map>
using namespace std;
int main()
{
    //创建 umap 容器
    unordered_map<string, string> umap{
        {"Python教程","http://c.biancheng.net/python/"},
        {"Java教程","http://c.biancheng.net/java/"},
        {"Linux教程","http://c.biancheng.net/linux/"} };
    //获取指定键对应的值
    string str = umap.at("Python教程");
```

```
14.     cout << str << endl;
15.
16.     //执行此语句会抛出 out_of_range 异常
17.     //cout << umap.at("GO教程");
18.     return 0;
19. }
```

程序执行结果为：

```
http://c.biancheng.net/python/
```

此程序中，第 13 行代码用于获取 umap 容器中键为“Python教程”对应的值，由于 umap 容器确实有符合条件的键值对，因此可以成功执行；而第 17 行代码，由于当前 umap 容器没有存储以“Go教程”为键的键值对，因此执行此语句会抛出 out_of_range 异常。

3) [ ] 运算符和 at() 成员方法基本能满足大多数场景的需要。除此之外，还可以借助 unordered_map 模板中提供的 find() 成员方法。

和前面方法不同的是，通过 find() 方法得到的是一个正向迭代器，该迭代器的指向分以下 2 种情况：

1. 当 find() 方法成功找到以指定元素作为键的键值对时，其返回的迭代器就指向该键值对；
2. 当 find() 方法查找失败时，其返回的迭代器和 end() 方法返回的迭代器一样，指向容器中最后一个键值对之后的位置。

举个例子：

```
01. #include <iostream>
02. #include <string>
03. #include <unordered_map>
04. using namespace std;
05. int main()
06. {
07.     //创建 umap 容器
08.     unordered_map<string, string> umap{
09.         {"Python教程","http://c.biancheng.net/python/"},
10.         {"Java教程","http://c.biancheng.net/java/"},
11.         {"Linux教程","http://c.biancheng.net/linux/"} };
12.     //查找成功
13.     unordered_map<string, string>::iterator iter = umap.find("Python教程");
14.     cout << iter->first << " " << iter->second << endl;
15.     //查找失败
16.     unordered_map<string, string>::iterator iter2 = umap.find("GO教程");
17.     if (iter2 == umap.end()) {
```

```
        cout << "当前容器中没有以\"GO教程\"为键的键值对";
    }
    return 0;
}
```

程序执行结果为：

```
Python教程 http://c.biancheng.net/python/
当前容器中没有以"GO教程"为键的键值对
```

4) 除了 find() 成员方法之外，甚至可以借助 begin()/end() 或者 cbegin()/cend()，通过遍历整个容器中的键值对来找到目标键值对。

举个例子：

```
#include <iostream>
#include <string>
#include <unordered_map>
using namespace std;
int main()
{
    //创建 umap 容器
    unordered_map<string, string> umap{
        {"Python教程","http://c.biancheng.net/python/"},
        {"Java教程","http://c.biancheng.net/java/"},
        {"Linux教程","http://c.biancheng.net/linux/"} };
    //遍历整个容器中存储的键值对
    for (auto iter = umap.begin(); iter != umap.end(); ++iter) {
        //判断当前的键值对是否就是要找的
        if (!iter->first.compare("Java教程")) {
            cout << iter->second << endl;
            break;
        }
    }
    return 0;
}
```

程序执行结果为：

```
http://c.biancheng.net/java/
```

> 以上 4 种方法中，前 2 种方法基本能满足多数场景的需要，建议初学者首选 at() 成员方法！