C++ 标准库 <unordered_map>

在 C++ 中，<unordered_map> 是标准模板库（STL）的一部分，提供了一种基于哈希表的键值对容器。

与 std::map 不同，unordered_map 不保证元素的排序，但通常提供更快的查找速度。

unordered_map 是一个关联容器，它存储了键值对（key-value pairs），其中每个键（key）都是唯一的。unordered_map 使用哈希表来存储元素，这使得它在查找、插入和删除操作中具有平均常数时间复杂度。

语法

以下是 unordered_map 的基本语法：

```
#include <unordered_map>

std::unordered_map<key_type, value_type> map_name;
```

key_type 是键的类型。

value_type 是值的类型。

构造函数

unordered_map 可以以多种方式构造：

```
// 默认构造
std::unordered_map<int, std::string> myMap;

// 构造并初始化
std::unordered_map<int, std::string> myMap = {{1, "one"}, {2, "two"}};

// 构造并指定初始容量
std::unordered_map<int, std::string> myMap(10);

// 构造并复制另一个 unordered_map
std::unordered_map<int, std::string> anotherMap = myMap;
```

基本操作

插入元素：

```
myMap.insert({3, "three"});
```

访问元素：

```
std::string value = myMap[1]; // 获取键为1的值
```

删除元素：

```
myMap.erase(1); // 删除键为1的元素
```

查找元素：

```
auto it = myMap.find(2); // 查找键为2的元素
if (it != myMap.end()) {
    std::cout << "Found: " << it->second << std::endl;
}
```

实例

下面是一个使用 unordered_map 的简单实例，包括输出结果。

实例

```
#include <iostream>
#include <unordered_map>

int main() {
    // 创建一个 unordered_map，键为 int，值为 string
    std::unordered_map<int, std::string> myMap;

    // 插入一些键值对
    myMap[1] = "one";
    myMap[2] = "two";
    myMap[3] = "three";

    // 打印所有元素
    for (const auto& pair : myMap) {
        std::cout << "Key: " << pair.first << ", Value: " << pair.second << std::endl;
    }

    // 访问特定键的值
    std::cout << "Value for key 2: " << myMap[2] << std::endl;

    // 删除键为1的元素
    myMap.erase(1);

    // 再次打印所有元素
    std::cout << "After erasing key 1:" << std::endl;
    for (const auto& pair : myMap) {
        std::cout << "Key: " << pair.first << ", Value: " << pair.second << std::endl;
    }

    return 0;
}
```

输出结果:

```
Key: 1, Value: one
Key: 2, Value: two
Key: 3, Value: three
Value for key 2: two
After erasing key 1:
Key: 2, Value: two
Key: 3, Value: three
```

注意事项

unordered_map 不保证元素的顺序，因此元素的迭代顺序可能在不同的运行中不同。

哈希表的性能依赖于良好的哈希函数，以避免过多的哈希冲突。

与 std::map 相比，unordered_map 在元素数量较少时可能占用更多的内存。