请为我解释unorder_map这个类怎么使用作为哈希表

使用 unordered_map 作为哈希表

unordered_map 是 C++ STL 中的一个关联容器，它提供基于哈希表的键值对存储。下面我将详细介绍如何使用它。

基本特性

基于哈希表实现

平均情况下插入、删除和查找操作的时间复杂度为 O(1)

元素无序存储（与 map 的有序存储不同）

需要包含头文件 <unordered_map>

基本用法

1. 创建 unordered_map

cpp

复制

下载

```cpp
#include <unordered_map>
#include <string>

std::unordered_map<std::string, int> wordCount;  // 键类型为string，值类型为int
```

2. 插入元素

cpp

复制

下载

```cpp
// 使用insert方法
wordCount.insert({"apple", 3});

// 使用下标操作符
wordCount["banana"] = 2;

// 使用emplace (C++11)
wordCount.emplace("orange", 5);
```

3. 访问元素

cpp

复制

下载

```cpp
// 使用下标操作符
int count = wordCount["apple"];  // 如果键不存在会创建新元素
```

```cpp
// 使用at方法（键不存在会抛出异常）
count = wordCount.at("apple");

// 查找元素
auto it = wordCount.find("apple");
if (it != wordCount.end()) {
    // 找到元素
    count = it->second;
}
```

4. 删除元素

cpp

复制

下载

```cpp
// 通过键删除
wordCount.erase("apple");

// 通过迭代器删除
auto it = wordCount.find("banana");
if (it != wordCount.end()) {
    wordCount.erase(it);
}

// 删除所有元素
wordCount.clear();
```

5. 遍历元素

cpp

复制

下载

```cpp
for (const auto& pair : wordCount) {
    std::cout << pair.first << ": " << pair.second << std::endl;
}

// 使用迭代器
for (auto it = wordCount.begin(); it != wordCount.end(); ++it) {
    std::cout << it->first << ": " << it->second << std::endl;
}
```

高级用法

1. 自定义哈希函数

cpp

复制

下载

```cpp
struct MyHash {
```

```
    size_t operator()(const std::string& key) const {
        // 简单的哈希函数示例
        return key.length();
    }
};

std::unordered_map<std::string, int, MyHash> customHashMap;
```

2. 自定义键比较函数

cpp

复制

下载

```
struct MyEqual {
    bool operator()(const std::string& lhs, const std::string& rhs) const {
        // 自定义比较逻辑
        return lhs[0] == rhs[0];  // 只比较首字母
    }
};

std::unordered_map<std::string, int, std::hash<std::string>, MyEqual> customEqua
```

3. 性能调优

cpp

复制

下载

```
// 设置桶的数量
wordCount.reserve(100);  // 预分配空间

// 获取负载因子
float loadFactor = wordCount.load_factor();

// 设置最大负载因子
wordCount.max_load_factor(0.7);  // 当负载因子超过0.7时会重新哈希
```

注意事项

下标操作符 [] 会在键不存在时插入新元素，而 at() 会抛出异常

对于自定义类型作为键，需要提供哈希函数和相等比较函数

频繁的插入删除可能导致重新哈希，影响性能

迭代器在插入操作后可能失效

unordered_map 是大多数情况下高性能的哈希表实现，但如果你需要有序遍历键，应该考虑使用 map。