`std::map` 是 C++ 标准库中的一种关联容器，用于存储键值对（key-value pairs）。它按照键的顺序自动排序，并且每个键是唯一的。`std::map` 基于红黑树（balanced binary search tree），提供高效的插入、删除、查找操作，时间复杂度通常为 O(log N)。

以下是 `std::map` 的标准用法总结：

1. 声明 `map`

`std::map` 的基本声明格式如下：

```
std::map<KeyType, ValueType> map_name;
```

其中：

- `KeyType`：键的类型。
- `ValueType`：值的类型。

例如：

```
std::map<int, std::string> myMap;
```

这个 `map` 使用 `int` 类型作为键，`std::string` 类型作为值。

2. 插入元素

直接插入：

可以使用 `insert` 或通过 `[]` 操作符插入键值对。

```
// 使用 insert 插入元素
myMap.insert({1, "one"});
myMap.insert(std::make_pair(2, "two"));

// 使用 [] 插入元素（如果键已存在，则不会更新值）
myMap[3] = "three";
```

注意：

- 使用 `[]` 插入时，如果键不存在，会自动插入该键并初始化值（默认为类型的默认值）。如果键已存在，则更新该键的值。
- 使用 `insert` 插入时，如果键已经存在，插入会失败。

使用 `insert_or_assign`（C++17 及以上）

如果你希望插入新键值对，或者在键已存在时更新值，可以使用 `insert_or_assign`：

```
myMap.insert_or_assign(2, "new_two"); // 如果键 2 存在，会更新为 "new_two"
```

3. 访问元素

可以通过 `[]` 操作符或者 `at()` 方法来访问 `map` 中的元素。

```
// 使用 [] 访问（如果键不存在会插入默认值）
std::cout << myMap[1] << std::endl;  // 输出 "one"

// 使用 at() 访问（如果键不存在会抛出异常）
try {
    std::cout << myMap.at(2) << std::endl;  // 输出 "two"
} catch (const std::out_of_range& e) {
    std::cerr << "Key not found!" << std::endl;
}
```

4. 查找元素

可以使用 `find()` 查找某个键是否存在。`find` 返回一个迭代器，如果键存在，指向该元素；否则，指向 `map` 的 `end()`。

```
auto it = myMap.find(2);
if (it != myMap.end()) {
    std::cout << "Found: " << it->second << std::endl;  // 输出 "Found: two"
} else {
    std::cout << "Key not found!" << std::endl;
}
```

5. 删除元素

可以使用 `erase()` 来删除元素。

- 使用键删除：

  ```
  myMap.erase(1);  // 删除键为 1 的元素
  ```

- 使用迭代器删除：

  ```
  auto it = myMap.find(2);
  if (it != myMap.end()) {
      myMap.erase(it);  // 删除找到的元素
  }
  ```

- 删除所有元素：

  ```
  myMap.clear();  // 清空所有元素
  ```

6. 遍历 `map`

可以使用范围 `for` 循环、迭代器或者传统的 `for` 循环遍历 `map`。

使用范围 `for` 循环：

```
for (const auto& [key, value] : myMap) {
    std::cout << key << ": " << value << std::endl;
}
```

使用迭代器：

```
for (auto it = myMap.begin(); it != myMap.end(); ++it) {
    std::cout << it->first << ": " << it->second << std::endl;
}
```

7. 检查 `map` 是否为空

可以使用 `empty()` 方法来检查 `map` 是否为空。

```
if (myMap.empty()) {
    std::cout << "Map is empty!" << std::endl;
} else {
    std::cout << "Map is not empty!" << std::endl;
}
```

8. 获取 `map` 的大小

可以使用 `size()` 方法来获取 `map` 中元素的数量。

```
std::cout << "Size of map: " << myMap.size() << std::endl;
```

9. 排序

`std::map` 会根据键自动排序，默认是升序排序（按键的 `<` 运算符）。如果需要自定义排序，可以传入一个比较函数：

```
// 自定义排序，按降序排列键
std::map<int, std::string, std::greater<int>> myMapDesc;
myMapDesc[1] = "one";
myMapDesc[3] = "three";
myMapDesc[2] = "two";
```

10. 其他常用函数

- `lower_bound()` 和 `upper_bound()`：用于查找范围。

  ```
  auto it1 = myMap.lower_bound(2);  // 第一个键大于或等于 2 的元素
  auto it2 = myMap.upper_bound(2);  // 第一个键大于 2 的元素
  ```

- `equal_range()`：返回一个键的范围。

```
auto range = myMap.equal_range(2);
// range.first 是指向第一个匹配元素的迭代器，range.second 是指向第一个大于该元素的迭代器
```

总结

- `std::map` 是一个关联容器，按键的顺序自动排序，键唯一。
- 支持插入、查找、删除等基本操作。
- 使用 `insert`、`at`、`[]` 等方法来操作元素。
- 使用迭代器或者范围 `for` 循环遍历元素。
- 提供了高效的查找和插入操作，时间复杂度为 O(log N)。

这种容器在需要通过键快速查找值，且需要保持键的有序性时非常有用。