`std::set` 是 C++ 标准库中的一个关联容器，位于 `<set>` 头文件中。它实现了一个 **有序集合**，其中的元素是唯一的，并且按照一定的顺序进行排列。通常，`std::set` 使用 **红黑树**（或其他平衡树结构）来实现，因此其大多数操作的时间复杂度为 O(log n)。

常见用法

1. 声明和初始化

你可以使用默认构造函数或通过初始化列表、范围构造函数来创建 `std::set`。

```
#include <iostream>
#include <set>

int main() {
    // 默认构造函数，创建一个空的 set
    std::set<int> s;

    // 使用初始化列表构造 set
    std::set<int> s2 = {1, 3, 5, 7};

    // 使用范围构造函数
    std::set<int> s3(s2.begin(), s2.end());

    // 打印 set
    for (const auto& item : s2) {
        std::cout << item << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

2. 插入元素

使用 `insert` 方法向 `set` 中插入元素。如果插入的元素已经存在，`set` 不会插入重复的元素。

```
#include <iostream>
#include <set>

int main() {
    std::set<int> s = {1, 2, 3};

    // 插入新元素
    s.insert(4);
    s.insert(2);  // 不会插入重复的 2

    // 插入的元素自动排序
    for (const auto& item : s) {
        std::cout << item << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

3. 删除元素

你可以使用 `erase` 删除 `set` 中的元素。删除可以通过元素值、迭代器或范围来指定。

- `erase(key)`：根据值删除元素。
- `erase(iterator)`：根据迭代器删除元素。
- `erase(first, last)`：删除指定范围的元素。

```
#include <iostream>
#include <set>

int main() {
    std::set<int> s = {1, 2, 3, 4, 5};

    // 删除单个元素
    s.erase(3);  // 删除值为 3 的元素

    // 删除指定位置的元素
    auto it = s.find(4);
    if (it != s.end()) {
        s.erase(it);  // 删除迭代器 it 指向的元素
    }

    // 删除范围内的元素
    auto first = s.find(1);
    auto last = s.find(5);
    s.erase(first, last);  // 删除 1 到 5 范围内的元素

    // 打印结果
    for (const auto& item : s) {
        std::cout << item << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

4. 查找元素

`std::set` 提供了 `find` 方法来查找元素。`find` 返回一个指向元素的迭代器，如果元素不存在，则返回 `end()`。

```
#include <iostream>
#include <set>

int main() {
    std::set<int> s = {1, 2, 3, 4, 5};

    // 查找元素
    auto it = s.find(3);
    if (it != s.end()) {
        std::cout << "Found: " << *it << std::endl;
    } else {
        std::cout << "Not found" << std::endl;
```

```
    }

    return 0;
}
```

5. 遍历元素

你可以使用基于范围的 `for` 循环或者传统的迭代器来遍历 `set`。

```
#include <iostream>
#include <set>

int main() {
    std::set<int> s = {1, 2, 3, 4, 5};

    // 使用基于范围的 for 循环
    for (const auto& item : s) {
        std::cout << item << " ";
    }
    std::cout << std::endl;

    // 使用迭代器遍历
    for (auto it = s.begin(); it != s.end(); ++it) {
        std::cout << *it << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

6. 检查元素是否存在

`std::set` 提供了 `count` 方法来检查元素是否存在。对于 `set`，该方法要么返回 0（元素不存在），要么返回 1（元素存在）。

```
#include <iostream>
#include <set>

int main() {
    std::set<int> s = {1, 2, 3, 4, 5};

    if (s.count(3)) {
        std::cout << "Element 3 exists in the set" << std::endl;
    } else {
        std::cout << "Element 3 does not exist in the set" << std::endl;
    }

    return 0;
}
```

7. 大小和空检查

你可以使用 `size` 和 `empty` 来获取 `set` 的大小并检查是否为空。

```
#include <iostream>
#include <set>

int main() {
    std::set<int> s = {1, 2, 3};

    std::cout << "Size of set: " << s.size() << std::endl;
    std::cout << "Is the set empty? " << (s.empty() ? "Yes" : "No") << std::endl;

    return 0;
}
```

8. 自定义排序规则

默认情况下，`std::set` 按照元素的升序排列（使用 `<` 比较）。你也可以通过提供自定义的比较器来改变排序方式，例如按降序排列或按自定义逻辑排序。

```
#include <iostream>
#include <set>

struct Descending {
    bool operator()(int a, int b) const {
        return a > b;  // 降序排列
    }
};

int main() {
    // 使用自定义比较器 Descending
    std::set<int, Descending> s = {1, 2, 3, 4, 5};

    // 打印结果（降序）
    for (const auto& item : s) {
        std::cout << item << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

9. 清空 `set`

你可以使用 `clear` 方法清空 `set` 中的所有元素。

```cpp
#include <iostream>
#include <set>

int main() {
    std::set<int> s = {1, 2, 3, 4, 5};

    // 清空 set
    s.clear();

    std::cout << "Size after clear: " << s.size() << std::endl;  // 应该是 0

    return 0;
}
```

总结

`std::set` 是一个非常有用的关联容器，它提供了以下常见操作：

- **插入元素**：使用 `insert` 插入元素，自动按升序排序。
- **删除元素**：使用 `erase` 删除元素，支持通过值、迭代器或范围删除。
- **查找元素**：使用 `find` 查找元素，返回迭代器。
- **遍历元素**：可以使用迭代器或基于范围的 `for` 循环进行遍历。
- **大小与空检查**：使用 `size` 和 `empty` 检查容器的状态。
- **自定义排序**：可以通过自定义比较器来改变排序规则。
- **检查元素存在性**：通过 `count` 方法检查元素是否存在。

由于 `set` 保证元素唯一且按照一定顺序排列，它非常适合用于处理需要自动排序且无重复元素的场景。