# STL拓展：常见STL容器及其数据结构

## ①常见STL容器及其数据结构

C++ 标准模板库（STL）提供了多种容器，每种容器都有其特定的用途和实现方式。以下是一些常见的 STL 容器及其底层数据结构：

### 1. `std::vector`

- **数据结构**：动态数组（Dynamic Array）
- **特点**：
  - 支持快速随机访问，时间复杂度为 O(1)。
  - 插入和删除操作在末尾是 O(1)，在中间或开头是 O(n)。
  - 动态调整大小，必要时会重新分配内存并复制元素。
- **用途**：适用于需要频繁随机访问和在末尾插入元素的场景。

### 2. `std::deque`

- **数据结构**：双端队列（Double-Ended Queue），通常实现为一组连续块（blocks）组成的双向链表。
- **特点**：
  - 支持快速随机访问，时间复杂度为 O(1)。
  - 在两端插入和删除元素的操作是 O(1)。
  - 在中间插入和删除元素的操作是 O(n)。
- **用途**：适用于需要在两端频繁插入和删除元素的场景。

### 3. `std::list`

- **数据结构**：双向链表（Doubly Linked List）
- **特点**：
  - 不支持随机访问，访问元素的时间复杂度为 O(n)。
  - 在任意位置插入和删除元素的操作是 O(1)。
- **用途**：适用于需要频繁在中间插入和删除元素的场景。

### 4. `std::forward_list`

- **数据结构**：单向链表（Singly Linked List）
- **特点**：
  - 不支持随机访问，访问元素的时间复杂度为 O(n)。
  - 在开头插入和删除元素的操作是 O(1)。
  - 比 `std::list` 更节省内存，因为没有存储前向指针。
- **用途**：适用于需要在开头频繁插入和删除元素的场景。

## 5. `std::set` 和 `std::multiset`

- **数据结构**：红黑树（Red-Black Tree）
- **特点**：
  - 元素自动排序，支持有序遍历。
  - 查找、插入和删除元素的操作时间复杂度为 O(log n)。
  - `std::set` 不允许重复元素，`std::multiset` 允许重复元素。
- **用途**：适用于需要有序存储和快速查找的场景。

## 6. `std::map` 和 `std::multimap`

- **数据结构**：红黑树（Red-Black Tree）
- **特点**：
  - 存储键值对，键自动排序，支持有序遍历。
  - 查找、插入和删除元素的操作时间复杂度为 O(log n)。
  - `std::map` 不允许重复键，`std::multimap` 允许重复键。
- **用途**：适用于需要有序存储键值对和快速查找的场景。

## 7. `std::unordered_map` 和 `std::unordered_multimap`

- **数据结构**：哈希表（Hash Table）
- **特点**：
  - 存储键值对，不保证元素顺序。
  - 查找、插入和删除元素的操作平均时间复杂度为 O(1)。
  - `std::unordered_map` 不允许重复键，`std::unordered_multimap` 允许重复键。
- **用途**：适用于需要快速查找和插入键值对的场景。

## 8. `std::unordered_set` 和 `std::unordered_multiset`

- **数据结构**：哈希表（Hash Table）
- **特点**：
  - 存储唯一元素，不保证元素顺序。
  - 查找、插入和删除元素的操作平均时间复杂度为 O(1)。
  - `std::unordered_set` 不允许重复元素，`std::unordered_multiset` 允许重复元素。
- **用途**：适用于需要快速查找和插入唯一元素的场景。

## 9. `std::stack`

- **数据结构**：基于 `std::deque` 或其他容器实现的栈（Stack）
- **特点**：
  - 后进先出（LIFO）数据结构。
  - 只允许在栈顶插入和删除元素，操作时间复杂度为 O(1)。
- **用途**：适用于需要后进先出的数据处理场景。

### 10. `std::queue`

- **数据结构**：基于 `std::deque` 或其他容器实现的队列（Queue）
- **特点**：
  - 先进先出（FIFO）数据结构。
  - 只允许在队列的两端插入和删除元素，操作时间复杂度为 O(1)。
- **用途**：适用于需要先进先出的数据处理场景。

### 11. `std::priority_queue`

- **数据结构**：基于堆（Heap）实现的优先队列（Priority Queue）
- **特点**：
  - 元素按照优先级排序，最大（或最小）元素总是位于队列的顶端。
  - 插入和删除操作的时间复杂度为 O(log n)。
- **用途**：适用于需要按优先级处理数据的场景。

### 总结

- **数组和链表**：`std::vector`（动态数组）、`std::deque`（双端队列）、`std::list`（双向链表）、`std::forward_list`（单向链表）
- **树结构**：`std::set`、`std::multiset`、`std::map`、`std::multimap`（红黑树）
- **哈希表**：`std::unordered_map`、`std::unordered_multimap`、`std::unordered_set`、`std::unordered_multiset`
- **栈和队列**：`std::stack`、`std::queue`、`std::priority_queue`

不同的容器适用于不同的场景，选择合适的容器可以显著提高程序的性能和可读性。

## ②补充：STL容器在Python中的对应关系

在 Python 中，也有与 C++ STL 容器类似的内置数据结构和模块。以下是一些对应的函数或容器：

### 1. `std::vector`

- **Python 对应**：`list`
- **特点**：动态数组，支持快速随机访问和在末尾插入元素。

### 2. `std::deque`

- **Python 对应**：`collections.deque`
- **特点**：双端队列，支持在两端快速插入和删除元素。

## 3. `std::list` 和 `std::forward_list`

- **Python 对应**：Python 没有直接的链表实现，但可以使用 `list` 和 `collections.deque` 来模拟。
- **特点**：`list` 实现为动态数组，`deque` 可以用于类似链表的操作。

## 4. `std::set` 和 `std::multiset`

- **Python 对应**：`set` 和 `collections.Counter`
- **特点**：`set` 用于存储唯一元素，`Counter` 可用于统计元素频率，类似 `multiset`。

## 5. `std::map` 和 `std::multimap`

- **Python 对应**：`dict` 和 `collections.defaultdict`
- **特点**：`dict` 用于存储键值对，`defaultdict` 可用于处理多值映射。

## 6. `std::unordered_map` 和 `std::unordered_multimap`

- **Python 对应**：`dict` 和 `collections.defaultdict`
- **特点**：Python 的 `dict` 本质上是哈希表，支持快速查找。

## 7. `std::unordered_set` 和 `std::unordered_multiset`

- **Python 对应**：`set` 和 `collections.Counter`
- **特点**：`set` 用于存储唯一元素，`Counter` 可用于统计元素频率。

## 8. `std::stack`

- **Python 对应**：`list` 或 `collections.deque`
- **特点**：可以用 `list` 的 `append` 和 `pop` 方法实现栈的操作。

## 9. `std::queue`

- **Python 对应**：`collections.deque`
- **特点**：使用 `deque` 的 `append` 和 `popleft` 方法实现队列操作。

## 10. `std::priority_queue`

- **Python 对应**：`heapq` 模块
- **特点**：提供堆操作，支持优先队列功能。

## 总结

- **列表和数组**：`list`
- **双端队列**：`collections.deque`
- **集合和多重集合**：`set` 和 `collections.Counter`
- **字典和多重字典**：`dict` 和 `collections.defaultdict`
- **栈和队列**：使用 `list` 和 `collections.deque`

- **优先队列**：`heapq`

Python 提供了灵活的内置数据结构和模块，可以满足大多数数据处理需求。