

在 C++ 中，`std::vector` 是一个非常常用的动态数组容器，位于 `<vector>` 头文件中。它提供了对元素的高效访问和操作，自动管理内存，并允许动态扩展其大小。下面是 C++ 中 `std::vector` 的常见标准用法总结：

1. 创建和初始化 `std::vector`

(1) 默认构造

```
#include <vector>

std::vector<int> vec1; // 创建一个空的整数 vector
```

(2) 使用初始化列表

```
std::vector<int> vec2 = {1, 2, 3, 4}; // 使用列表初始化
```

(3) 指定大小和初始值

```
std::vector<int> vec3(5, 10); // 创建一个包含 5 个元素的 vector，每个元素的值为 10
```

(4) 使用其他 vector 初始化

```
std::vector<int> vec4(vec2); // 使用另一个 vector 进行初始化，复制 vec2 的元素
```

2. 常用成员函数

(1) 获取大小

```
std::vector<int> vec = {1, 2, 3};
std::cout << "Size: " << vec.size() << std::endl; // 获取 vector 的大小
std::cout << "Capacity: " << vec.capacity() << std::endl; // 获取 vector 的容量
```

(2) 访问元素

使用下标操作符 `[]`

```
std::cout << "First element: " << vec[0] << std::endl; // 使用下标访问元素
```

使用 `at()` 方法

```
std::cout << "Second element: " << vec.at(1) << std::endl; // 使用 at() 方法访问，带边界检查
```

获取首元素和尾元素

```
std::cout << "First element: " << vec.front() << std::endl; // 获取首元素
std::cout << "Last element: " << vec.back() << std::endl; // 获取尾元素
```

(3) 添加元素

push_back()：向 vector 末尾添加元素

```
vec.push_back(4); // 在 vector 末尾添加元素
```

emplace_back()：直接在末尾构造元素（性能更优）

```
vec.emplace_back(5); // 直接在 vector 末尾构造一个元素
```

insert()：在指定位置插入元素

```
vec.insert(vec.begin() + 1, 100); // 在索引 1 位置插入元素 100
```

(4) 删除元素

pop_back()：删除最后一个元素

```
vec.pop_back(); // 删除 vector 末尾的元素
```

erase()：删除指定位置的元素

```
vec.erase(vec.begin() + 2); // 删除索引 2 处的元素
```

clear()：清空所有元素

```
vec.clear(); // 删除所有元素
```

(5) 修改元素

```
vec[0] = 10; // 修改指定位置的元素
```

(6) 交换内容

```
std::vector<int> vec2 = {7, 8, 9};
vec.swap(vec2); // 交换 vec 和 vec2 的内容
```

3. 迭代器

`std::vector` 支持使用迭代器来遍历其元素。

```
std::vector<int> vec = {1, 2, 3, 4};
for (auto it = vec.begin(); it != vec.end(); ++it) {
    std::cout << *it << " "; // 使用迭代器访问元素
}
std::cout << std::endl;
```

或者使用范围基 for 循环：

```
for (const auto& element : vec) {
    std::cout << element << " "; // 直接遍历元素
}
std::cout << std::endl;
```

4. 容量相关操作

(1) 容量

```
std::vector<int> vec = {1, 2, 3};
std::cout << "Size: " << vec.size() << std::endl; // 当前元素个数
std::cout << "Capacity: " << vec.capacity() << std::endl; // 当前分配的内存容量
```

(2) 扩容

`vector` 在元素增加时，如果超出当前容量，它会自动扩展其容量，通常是原容量的两倍。

(3) 保持容量不变


```
vec.shrink_to_fit(); // 减少不必要的内存分配，调整为当前大小
```

5. 常见操作

(1) 排序

使用 `std::sort()` 对 `vector` 进行排序（需要 `#include <algorithm>`）：

```
#include <algorithm>

std::vector<int> vec = {4, 1, 3, 2};
std::sort(vec.begin(), vec.end());  从小到大排序
```

(2) 查找

使用 `std::find()` 查找元素（需要 `#include <algorithm>`）：

```
auto it = std::find(vec.begin(), vec.end(), 3);
if (it != vec.end()) {
    std::cout << "Found element: " << *it << std::endl;
} else {
    std::cout << "Element not found!" << std::endl;
}
```

(3) 反转

使用 `std::reverse()` 反转 `vector` 中的元素（需要 `#include <algorithm>`）：

```
std::reverse(vec.begin(), vec.end()); // 反转 vector
```

(4) 转换为其他容器

可以将 `vector` 转换为其他 STL 容器，如 `std::list`：

```
#include <list>
std::list<int> lst(vec.begin(), vec.end()); // 使用 vector 初始化 list
```

6. 高效操作

- `emplace_back()` 优于 `push_back()`：当需要插入复杂类型的对象时，`emplace_back()` 可以直接在容器末尾构造元素，避免了不必要的拷贝或移动。
- **预分配空间**：通过 `reserve()` 方法预分配足够的空间，避免在插入元素时频繁重新分配内存。

```
std::vector<int> vec;
vec.reserve(1000); // 预分配空间，避免多次扩容
```

7. 示例代码

```
#include <iostream>
#include <vector>
#include <algorithm>

int main() {
    // 创建一个包含初始元素的 vector
    std::vector<int> vec = {3, 1, 4, 1, 5, 9};

    // 输出原始元素
    std::cout << "Original vector: ";
    for (const auto& val : vec) {
        std::cout << val << " ";
    }
    std::cout << std::endl;

    // 排序
    std::sort(vec.begin(), vec.end());

    // 输出排序后的元素
    std::cout << "Sorted vector: ";
    for (const auto& val : vec) {
        std::cout << val << " ";
    }
    std::cout << std::endl;

    // 查找元素
    auto it = std::find(vec.begin(), vec.end(), 4);
    if (it != vec.end()) {
```

```
        std::cout << "Found element: " << *it << std::endl;
    }

    // 删除元素
    vec.erase(it); // 删除找到的元素
    std::cout << "After erase: ";
    for (const auto& val : vec) {
        std::cout << val << " ";
    }
    std::cout << std::endl;

    // 添加新元素
    vec.push_back(7);
    std::cout << "After push_back: ";
    for (const auto& val : vec) {
        std::cout << val << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

总结

`std::vector` 是一个灵活且高效的容器，适用于大多数需要动态数组的场景。它提供了动态大小调整、高效的访问和修改操作。通过合理使用 `push_back()`、`emplace_back()`、`reserve()`、`erase()` 等函数，可以极大地提高代码的性能和可读性。