vector类型标准库函数

1. 构造函数与赋值

- 构造函数：

```
vector<int> v1;          // 空 vector
vector<int> v2(5, 10);     // 5 个元素，每个值为 10
vector<int> v3 = {1, 2, 3}; // 初始化列表（C++11）
vector<int> v4(v3);        // 拷贝构造
```

- 赋值 `=` 和 `assign()`：

```
v1 = v2;                  // 复制 v2 的内容
v1.assign(3, 100);        // 替换内容为 3 个 100
v1.assign(v3.begin(), v3.end()); // 用迭代器范围赋值
```

2. 元素访问

- 下标访问 `[]` 和 `at()`：

```
int a = v1[0];          // 不检查越界（高效）
int b = v1.at(0);        // 越界时抛出 `std::out_of_range`
```

- 首尾元素 `front()` 和 `back()`：

```
int first = v1.front(); // 第一个元素
int last = v1.back();    // 最后一个元素
```

- 底层数组指针 `data()`：

```
int* ptr = v1.data();    // 返回指向底层数组的指针
```

3. 容量操作

- 大小与容量：

```
v1.size();            // 当前元素个数
v1.empty();           // 是否为空
v1.capacity();        // 当前分配的存储空间大小
```

- 调整容量 `reserve()` 和 `resize()`：

```
v1.reserve(100);       // 预分配至少 100 个元素的空间（不改变 size）
v1.resize(10, 5);      // 调整 size 为 10，新增元素初始化为 5
```

- 释放未使用空间 `shrink_to_fit()` (C++11)：

```
v1.shrink_to_fit();   // 减少 capacity 到与 size 匹配（请求，非强制）
```

4. 修改操作

- 添加元素 `push_back()` 和 `emplace_back()` (C++11)：

```
v1.push_back(42);         // 在末尾插入 42（拷贝或移动）
v1.emplace_back(42);     // 直接在末尾构造元素（更高效）
```

- 插入元素 `insert()`：

```
auto it = v1.begin() + 2;
v1.insert(it, 99);         // 在位置 2 插入 99
v1.insert(it, 3, 88);     // 插入 3 个 88
```

- 删除元素 `pop_back()` 和 `erase()`：

```
v1.pop_back();             // 删除最后一个元素
v1.erase(v1.begin());     // 删除第一个元素
v1.erase(v1.begin(), v1.begin() + 3); // 删除前 3 个元素
```

- 清空 `clear()`：

```
v1.clear();               // 清空所有元素（size=0，capacity 不变）
```

- 交换 `swap()`：

```
v1.swap(v2);              // 交换两个 vector 的内容
```

5. 迭代器

- 迭代器访问:

```
for (auto it = v1.begin(); it != v1.end(); ++it) {
    cout << *it << " ";
}
```

- 反向迭代器:

```
for (auto rit = v1.rbegin(); rit != v1.rend(); ++rit) {
    cout << *rit << " ";
}
```

6. 其他操作

- 比较运算符（`==`，`!=`，`<`，`>` 等）：

```
if (v1 == v2) { ... }    // 按顺序比较元素
```

- 移动语义（C++11）：

```
vector<int> v4 = std::move(v1); // 移动构造（v1 变为空）
```

示例代码

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> vec = {1, 2, 3, 4, 5};

    vec.push_back(6);           // 添加元素
    vec.emplace_back(7);        // 高效添加

    vec.insert(vec.begin() + 2, 99); // 在位置 2 插入 99

    cout << "Size: " << vec.size() << endl; // 输出 7
    cout << "Capacity: " << vec.capacity() << endl;

    vec.pop_back();             // 删除最后一个元素

    for (int num : vec) {       // 范围循环（C++11）
        cout << num << " ";     // 输出 1 2 99 3 4 5 6
    }

    return 0;
}
```

关键注意事项

1. 迭代器失效：插入或删除元素可能导致迭代器、指针或引用失效。
2. 预分配空间：频繁插入时使用 `reserve()` 减少重新分配次数。
3. 移动语义：大对象优先用 `emplace_back` 或 `std::move` 避免拷贝。
4. 越界访问：`[]` 不检查越界，`at()` 安全但性能略低。