C++ STL vector删除元素的几种方式（超级详细）

前面提到，无论是向现有 vector 容器中访问元素、添加元素还是插入元素，都只能借助 vector 模板类提供的成员函数，但删除 vector 容器的元素例外，完成此操作除了可以借助本身提供的成员函数，还可以借助一些全局函数。

基于不同场景的需要，删除 vecotr 容器的元素，可以使用表 1 中所示的函数（或者函数组合）。

表 1 删除 vector 容器元素的几种方式

函数	功能
pop_back()	删除 vector 容器中最后一个元素，该容器的大小（size）会减 1，但容量（capacity）不会发生改变。
erase(pos)	删除 vector 容器中 pos 迭代器指定位置处的元素，并返回指向被删除元素下一个位置元素的迭代器。该容器的大小（size）会减 1，但容量（capacity）不会发生改变。
swap(beg)、pop_back()	先调用 swap() 函数交换要删除的目标元素和容器最后一个元素的位置，然后使用 pop_back() 删除该目标元素。
erase(beg,end)	删除 vector 容器中位于迭代器 [beg,end)指定区域内的所有元素，并返回指向被删除区域下一个位置元素的迭代器。该容器的大小（size）会减小，但容量（capacity）不会发生改变。
remove()	删除容器中所有和指定元素值相等的元素，并返回指向最后一个元素下一个位置的迭代器。值得一提的是，调用该函数不会改变容器的大小和容量。
clear()	删除 vector 容器中所有的元素，使其变成空的 vector 容器。该函数会改变 vector 的大小（变为 0），但不是改变其容量。

下面就表 1 中罗列的这些函数，一一讲解它们的具体用法。

pop_back() 成员函数的用法非常简单，它不需要传入任何的参数，也没有返回值。举个例子：

```
01.  #include <vector>
02.  #include <iostream>
03.  using namespace std;
04.
05.  int main()
06.  {
07.      vector<int>demo{ 1,2,3,4,5 };
08.      demo.pop_back();
```

```
    //输出 dmeo 容器新的size
    cout << "size is :" << demo.size() << endl;
    //输出 demo 容器新的容量
    cout << "capacity is :" << demo.capacity() << endl;
    for (int i = 0; i < demo.size(); i++) {
        cout << demo[i] << " ";
    }
    return 0;
}
```

运行结果为：

```
size is :4
capacity is :5
1 2 3 4
```

可以发现，相比原 demo 容器，新的 demo 容器删除了最后一个元素 5，容器的大小减了 1，但容量没变。

如果想删除 vector 容器中指定位置处的元素，可以使用 erase() 成员函数，该函数的语法格式为：

```
iterator erase (pos);
```

其中，pos 为指定被删除元素位置的迭代器，同时该函数会返回一个指向删除元素所在位置下一个位置的迭代器。

下面的例子演示了 erase() 函数的具体用法：

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

int main()
{
    vector<int>demo{ 1,2,3,4,5 };
    auto iter = demo.erase(demo.begin() + 1);//删除元素 2
    //输出 dmeo 容器新的size
    cout << "size is :" << demo.size() << endl;
    //输出 demo 容器新的容量
    cout << "capacity is :" << demo.capacity() << endl;
    for (int i = 0; i < demo.size(); i++) {
        cout << demo[i] << " ";
    }
```

```
    //iter迭代器指向元素 3
    cout << endl << *iter << endl;
    return 0;
}
```

运行结果为：

```
size is :4
capacity is :5
1 3 4 5
3
```

通过结果不能看出，erase() 函数在删除元素时，会将删除位置后续的元素陆续前移，并将容器的大小减 1。

另外，如果不在意容器中元素的排列顺序，可以结合 swap() 和 pop_back() 函数，同样可以实现删除容器中指定位置元素的目的。

> 注意，swap() 函数在头文件 <algorithm> 和 <utility> 中都有定义，使用时引入其中一个即可。

例如：

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

int main()
{
    vector<int>demo{ 1,2,3,4,5 };
    //交换要删除元素和最后一个元素的位置
    swap(*(std::begin(demo)+1),*(std::end(demo)-1));//等同于 swap(demo[1],demo[4])

    //交换位置后的demo容器
    for (int i = 0; i < demo.size(); i++) {
        cout << demo[i] << " ";
    }
    demo.pop_back();
    cout << endl << "size is :" << demo.size() << endl;
    cout << "capacity is :" << demo.capacity() << endl;
    //输出demo 容器中剩余的元素
    for (int i = 0; i < demo.size(); i++) {
        cout << demo[i] << " ";
```

```
    }
    return 0;
}
```

运行结果为：

```
1 5 3 4 2
size is :4
capacity is :5
1 5 3 4
```

当然，除了删除容器中单个元素，还可以删除容器中某个指定区域内的所有元素，同样可以使用 erase() 成员函数实现。该函数有 2 种基本格式，前面介绍了一种，这里使用另一种：

```
iterator erase (iterator first, iterator last);
```

其中 first 和 last 是指定被删除元素区域的迭代器，同时该函数会返回指向此区域之后一个位置的迭代器。

举个例子：

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

int main()
{
    std::vector<int> demo{ 1,2,3,4,5 };
    //删除 2、3
    auto iter = demo.erase(demo.begin()+1, demo.end() - 2);
    cout << "size is :" << demo.size() << endl;
    cout << "capacity is :" << demo.capacity() << endl;

    for (int i = 0; i < demo.size(); i++) {
        cout << demo[i] << " ";
    }
    return 0;
}
```

运行结果为：

```
size is :3
capacity is :5
1 4 5
```

可以看到，和删除单个元素一样，删除指定区域内的元素时，也会将该区域后续的元素前移，并缩小容器的大小。

如果要删除容器中和指定元素值相同的所有元素，可以使用 remove() 函数，该函数定义在 <algorithm> 头文件中。例如：

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

int main()
{
    vector<int>demo{ 1,3,3,4,3,5 };
    //交换要删除元素和最后一个元素的位置
    auto iter = std::remove(demo.begin(), demo.end(), 3);

    cout << "size is :" << demo.size() << endl;
    cout << "capacity is :" << demo.capacity() << endl;
    //输出剩余的元素
    for (auto first = demo.begin(); first < iter;++first) {
        cout << *first << " ";
    }
    return 0;
}
```

运行结果为：

```
size is :6
capacity is :6
1 4 5
```

注意，在对容器执行完 remove() 函数之后，由于该函数并没有改变容器原来的大小和容量，因此无法使用之前的方法遍历容器，而是需要向程序中那样，借助 remove() 返回的迭代器完成正确的遍历。

> remove() 的实现原理是，在遍历容器中的元素时，一旦遇到目标元素，就做上标记，然后继续遍历，直到找到一个非目标元素，即用此元素将最先做标记的位置覆盖掉，同时将此非目标元素所在的位置也做上标记，等待找到新的非目标元素将其覆盖。因此，如果将上面程序中 demo 容器的元素全部输出，得到的结果为 `1 4 5 4 3 5` 。

另外还可以看到，既然通过 remove() 函数删除掉 demo 容器中的多个指定元素，该容器的大小和容量都没有改变，其剩余位置还保留了之前存储的元素。我们可以使用 erase() 成员函数删掉这些 "无用" 的元素。

比如，修改上面的程序：

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

int main()
{
    vector<int>demo{ 1,3,3,4,3,5 };
    //交换要删除元素和最后一个元素的位置
    auto iter = std::remove(demo.begin(), demo.end(), 3);
    demo.erase(iter, demo.end());
    cout << "size is :" << demo.size() << endl;
    cout << "capacity is :" << demo.capacity() << endl;
    //输出剩余的元素
    for (int i = 0; i < demo.size();i++) {
        cout << demo[i] << " ";
    }
    return 0;
}
```

运行结果为：

```
size is :3
capacity is :6
1 4 5
```

> remove()用于删除容器中指定元素时，常和 erase() 成员函数搭配使用。

如果想删除容器中所有的元素，则可以使用 clear() 成员函数，例如：

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

```

```
int main()
{
    vector<int>demo{ 1,3,3,4,3,5 };
    //交换要删除元素和最后一个元素的位置
    demo.clear();
    cout << "size is :" << demo.size() << endl;
    cout << "capacity is :" << demo.capacity() << endl;
    return 0;
}
```

运行结果为：

```
size is :0
capacity is :6
```