C++ STL deque容器添加和删除元素方法完全攻略

deque 容器中，无论是添加元素还是删除元素，都只能借助 deque 模板类提供的成员函数。表 1 中罗列的是所有和添加或删除容器内元素相关的 deque 模板类中的成员函数。

表 1 和添加或删除deque容器中元素相关的成员函数

成员函数	功能
push_back()	在容器现有元素的尾部添加一个元素，和 emplace_back() 不同，该函数添加新元素的过程是，先构造元素，然后再将该元素移动或复制到容器的尾部。
pop_back()	移除容器尾部的一个元素。
push_front()	在容器现有元素的头部添加一个元素，和 emplace_back() 不同，该函数添加新元素的过程是，先构造元素，然后再将该元素移动或复制到容器的头部。
pop_front()	移除容器尾部的一个元素。
emplace_back()	C++ 11 新添加的成员函数，其功能是在容器尾部生成一个元素。和 push_back() 不同，该函数直接在容器头部构造元素，省去了复制或移动元素的过程。
emplace_front()	C++ 11 新添加的成员函数，其功能是在容器头部生成一个元素。和 push_front() 不同，该函数直接在容器头部构造元素，省去了复制或移动元素的过程。
insert()	在指定的位置直接生成一个元素。和 emplace() 不同的是，该函数添加新元素的过程是，先构造元素，然后再将该元素移动或复制到容器的指定位置。
emplace()	C++ 11 新添加的成员函数，其功能是 insert() 相同，即在指定的位置直接生成一个元素。和 insert() 不同的是，emplace() 直接在容器指定位置构造元素，省去了复制或移动元素的过程。
erase()	移除一个元素或某一区域内的多个元素。
clear()	删除容器中所有的元素。

> 在实际应用中，常用 emplace()、emplace_front() 和 emplace_back() 分别代替 insert()、push_front() 和 push_back()，具体原因本节后续会讲。

以上这些成员函数中，除了 insert() 函数的语法格式比较多，其他函数都只有一种用法（erase() 有 2 种语法格式），下面这段程序演示了它们的具体用法：

```
01.  #include <deque>
02.  #include <iostream>
03.  using namespace std;
04.  int main()
```

```
{
    deque<int>d;
    //调用push_back()向容器尾部添加数据。
    d.push_back(2); //{2}
    //调用pop_back()移除容器尾部的一个数据。
    d.pop_back(); //{}

    //调用push_front()向容器头部添加数据。
    d.push_front(2);//{2}
    //调用pop_front()移除容器头部的一个数据。
    d.pop_front();//{}

    //调用 emplace 系列函数，向容器中直接生成数据。
    d.emplace_back(2); //{2}
    d.emplace_front(3); //{3,2}
    //emplace() 需要 2 个参数，第一个为指定插入位置的迭代器，第二个是插入的值。
    d.emplace(d.begin() + 1, 4);//{3,4,2}
    for (auto i : d) {
        cout << i << " ";
    }
    //erase()可以接受一个迭代器表示要删除元素所在位置
    //也可以接受 2 个迭代器，表示要删除元素所在的区域。
    d.erase(d.begin());//{4,2}
    d.erase(d.begin(), d.end());//{}，等同于 d.clear()
    return 0;
}
```

运行结果为：

```
3 4 2
```

这里重点讲一下 insert() 函数的用法。insert() 函数的功能是在 deque 容器的指定位置插入一个或多个元素。该函数的语法格式有多种，如表 2 所示。

表 2 insert() 成员函数语法格式

语法格式	功能
iterator insert(pos,elem)	在迭代器 pos 指定的位置之前插入一个新元素elem，并返回表示新插入元素位置的迭代器。
iterator insert(pos,n,elem)	在迭代器 pos 指定的位置之前插入 n 个元素 elem，并返回表示第一个新插入元素位置的迭代器。

iterator insert(pos,first,last)	在迭代器 pos 指定的位置之前，插入其他容器（不仅限于vector）中位于 [first,last) 区域的所有元素，并返回表示第一个新插入元素位置的迭代器。
iterator insert(pos,initlist)	在迭代器 pos 指定的位置之前，插入初始化列表（用大括号{}括起来的多个元素，中间有逗号隔开）中所有的元素，并返回表示第一个新插入元素位置的迭代器。

下面的程序演示了 insert() 函数的这几种用法：

```
#include <iostream>
#include <deque>
#include <array>
using namespace std;
int main()
{
    std::deque<int> d{ 1,2 };
    //第一种格式用法
    d.insert(d.begin() + 1, 3);//{1,3,2}

    //第二种格式用法
    d.insert(d.end(), 2, 5);//{1,3,2,5,5}

    //第三种格式用法
    std::array<int, 3>test{ 7,8,9 };
    d.insert(d.end(), test.begin(), test.end());//{1,3,2,5,5,7,8,9}

    //第四种格式用法
    d.insert(d.end(), { 10,11 });//{1,3,2,5,5,7,8,9,10,11}

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

运行结果为：

```
1,3,2,5,5,7,8,9,10,11
```

emplace系列函数的优势

有关 emplace()、emplace_front() 和 emplace_back() 分别和 insert()、push_front() 和 push_back() 在运行效率上的对比，可以通过下面的程序体现出来：

```
#include <deque>
#include <iostream>
using namespace std;
class testDemo
{
public:
    testDemo(int num) :num(num) {
        std::cout << "调用构造函数" << endl;
    }
    testDemo(const testDemo& other) :num(other.num) {
        std::cout << "调用拷贝构造函数" << endl;
    }
    testDemo(testDemo&& other) :num(other.num) {
        std::cout << "调用移动构造函数" << endl;
    }
    testDemo& operator=(const testDemo& other);
private:
    int num;
};

testDemo& testDemo::operator=(const testDemo& other) {
    this->num = other.num;
    return *this;
}
int main()
{
    //emplace和insert
    cout << "emplace:" << endl;
    std::deque<testDemo> demo1;
    demo1.emplace(demo1.begin(), 2);
    cout << "insert:" << endl;
    std::deque<testDemo> demo2;
    demo2.insert(demo2.begin(), 2);

    //emplace_front和push_front
    cout << "emplace_front:" << endl;
    std::deque<testDemo> demo3;
    demo3.emplace_front(2);
    cout << "push_front:" << endl;
    std::deque<testDemo> demo4;
    demo4.push_front(2);

    //emplace_back()和push_back()
    cout << "emplace_back:" << endl;
```

```
45.     std::deque<testDemo> demo5;
46.     demo5.emplace_back(2);
47.
48.     cout << "push_back:" << endl;
49.     std::deque<testDemo> demo6;
50.     demo6.push_back(2);
51.     return 0;
52. }
```

运行结果为：

```
emplace:
调用构造函数
insert:
调用构造函数
调用移动构造函数
emplace_front:
调用构造函数
push_front:
调用构造函数
调用移动构造函数
emplace_back:
调用构造函数
push_back:
调用构造函数
调用移动构造函数
```

可以看到，相比和它同功能的函数，emplace 系列函数都只调用了构造函数，而没有调用移动构造函数，这无疑提高了代码的运行效率。