C++ STL vector添加元素（push_back()和emplace_back()）详解

要知道，向 vector 容器中添加元素的唯一方式就是使用它的成员函数，如果不调用成员函数，非成员函数既不能添加也不能删除元素。这意味着，vector 容器对象必须通过它所允许的函数去访问，迭代器显然不行。

在《STL vector容器详解》一节中，已经给大家列出了 vector 容器提供的所有成员函数，在这些成员函数中，可以用来给容器中添加元素的函数有 2 个，分别是 push_back() 和 emplace_back() 函数。

> 有读者可能认为还有 insert() 和 emplace() 成员函数，严格意义上讲，这 2 个成员函数的功能是向容器中的指定位置插入元素，后续章节会对它们做详细的介绍。

push_back()

该成员函数的功能是在 vector 容器尾部添加一个元素，用法也非常简单，比如：

```
01.  #include <iostream>
02.  #include <vector>
03.  using namespace std;
04.  int main()
05.  {
06.      vector<int> values{};
07.      values.push_back(1);
08.      values.push_back(2);
09.      for (int i = 0; i < values.size(); i++) {
10.          cout << values[i] << " ";
11.      }
12.      return 0;
13.  }
```

程序中，第 7 行代码表示向 values 容器尾部添加一个元素，但由于当前 values 容器是空的，因此新添加的元素 1 无疑成为了容器中首个元素；第 8 行代码实现的功能是在现有元素 1 的后面，添加元素 2。

运行程序，输出结果为：

```
1 2
```

emplace_back()

该函数是 C++ 11 新增加的，其功能和 push_back() 相同，都是在 vector 容器的尾部添加一个元素。

emplace_back() 成员函数的用法也很简单，这里直接举个例子：

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> values{};
    values.emplace_back(1);
    values.emplace_back(2);
    for (int i = 0; i < values.size(); i++) {
        cout << values[i] << " ";
    }
    return 0;
}
```

运行结果为：

```
1 2
```

读者可能会发现，以上 2 段代码，只是用 emplace_back() 替换了 push_back()，既然它们实现的功能是一样的，那么 C++ 11 标准中为什么要多此一举呢?

emplace_back()和push_back()的区别

emplace_back() 和 push_back() 的区别，就在于底层实现的机制不同。push_back() 向容器尾部添加元素时，首先会创建这个元素，然后再将这个元素拷贝或者移动到容器中（如果是拷贝的话，事后会自行销毁先前创建的这个元素）；而 emplace_back() 在实现时，则是直接在容器尾部创建这个元素，省去了拷贝或移动元素的过程。

为了让大家清楚的了解它们之间的区别，我们创建一个包含类对象的 vector 容器，如下所示：

```
#include <vector>
#include <iostream>
using namespace std;
class testDemo
{
```

```
06.  public:
07.      testDemo(int num):num(num){
08.          std::cout << "调用构造函数" << endl;
09.      }
10.      testDemo(const testDemo& other) :num(other.num) {
11.          std::cout << "调用拷贝构造函数" << endl;
12.      }
13.      testDemo(testDemo&& other) :num(other.num) {
14.          std::cout << "调用移动构造函数" << endl;
15.      }
16.  private:
17.      int num;
18.  };
19.
20.  int main()
21.  {
22.      cout << "emplace_back:" << endl;
23.      std::vector<testDemo> demo1;
24.      demo1.emplace_back(2);
25.
26.      cout << "push_back:" << endl;
27.      std::vector<testDemo> demo2;
28.      demo2.push_back(2);
29.  }
```

运行结果为：

```
emplace_back:
调用构造函数
push_back:
调用构造函数
调用移动构造函数
```

在此基础上，读者可尝试将 testDemo 类中的移动构造函数注释掉，再运行程序会发现，运行结果变为：

```
emplace_back:
调用构造函数
push_back:
调用构造函数
调用拷贝构造函数
```

由此可以看出，push_back() 在底层实现时，会优先选择调用移动构造函数，如果没有才会调用拷贝构造函数。

显然完成同样的操作，push_back() 的底层实现过程比 emplace_back() 更繁琐，换句话说，emplace_back() 的执行效率比 push_back() 高。因此，在实际使用时，建议大家优先选用 emplace_back()。

> 由于 emplace_back() 是 C++ 11 标准新增加的，如果程序要兼顾之前的版本，还是应该使用 push_back()。