emplace_back push_back

emplace_back 效率高些，在vector末尾原地创建对象，不用移动；push_back 是在vector外面拷贝函数，然后移动到vector里

3. 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：

```cpp
#include <vector>
#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;
    }
private:
    int num;
};

int main()
{
    cout << "emplace_back:" << endl;
    std::vector<testDemo> demo1;
    demo1.emplace_back(2);

    cout << "push_back:" << endl;
    std::vector<testDemo> demo2;
    demo2.push_back(2);
}
```

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
```

登录复制

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()。

使用emplace_back() 函数可以减少一次拷贝或移动构造的过程，提升容器插入数据的效率。

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()。