

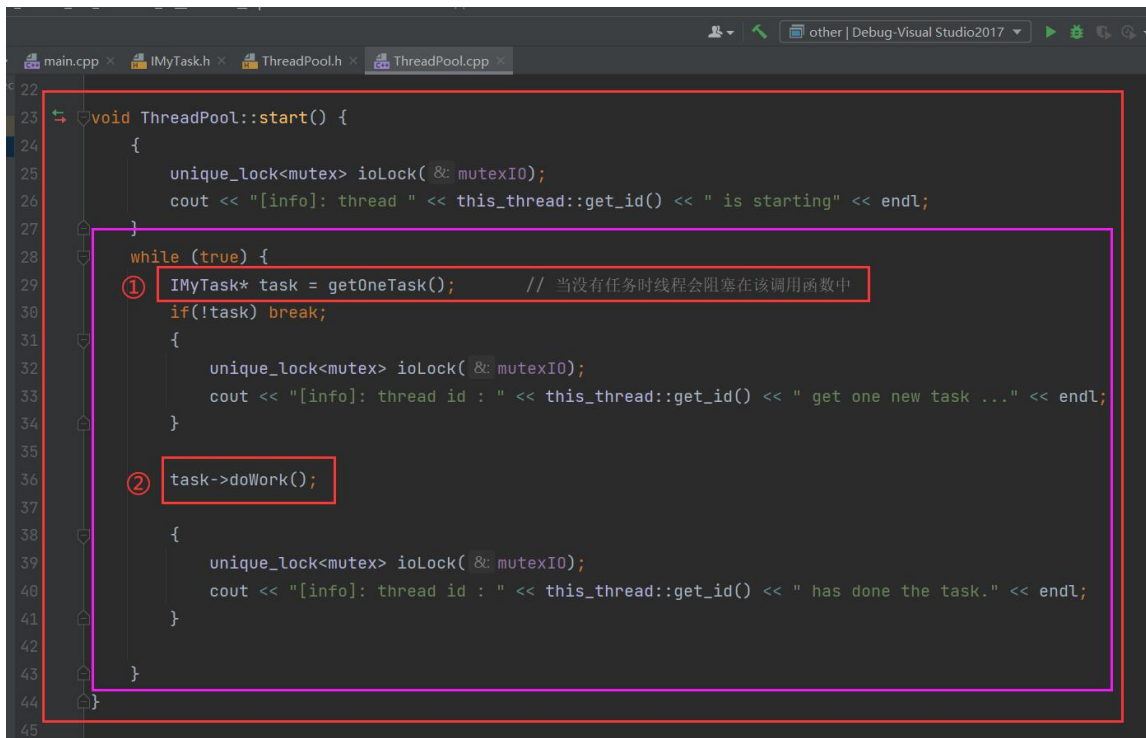
```
1 //
2 // Created by 19216 on 2021/5/8.
3 //
4
5 #include "ThreadPool.h"
6
7 ThreadPool::ThreadPool(int number): threadNum(number), stop(_Val: false) {
8     for (int i = 0; i < number; ++i) {
9         threadPool.emplace_back(&ThreadPool::start, this);
10     }
11 }
12
13 void ThreadPool::Enqueue(IMyTask *task) {
14     unique_lock<mutex> lock( & mutexTask);
15     taskQueue.push(task);
16     conditionVariableTask.notify_all();
17 }
18
19 void ThreadPool::Dispose() {
20     stop.store( _Value: true);
21 }
22
23 void ThreadPool::start() {
24     {
25         unique_lock<mutex> ioLock( & mutexIO);
```

Инициализируйте количество потоков в конструкторе, который должен подготовиться к запуску. Конкретной исполнительской функцией является функция запуска.

```
22
23 void ThreadPool::start() {
24     {
25         unique_lock<mutex> ioLock( & mutexIO);
26         cout << "[info]: thread " << this_thread::get_id() << " is starting" << endl;
27     }
28     while (true) {
29         IMyTask* task = getOneTask(); // 当没有任务时线程会阻塞在该调用函数中
30         if(!task) break;
31         {
32             unique_lock<mutex> ioLock( & mutexIO);
33             cout << "[info]: thread id : " << this_thread::get_id() << " get one new task ..." << endl;
34         }
35         task->doWork();
36         {
37             unique_lock<mutex> ioLock( & mutexIO);
38             cout << "[info]: thread id : " << this_thread::get_id() << " has done the task." << endl;
39         }
40     }
41 }
42
43
44
45
```

В функции start есть бесконечный цикл. Вместо ① будет заблокировано ожидание новой задачи. Если есть новая задача, задача будет получена напрямую. Если задачи в очереди нет, она будет заблокирована здесь. Тогда ② - приступить к выполнению этой задачи после получения новой задачи.

То есть поток будет иметь функцию запуска, поэтому в пуле потоков будет несколько запущенных функций.



```
22
23 void ThreadPool::start() {
24 {
25     unique_lock<mutex> ioLock( &mutexIO);
26     cout << "[info]: thread " << this_thread::get_id() << " is starting" << endl;
27 }
28     while (true) {
29         ① IMyTask* task = getOneTask(); // 当没有任务时线程会阻塞在该调用函数中
30         if(!task) break;
31         {
32             unique_lock<mutex> ioLock( &mutexIO);
33             cout << "[info]: thread id : " << this_thread::get_id() << " get one new task ..." << endl;
34         }
35
36         ② task->doWork();
37
38         {
39             unique_lock<mutex> ioLock( &mutexIO);
40             cout << "[info]: thread id : " << this_thread::get_id() << " has done the task." << endl;
41         }
42     }
43 }
44 }
45
```

Строка журнала будет печататься каждый раз при запуске потока.