**C++ 多线程：线程池从设计到实现**

**介绍**

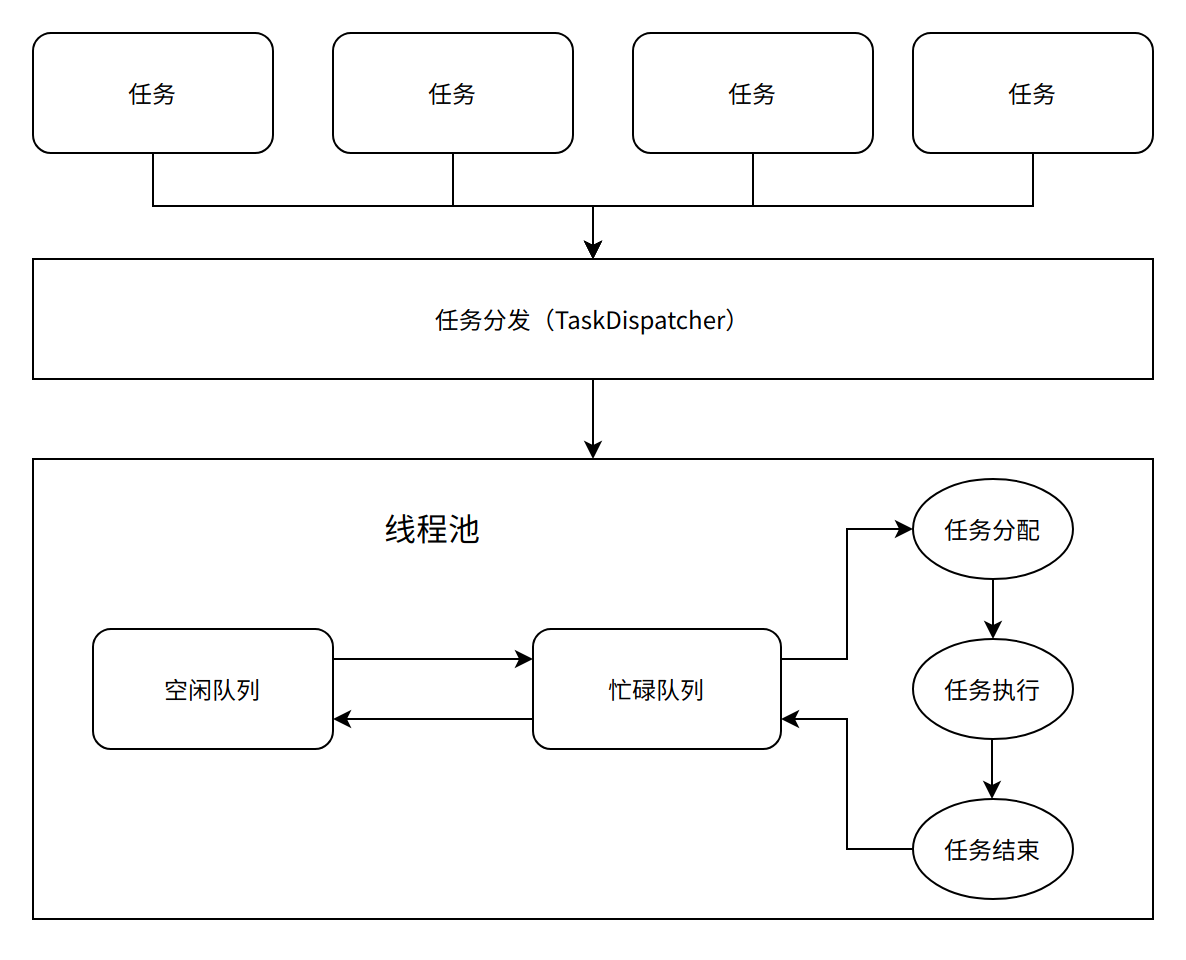
**为什么用多线程？**

1. 进程之间切换代价比较高，线程之间切换代价比较小
2. 解决 CPU 和 IO 速度不匹配的问题，多线程更适合在 IO 切换频繁的场景
3. 充分利用多核 CPU 资源、提高程序的并发效率

**课程目标**

1. 实现一个基于 posix thread 的线程池
2. 实现一个基于 c++11 thread 的线程池
3. 两个版本的线程池架构设计完全一样

**整体架构**



**实际应用**

**任务接口**

|  |
| --- |
| C++ class Task { public:  Task();  Task(void\* data);  virtual ~Task();   void\* get\_data();  void set\_data(void\* data);   virtual void run() = 0;  virtual void destroy() = 0;  protected:  void\* m\_data;  Mutex m\_mutex; }; |

**初始化线程池**

|  |
| --- |
| C++ int threads = 8; Singleton<TaskDispatcher>::instance()->init(threads); |

**任务分发**

|  |
| --- |
| C++ Task \* task = new EchoTask(); Singleton<TaskDispatcher>::instance()->assign(task); |