线程池项目

> **施磊老师QQ：2491016871　学习课程加入我们的学习群，解答大家学习过程中的各类问题**
>
> **环境：vs2019开发，C++17标准；centos7编译so动态库**

项目介绍

作为五大池之一（内存池、连接池、线程池、进程池、协程池），线程池的应用非常广泛，不管是客户端程序，还是后台服务程序，都是提高业务处理能力的必备模块。有很多开源的线程池实现，虽然各自接口使用上稍有区别，但是其核心实现原理都是基本相同的。

知识背景

> **推荐先学习完施磊老师出品的C++基础和高级课程，熟练C++面向对象编程和C++ 11编程。**

- 熟练基于C++ 11标准的面向对象编程

  组合和继承、继承多态、STL容器、智能指针、函数对象、绑定器、可变参模板编程等。
- 熟悉C++11多线程编程

  thread、mutex、atomic、condition_variable、unique_lock等。
- C++17和C++20标准的内容

  C++17的any类型和C++20的信号量semaphore，项目上都我们自己用代码实现。
- 熟悉多线程理论

  多线程基本知识、线程互斥、线程同步、原子操作、CAS等。

项目代码片段

```cpp
template<typename Func, typename... Args>
auto submitTask(Func&& func, Args&&... args) -> std::future<decltype(func(args...))>
{
    // 打包任务，放入任务队列里面
    using RType = decltype(func(args...));
    auto task = std::make_shared<std::packaged_task<RType()>>(
        std::bind(std::forward<Func>(func), std::forward<Args>(args)...));
    std::future<RType> result = task->get_future();

    // 获取锁
    std::unique_lock<std::mutex> lock(taskQueMtx_);
    // 用户提交任务，最长不能阻塞超过1s，否则判断提交任务失败，返回
    if (!notFull_.wait_for(lock, std::chrono::seconds(1),
        [&]()->bool { return taskQue_.size() < (size_t)taskQueMaxThreshHold_; }))
    {
        // 表示notFull_等待1s种，条件依然没有满足
        std::cerr << "task queue is full, submit task fail." << std::endl;
        auto task = std::make_shared<std::packaged_task<RType()>>(
            []()->RType { return RType(); });
        (*task)();
```

并发和并行

- CPU单核
- CPU多核、多CPU

并发

单核上，多个线程占用不同的CPU时间片，物理上还是串行执行的，但是由于每个线程占用的CPU时间片非常短（比如10ms），看起来就像是多个线程都在共同执行一样，这样的场景称作并发（concurrent）。

并行

在多核或者多CPU上，多个线程是在真正的同时执行，这样的场景称作并行（parallel）。

多线程的优势

多线程程序一定就好吗？不一定，要看具体的应用场景：

IO密集型

无论是CPU单核、CPU多核、多CPU，都是比较适合多线程程序的

CPU密集型

- CPU单核

多线程存在上下文切换，是额外的花销，线程越多上下文切换所花费的额外时间也越多，倒不如一个线程一直进

行计算。

- CPU多核、多CPU

多个线程可以并行执行，对CPU利用率好。

线程池

线程的消耗

为了完成任务，创建很多的线程可以吗？线程真的是越多越好？

- 线程的创建和销毁都是非常"重"的操作
- 线程栈本身占用大量内存
- 线程的上下文切换要占用大量时间
- 大量线程同时唤醒会使系统经常出现锯齿状负载或者瞬间负载量很大导致宕机

线程池的优势

操作系统上创建线程和销毁线程都是很"重"的操作，耗时耗性能都比较多，那么在服务执行的过程中，如果业务量比较大，实时的去创建线程、执行业务、业务完成后销毁线程，那么会导致系统的实时性能降低，业务的处理能力也会降低。

线程池的优势就是（每个池都有自己的优势），在服务进程启动之初，就事先创建好线程池里面的线程，当业务流量到来时需要分配线程，直接从线程池中获取一个空闲线程执行task任务即可，task执行完成后，也不用释放线程，而是把线程归还到线程池中继续给后续的task提供服务。

fixed模式线程池

线程池里面的线程个数是固定不变的，一般是ThreadPool创建时根据当前机器的CPU核心数量进行指定。

cached模式线程池

线程池里面的线程个数是可动态增长的，根据任务的数量动态的增加线程的数量，但是会设置一个线程数量的阈值（线程过多的坏处上面已经讲过了），任务处理完成，如果动态增长的线程空闲了60s还没有处理其它任务，那么关闭线程，保持池中最初数量的线程即可。

线程同步

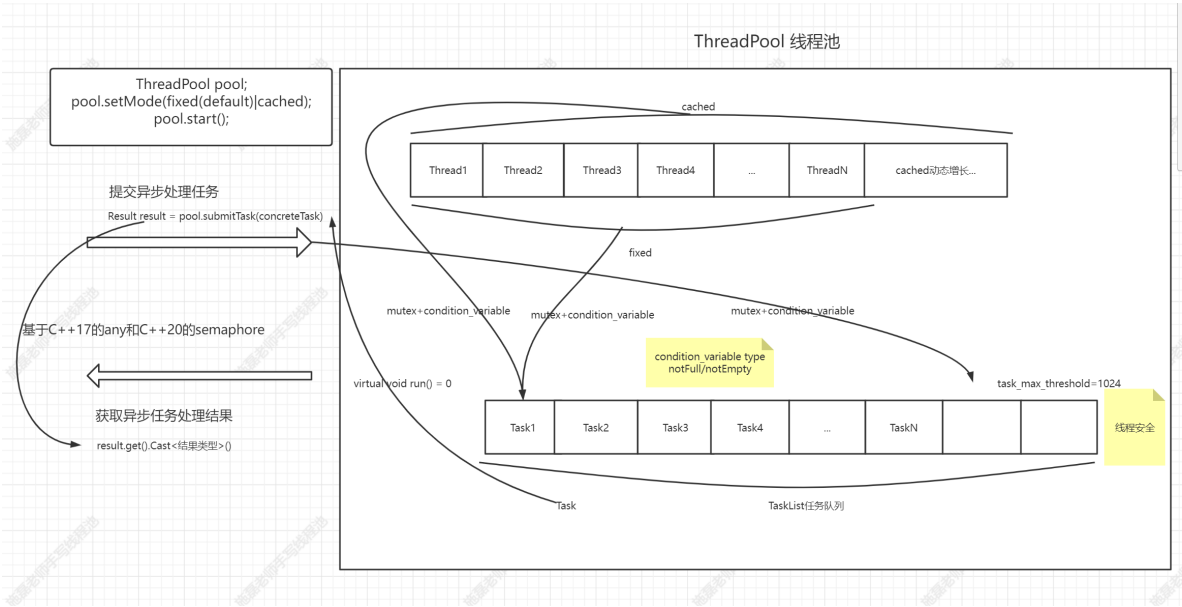
线程互斥

- 互斥锁mutex
- atomic原子类型

线程通信

- 条件变量 condition_variable
- 信号量 semaphore

项目设计



项目输出

应用到项目中

- 高并发网络服务器
- master-slave线程模型
- 耗时任务处理

输出到简历上

项目名称：基于可变参模板实现的线程池

git地址：git@gitee.com:xxx/xxx.git

平台工具：vs2019开发，centos7 g++编译so库，gdb调试分析定位死锁问题

项目描述：

1、基于可变参模板编程和引用折叠原理，实现线程池submitTask接口，支持任意任务函数和任意参数的传递

2、使用future类型定制submitTask提交任务的返回值

3、使用map和queue容器管理线程对象和任务

4、基于条件变量condition_variable和互斥锁mutex实现任务提交线程和任务执行线程间的通信机制

5、支持fixed和cached模式的线程池定制

6、xxxx（自由发挥）

项目问题：

1、在ThreadPool的资源回收，等待线程池所有线程退出时，发生死锁问题，导致进程无法退出

2、在windows平台下运行良好的线程池，在linux平台下运行发生死锁问题，平台运行结果有差异化

分析定位问题：

主要通过gdb attach到正在运行的进程，通过info threads，thread tid，bt等命令查看各个线程的调用堆栈信息，结合项目代码，定位到发生死锁的代码片段，分析死锁问题发生的原因，xxxx，以及最终的解决方案。