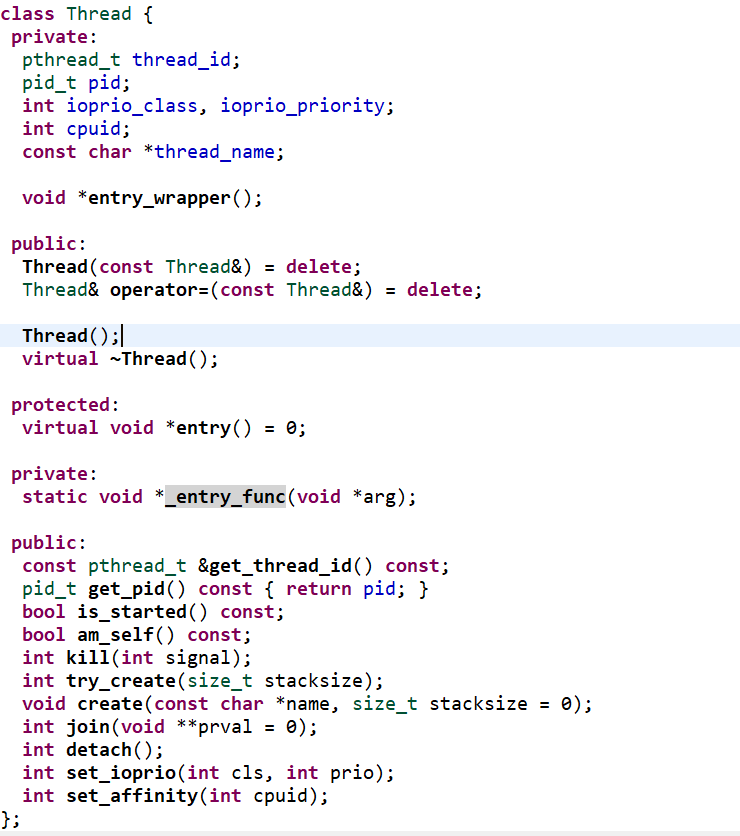
ceph 为了支持高并发读写，源码设计中大量采用线程池来进行io的推进。本文主要介绍其线程池的实现机制.

线程的实现

ceph自己封装了一个线程类，这个线程类其实就是对Linux线程接口的一层封装



分析下这个类中的一些接口：

virtual void \*entry() = 0 就是该线程的执行函数，任何继承Thread的具体线程类都要实现这个接口。

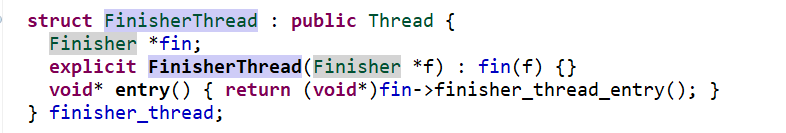
int try\_create(size\_t stacksize) 创建线程的接口。

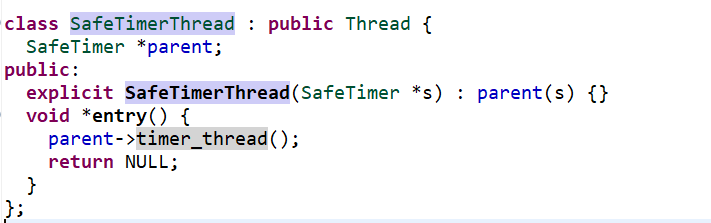
int join(void \*\*prval = 0) 等待线程结束，资源回收

int detach()，父子线程分离

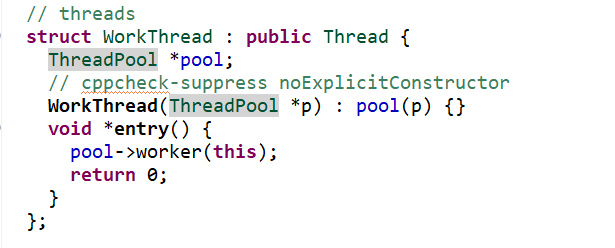
int kill(int signal)，线程自杀。

可以看到所有接口都只是对相应的Linux接口的封装。继承其的子类主要在于实现entry()函数：

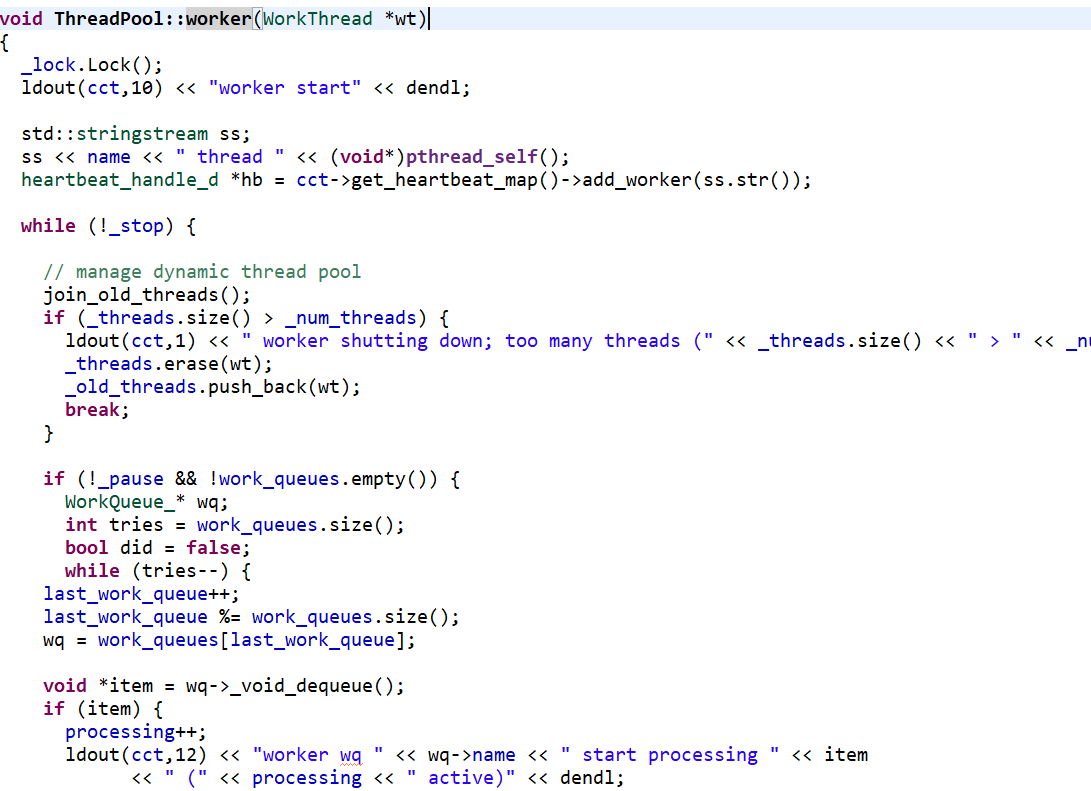




线程池里面也定义了自己的线程，其entry函数其实就是调用线程池的worker函数进行具体的工作。



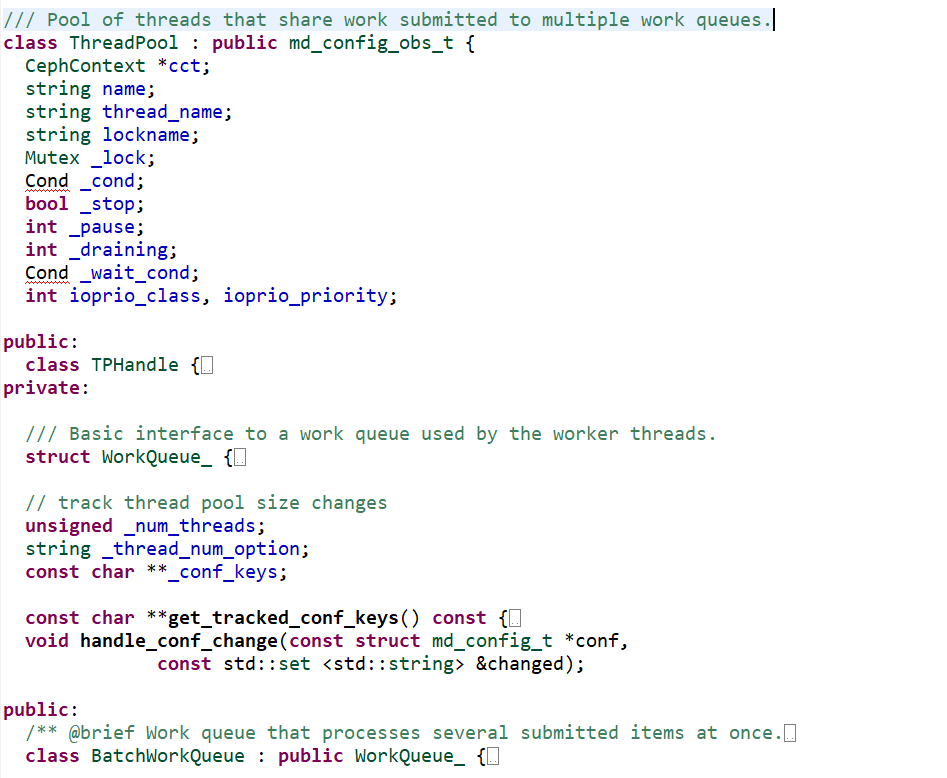
我们来看下worker函数的实现：



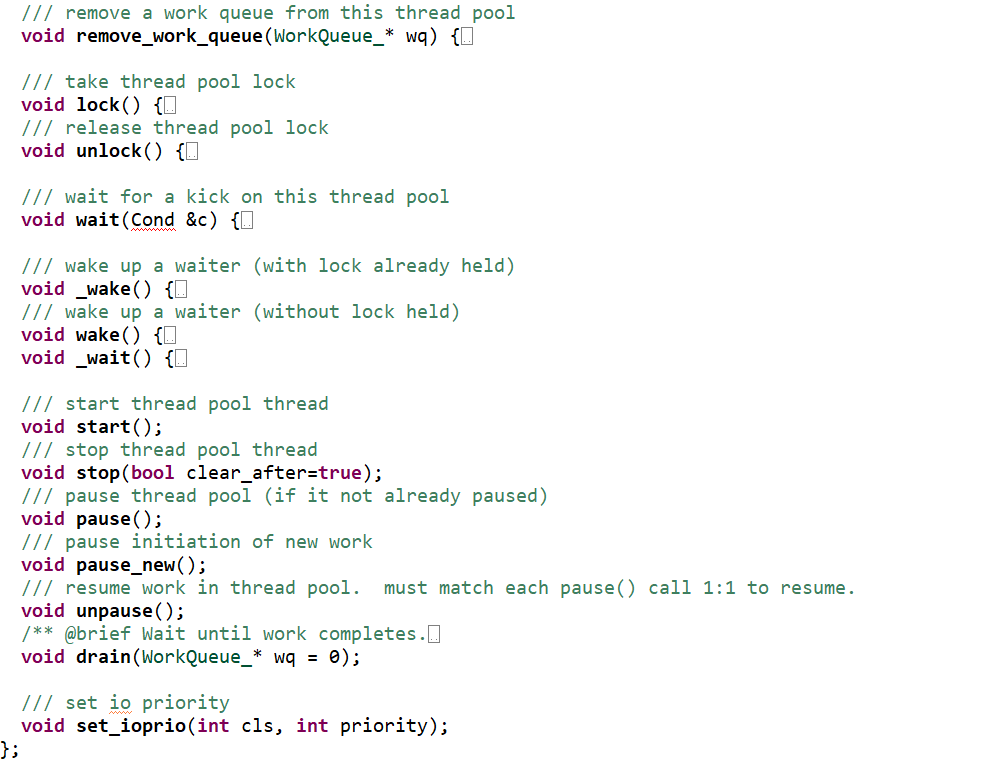


worker函数的主要实现其实很常规，就是遍历work\_queues，从其中找出每一个消息队列实例，并调用WorkQueue\_自己定义的函数\_void\_process和\_void\_process\_finish等函数来顺序执行操作。

线程池的实现：





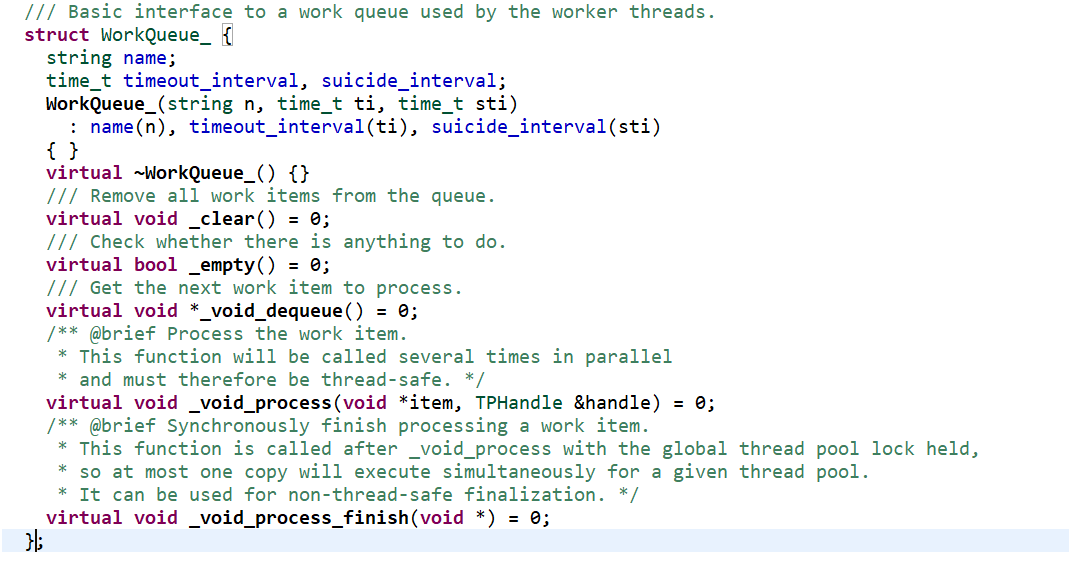


在这个类中，set<WorkThread\*> \_threads保存着线程池中的多个线程

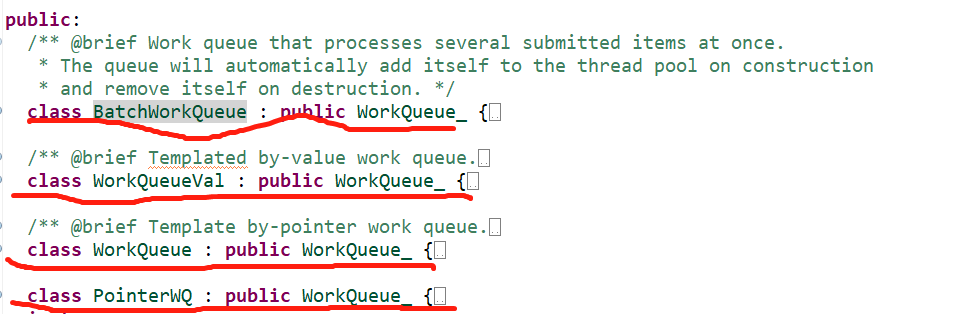
vector<WorkQueue\_\*> work\_queues保存着线程池中的待线程处理的消息队列。

整个线程池的原理思想比较简单就是生成一定数目的线程，然后线程从队列中遍历获取队列实例，调用实例自带的处理函数\_void\_process和\_void\_process\_finish处理。

我们在看看存放在work\_queues里面的WorkQueue\_类：



这是一个纯虚基类，也就是说不同的线程池要实现自己的队列，继承WorkQueues\_并且实现其接口。线程池已经有4个纯虚基类继承这个类：



BatchWorkQueue: 批量处理队列

WorkQueueVal：存值队列

WorkQueue： 存指针队列

PointerWQ：存指针队列

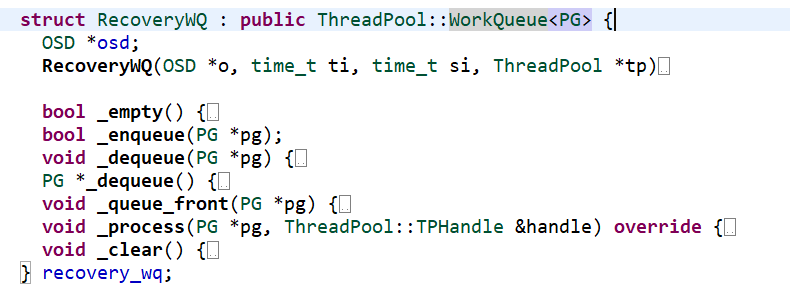
在ceph中有不少线程池会实现继承以上基类：

ThreadPool op\_tp：处理client请求



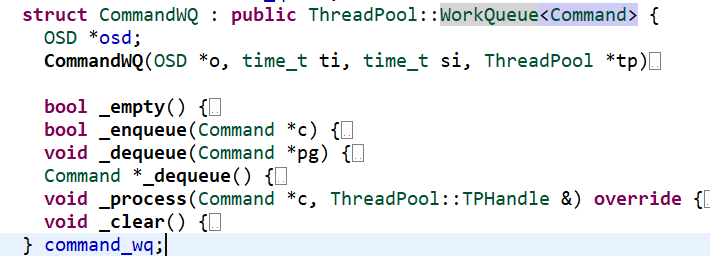
recovery\_tp: 处理recovery\_tp操作





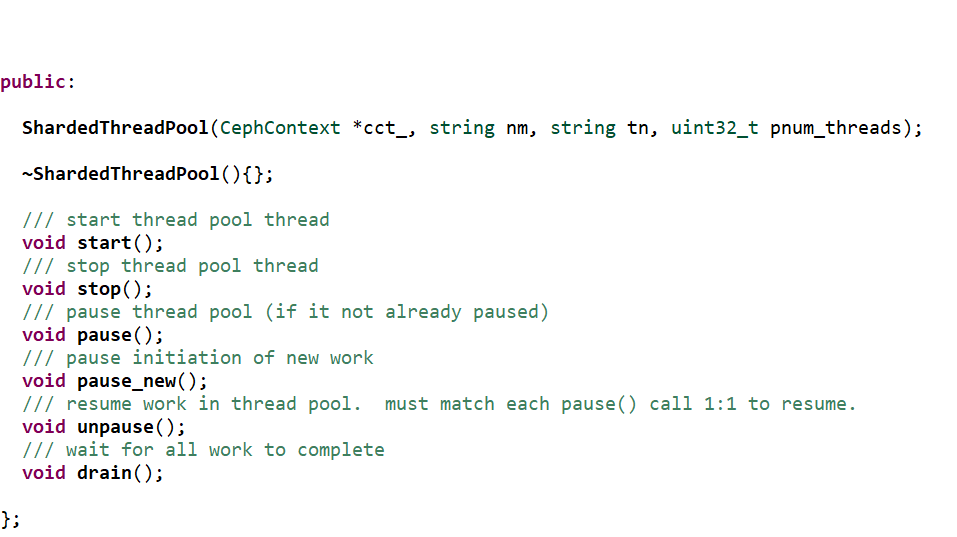
command\_tp:  处理命令行来的操作



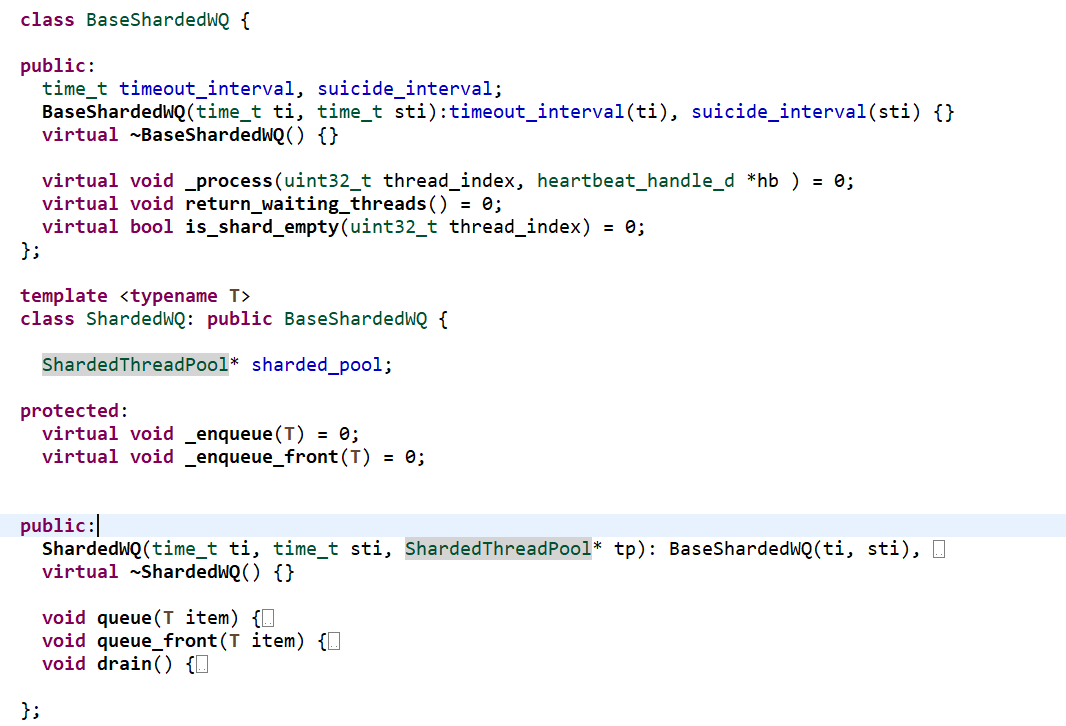


ceph还实现了另外一种线程池ShardedThreadPool，这种线程池与上面的线程池不同之处在于这种线程池是多线程共享队列的方式。只有一个队列，多个线程同时对这个队列进行处理。





shardedThreadPool类型线程池内部有个比较重要的消息队列SharededWQ,该队列将多种OP放入其中



ceph 在实际使用中，会用到这种线程池

