muduo库是如何实现跨线程函数调用的

lovebasamessi 2020-03-05 01:48:09 259 收藏

分类专栏： muduo

版权

        muduo库的跨线程函数主要是通过EventLoop类的成员函数runInLoop函数实现的，还是以具体的例子来一步一步的说明吧！

#include <muduo/net/EventLoop.h>

#include <muduo/net/EventLoopThread.h>

#include <stdio.h>

using namespace muduo;

using namespace muduo::net;

void th\_fn()

{

printf("th\_fn() : pid = %d, tid = %d\n", getpid(), CurrentThread::tid());

}

int main()

{

printf("main(): pid = %d, tid = %d\n",

getpid(), CurrentThread::tid());

EventLoopThread loopThread;

EventLoop\* loop = loopThread.startLoop();//创建了一个IO线程 并返回IO线程所对应的EventLoop对象的指针

loop->runInLoop(th\_fn);//跨线程调用函数th\_fn

sleep(1);

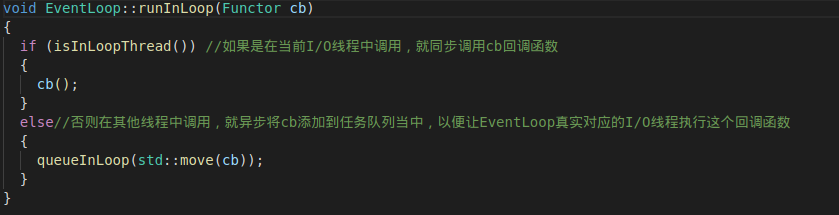
loop->quit();

printf("exit main().\n");

return 0;

}

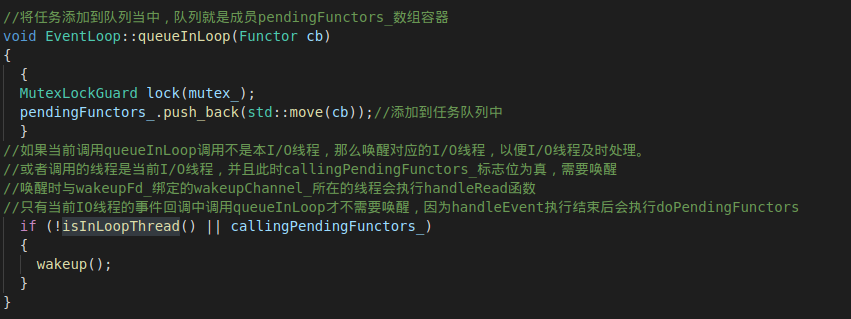
       上述程序中，loopThread.startLoop函数执行完毕会创建一个IO线程，并返回IO线程所对应的EventLoop对象的指针，也就是loop指针指向的EventLoop并不在主线程中，我们在主线程执行loop->runInLoop(th\_fn)怎么就能在loop指针指向的EventLoop所在的IO线程执行th\_fn函数呢？(为了分析方便，以下简称loop指针指向的EventLoop对象所对应的IO线程为loop指针指向的线程)我们一步一步来看，先来看EventLoop::runInLoop函数。



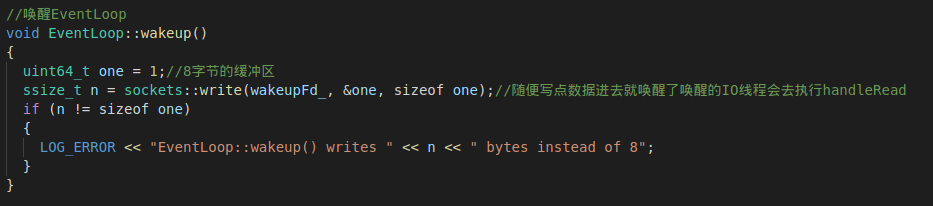
       该函数中，其首先会调用isInLoopThread()判断调用runInLoop函数的线程和loop指针指向的线程是不是同一个线程，如果是，就执行传过来的函数cb，在我们的例子中，就是th\_fn。如果不是，则执行queueInLoop函数。我们先来看看isInLoopThread()函数。



       首先，threadId\_是EventLoop对象的成员，在执行loop->runInLoop函数时，由于runInLoop是EventLoop的非静态成员函数，所以会传入一个隐式参数this指针，因此，这里的threadId\_就是this->threadId\_，也即loop->threadId\_，而currentThread::tid()表示调用currentThread::tid()函数的线程所对应的线程id，因为我们是在主线程中调用了loop->runInLoop，进而调用了isInLoopThread，又进而调用了currentThread::tid()，所以这里currentThread::tid()就是主线程的id，所以这里isInLoopThread返回false，也就是调用runInLoop函数的线程和loop指针指向的线程不是同一个线程。所以会去调用queueInLoop。我们来看看这个函数



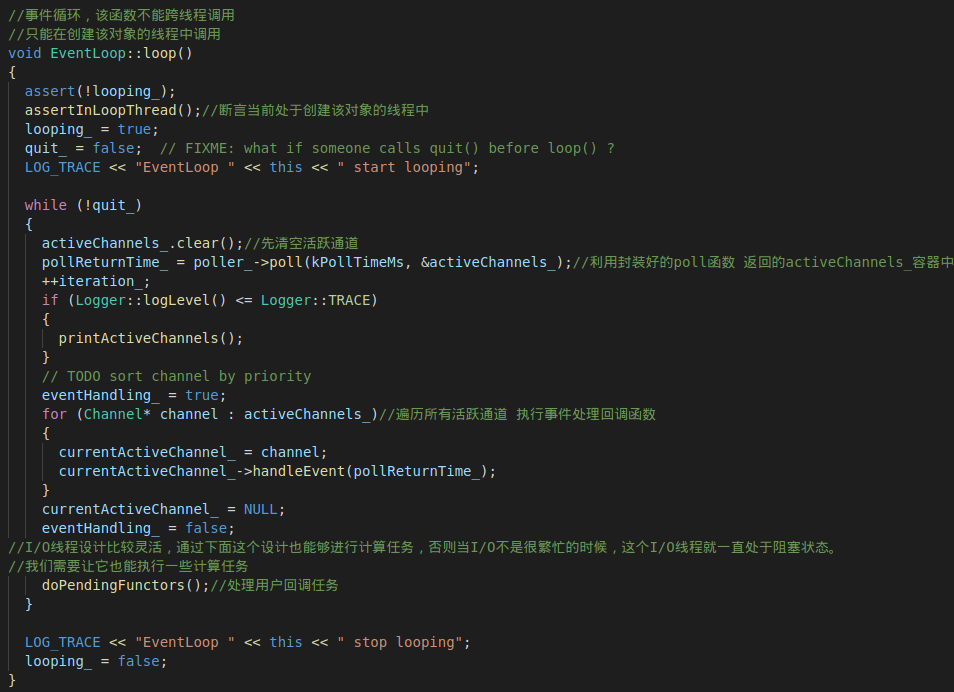
       可以看到，这里它先把要跨线程调用的函数添加到一个任务队列的容器pendingFunctors\_中(这里的pendingFunctors\_是loop指针指向的EventLoop的成员)，然后如果是跨线程调用，显然isInLoopThread为false，所以if条件必然成立，也就是说会去执行wakeup函数。注意这里的wakeup()相当于loop->wakeup()。



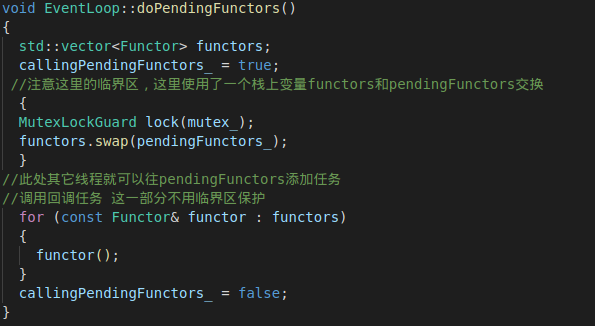
       wakeup函数其实就是向wakeupFd\_文件描述符8字节的数据，我们都知道，wakeupFd\_与wakeupChannel\_绑定，都是EventLoop对象的成员，并且在EventLoop的构造函数就被初始化，并注册了wakeupChannel\_的读回调以及设置关注可读事件。由于之前相当于是loop->wakeup，所以这里的wakeupFd\_和wakeupChannel\_都是loop指针指向的EventLoop对象的成员。



        那么很显然，由于刚刚我们像loop指向的EventLoop对象中的wakeupFd\_发了8字节数据，这会导致wakeupFd\_所在的IO线程的loop循环中原先阻塞监听的poll函数返回活跃通道，这样他就去执行注册的回调函数，执行完毕之后，我们看到其会执行doPendingFunctors()函数

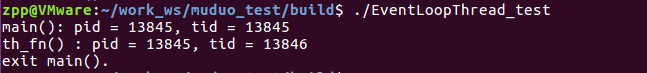


        我们来看看doPendingFunctors()函数，发现了什么，该函数相当于对存放函数指针的容器的函数一一执行，由于我们之前向该容器添加了th\_fn函数指针，所以th\_fn函数得以执行。注意，这里th\_fn函数是在loop指针指向的IO线程中执行的。



       综上所述，muduo库实现跨线程调用其实就是通过向指定的线程的wakeupFd\_上随便发点数据，让对应线程中loop()循环中之前阻塞监听的poll函数返回活跃通道，好让线程赶快执行回调函数。这样，回调函数执行完毕，其就可以调用doPendingFunctors函数执行我们之前添加的th\_fn函数了。也就是说，之前对应的线程可能没有活跃事件，所以一直阻塞，我们随便向wakeupFd\_上发点数据就可以让poll返回了，然后赶快处理对应通道上注册的回调，其实处理回到并不是我们的目的，只是说，线程只有执行完回调我们跨线程调用的函数才能被执行。wakeup的目的其实就是通知线程赶快执行我们跨线程调用的函数，不然，其实以后只要该线程上有活跃事件，其实还是跨线程调用的函数还是会被执行的。也就是说，不wakeup我们跨线程调用的函数也会被执行，只是什么时候被执行就不好说了，只要线程检测到活跃事件才行。另外，为什么我们可以在主线程中向loop指针指向的EventLoop对象的成员pendingFunctors\_中添加数据呢，这都是因为我们在主线程有了指向该EventLoop对象的指针loop，通过loop指针，我们就可以修改它上面的数据。

       我们最后来看一下上面程序的执行结果，可以看到执行main函数和th\_fn函数的线程id是不同的，说明实现了跨线程调用。



————————————————

原文链接：https://blog.csdn.net/lovebasamessi/article/details/104666853/