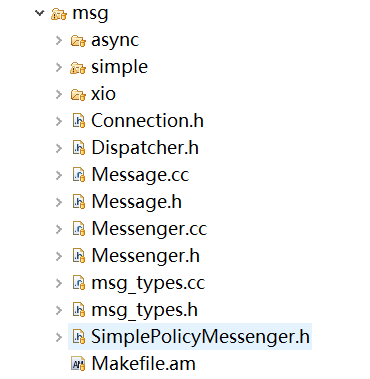
ceph作为分布式存储软件，其网络通信模块必不可少，是其关键的设计部分。Ceph有着自己的网络通信机制。本章分析ceph的网络通信实现。

先大概浏览下ceph的网络通信模块代码，集中在msg目录下面：

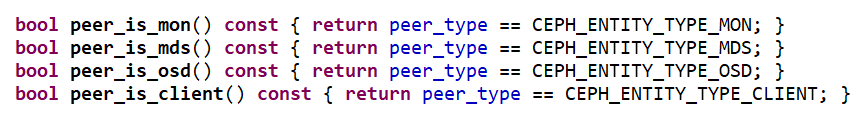


其中3个目录async,simple,xio分别是3种不同的通信模式。3种通信模式这里先不展开讨论

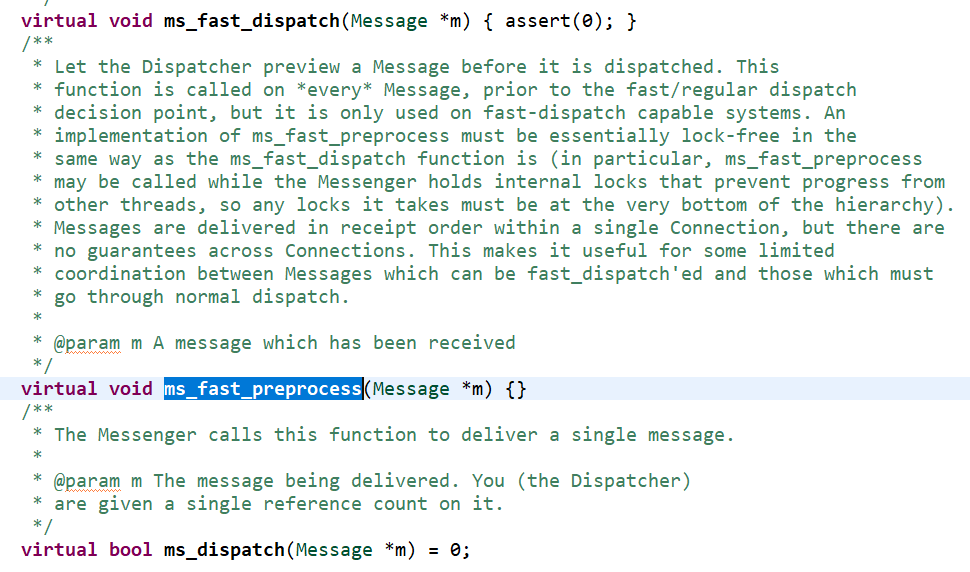
Connection.h主要定义一个网络连接以及与这次连接的相应操作。其中最主要的接口就是发送消息：

virtual int send\_message(Message \*m) = 0;

这个类是纯虚基类，需要子类继承实现。每个连接都属于不同的连接类型：

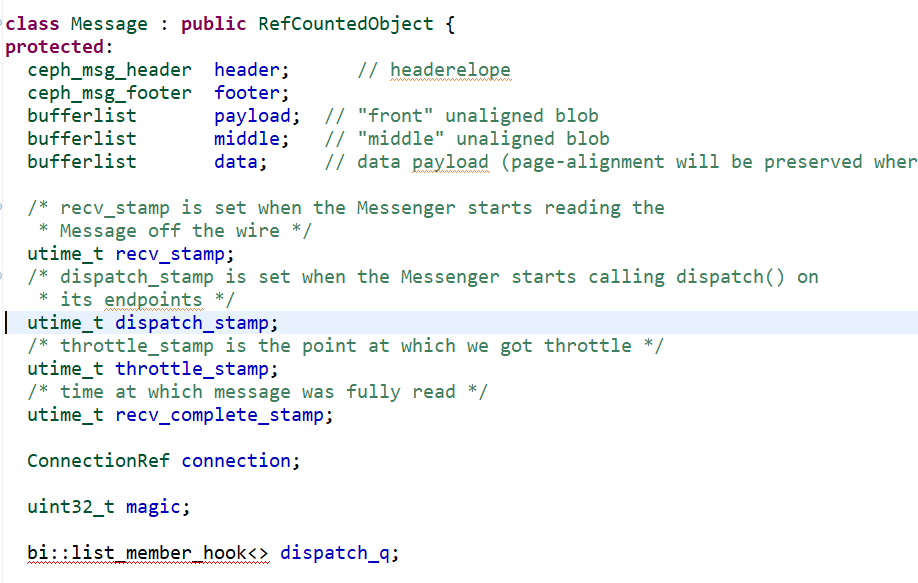
连接点是monitor，还是mds, 还是osd, 还是client, 不同的连接有不同的网络数据处理

Dispatcher.h主要定义网络消息的分发处理，主要是对接收到的网络消息分发给具体处理的应用层。类Dispatcher也是个纯虚基类，需要子类继承实现其功能。其中最主要的接口就是处理分发接收到的消息接口：



这3个接口都是消息的分发处理接口。是这个类的最主要的接口。其他的接口处理分发过程中的网络连接问题。

Message.h主要是定义网络数据结构体。类Message也是基类，任何要发送的消息，都要继承该类。



Ceph的各个网络模块通信都要设计符合message的数据格式：

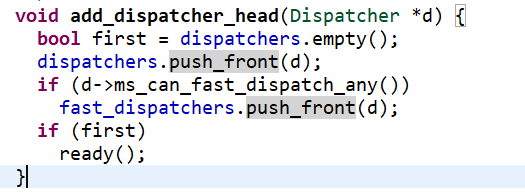
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| header | payload | middle | data | footer |

Messenger是整个网络抽象模块，定义网络模块的基本API接口。提供网络模块基本功能，能在节点之间发送和接收消息。

向一个节点发送消息的命令：

virtual int send\_message(Message \*m, const entity\_inst\_t& dest) = 0;

注册一个Dispatcher用来分发消息的命令如下：



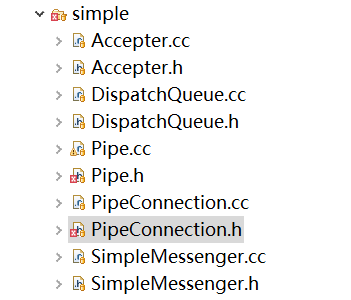
msg\_types.h定义网络连接的类型及其网络地址信息：



SimplePolicyMessenger.h主要设置一些策略，处理网络连接的策略。

上面说到了三种网络通信模式，这次专门讲解下simple模式

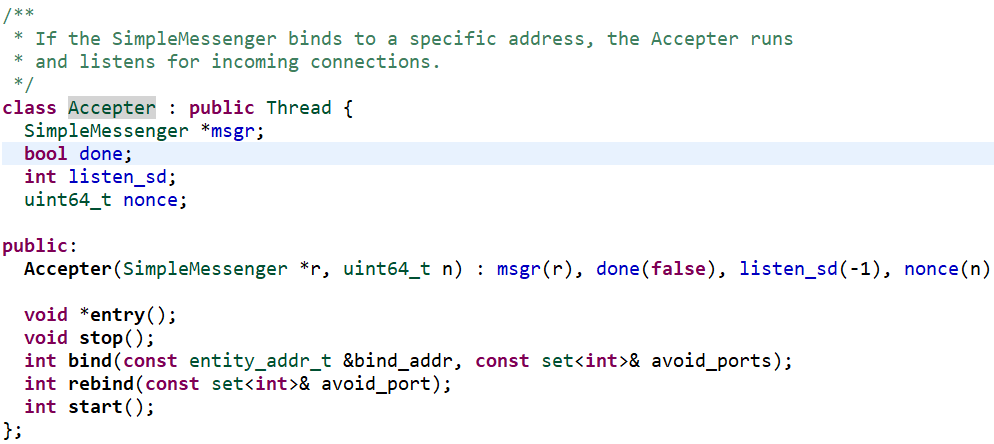
Simple代码结构如下：



同样的也大概分析了如下代码：

Accepter.h：主要是用来在Server端监听端口，接收连接，它继承了Thread类，本身是一个线程，不断的监听Server的端口：

类Accepter有如下接口：



函数entry()是函数的执行体，这个函数有个循环，不停的接收客户端的连接，并且将连接交给pipe处理。

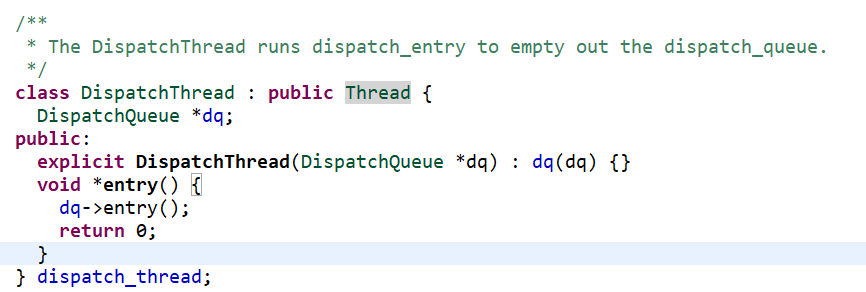
stop()关闭连接，停止监听端口。

bind()接口建立socket监听

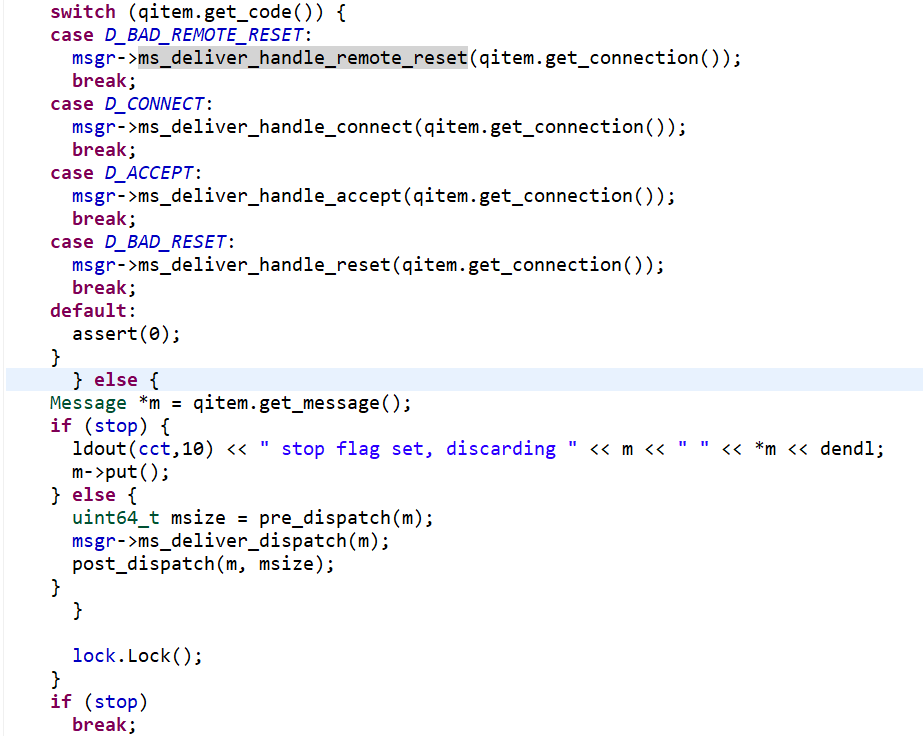
start()启动线程

DispatchQueue.h：主要是设计消息分发队列和消息分发线程，实现异步分发和并发功能。class DispatchQueue有几个重要的数据结构：

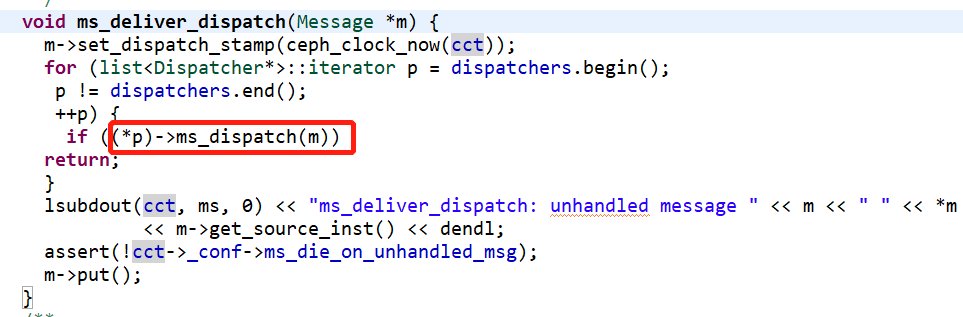
PrioritizedQueue<QueueItem, uint64\_t> mqueue; //优先级队列，所有要分发的消息存放在队列里面



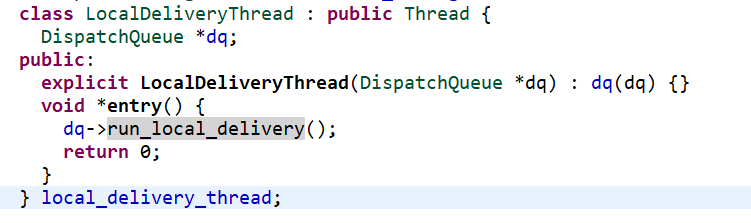
线程dispatch\_thread主要负责从队列里面取出消息，根据消息类型做相应的处理：



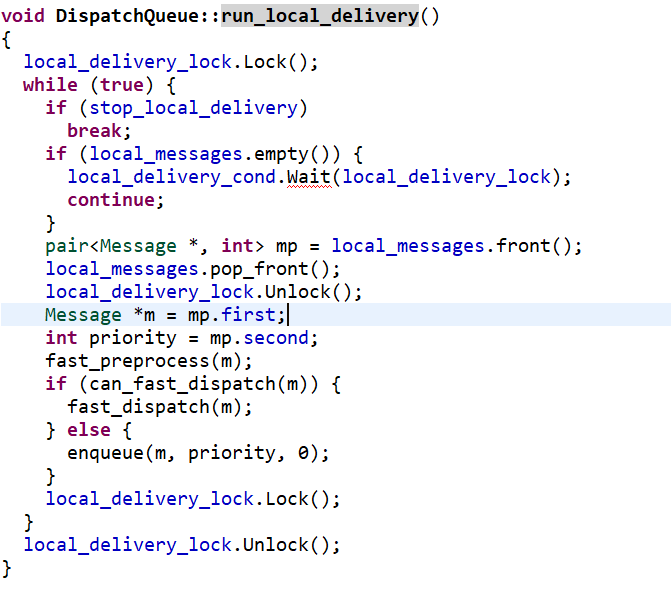
其中ms\_deliver\_dispatch()就是将消息分发各个dispatcher.



类LocalDeliveryThread主要用于本节点消息的分发处理：

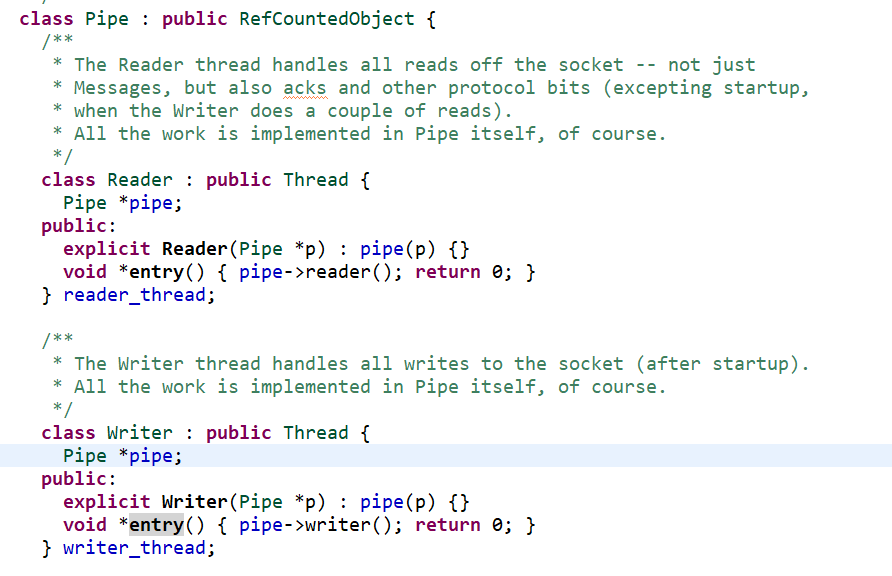


同样的，线程LocalDeliveryThread从local\_messages列表里取出消息处理：



其中fast\_dispatch就是对消息进行分发处理，不同的模块实现接口不同。

Pipe.h主要是负责处理每个连接的数据收发工作。Pipe实现了两个端口之间的类似管道的功能。对应每一个连接，都有一个pipe，内部有两个线程，一个reader用来接收消息，write用来发送消息：



PipeConnection.h： PipeConnection继承Connection,只要处理消息发送，pipe实现了其功能。

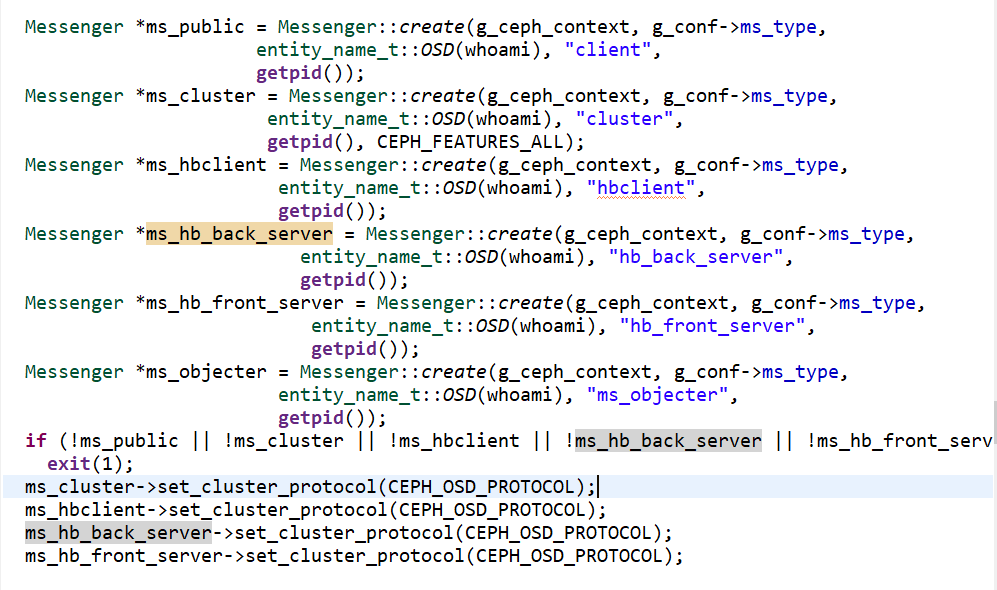
这里不再详述

SimpleMessenger.h： 类SimpleMessager实现了Messager的主要接口，是主要的类，ceph的每个服务都会生成一个SimpleMessenger实例来处理网络连接与发送处理。

上面粗略的描述了各个代码模块的基本功能。

以simple为例，以ceph-osd进程为例讲解网络模块在ceph是怎么运用的

ceph osd启动时，会创建SimpleMessenger实例,见ceph\_osd.cc:



可以看到osd 创建了6个Messenger，分别是client，用于与客户端的通信，cluster用于与其他osd的通信，hbclient用于发送ping心跳，hb\_back\_server用于接收back地址的ping心跳，hb\_front\_server接收来自front地址的ping心跳,ms\_objecter用于处理objecter的请求。

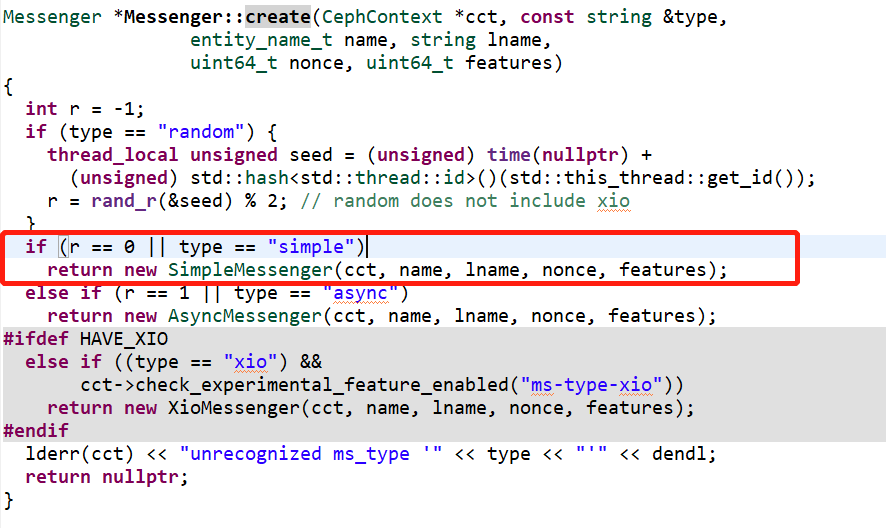
这次专门讲client的messenger, 其他的会专门展开讨论。

Messenger \*ms\_public = Messenger::*create*(g\_ceph\_context, g\_conf->ms\_type,

entity\_name\_t::*OSD*(whoami), "client",

**getpid**());

函数创建Messenger实例：

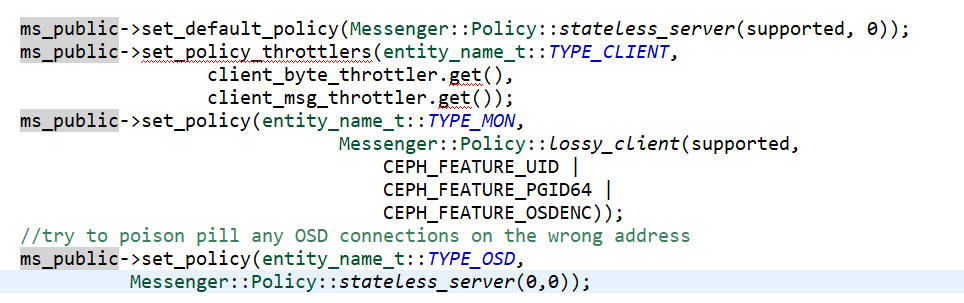


根据type是simple，new SimpleMessenger：



上面只是做一些参数的初始化工作。

接下来就是设置一些policy:

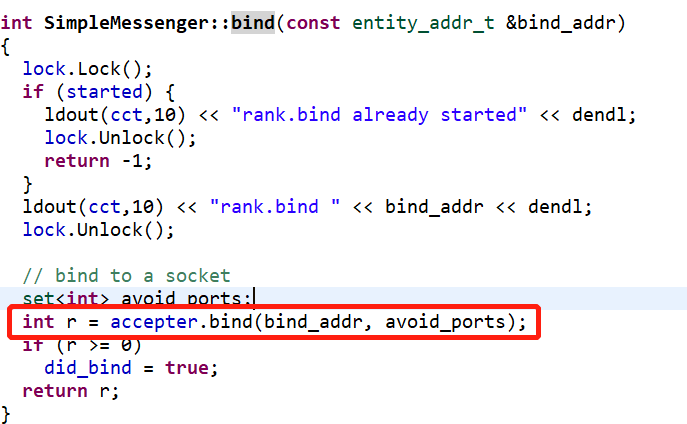


设置策略主要是为了兼容各个网络通信双方的兼容。这里先忽略不管。

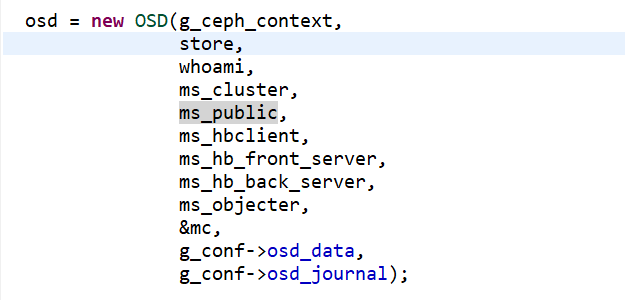
接下来就是绑定IP地址：



绑定socket，这个工作交由accepter去做，accepter本来就是用来监听端口，接收连接的。



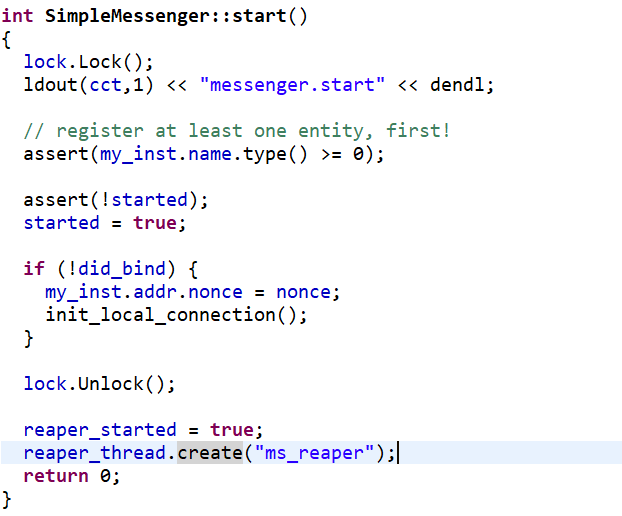
接下来就是创建OSD实例：



OSD类继承Dispatcher,就是做网络消息分发给具体的应用处理。

接下来就是启动SimpleMessenger:

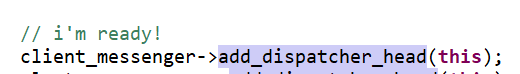




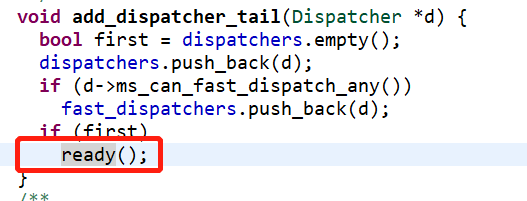
接着就是注册dispatcher了。即注册osd 实例：

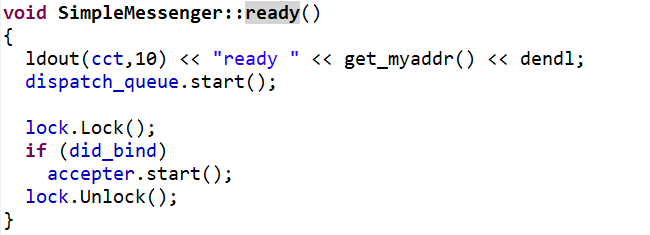


在函数中会注册进去：



注册的过程中也启动了accepter, 启动监听端口，接收连接：





接着进入wait()



至此ceph osd网络模块就启动了。