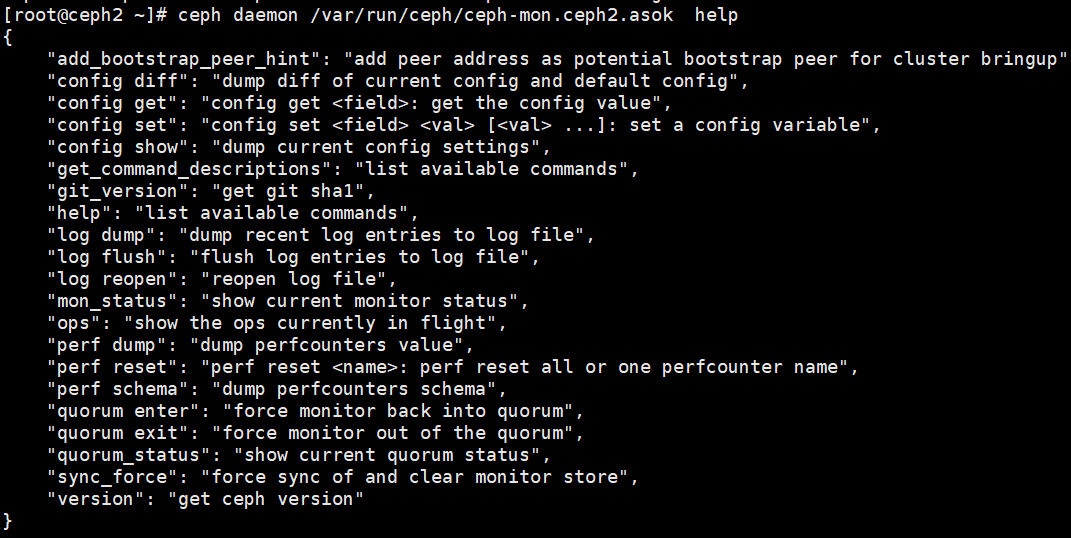
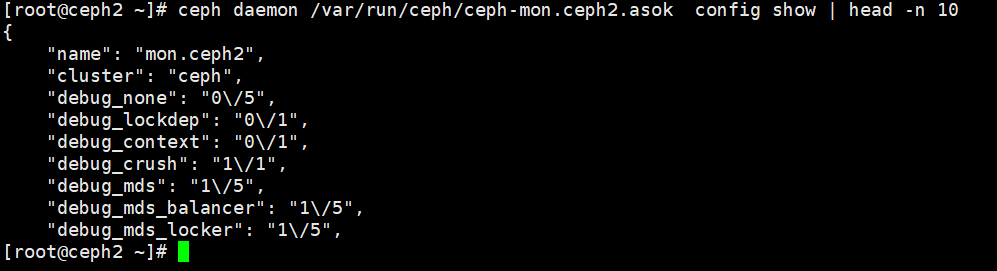
ceph作为大型软件，必须提供相应的方法能够探查当前进程内部运行的上下文环境，例如当前运行的配置信息，状态信息，同时也能够相应的做出修改。Ceph的admin socket机制提供了这样的方法。本节内容，我们分析下ceph admin socket的实现。

用ceph-mon为例，分析admin socket机制。

ceph 使用如下命令使用admin socket:



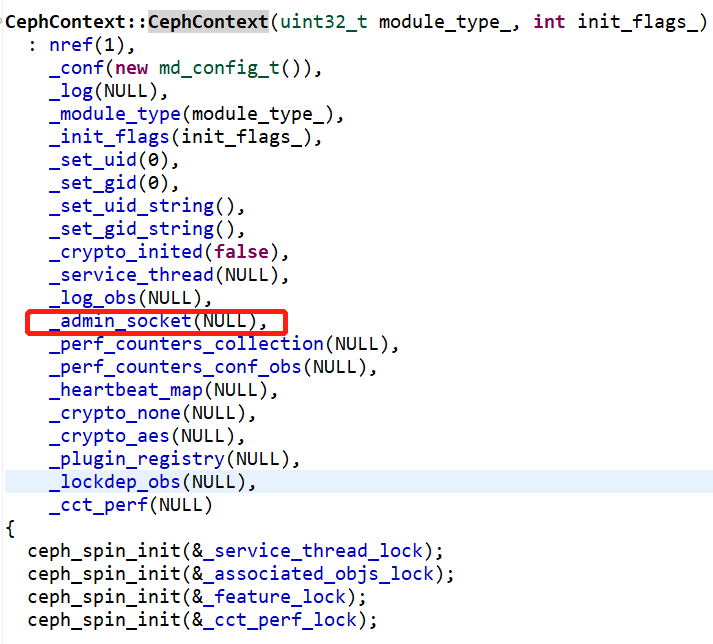
例如查看当前的配置信息：



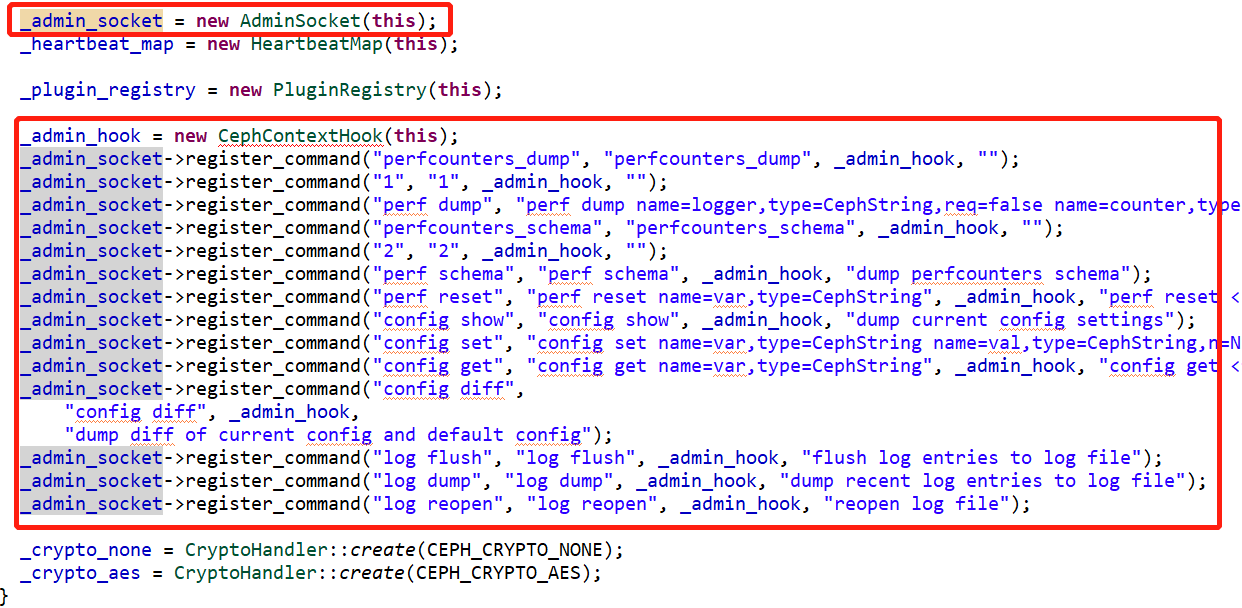
这个是如何实现的呢？就是借助于ceph admin socket机制

CephContext中会创建一个AdminSocket对象，该对象本质是一个线程。ceph-mon/ceph-osd/ceph-mds这些进程都会有创建一个AdminSocket的线程，负责响应用户的探查命令。

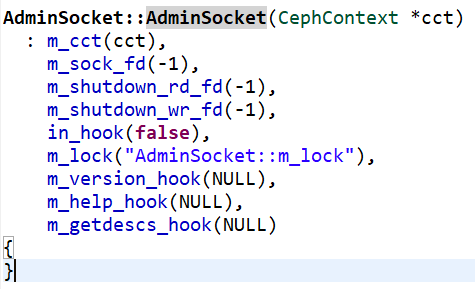
Monitor 启动过程中会构造CephContext, 初始化admin socket:



在生成CephContext实例中会new 一个AdminScoket实例



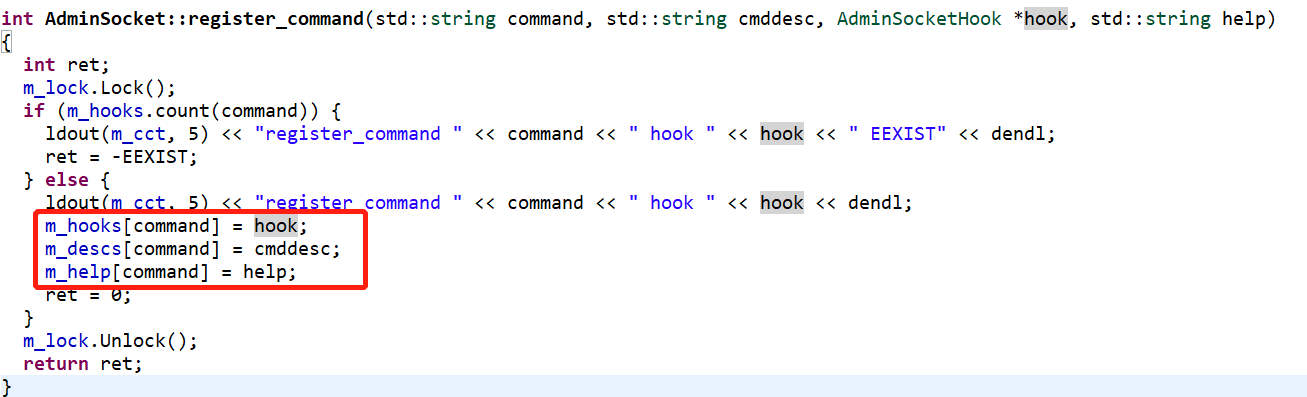
new AdminSocket只是初始化变量：



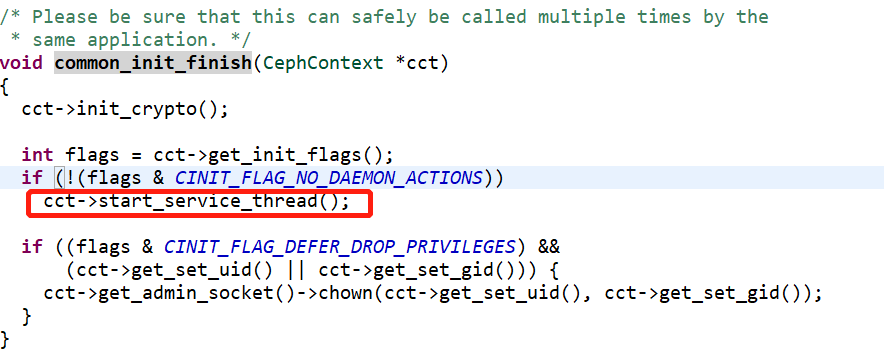
定义CephContextHook，new CephContextHook，就是一个钩子，注册命令和这个钩子对应，执行命令的时候就调用这个钩子。



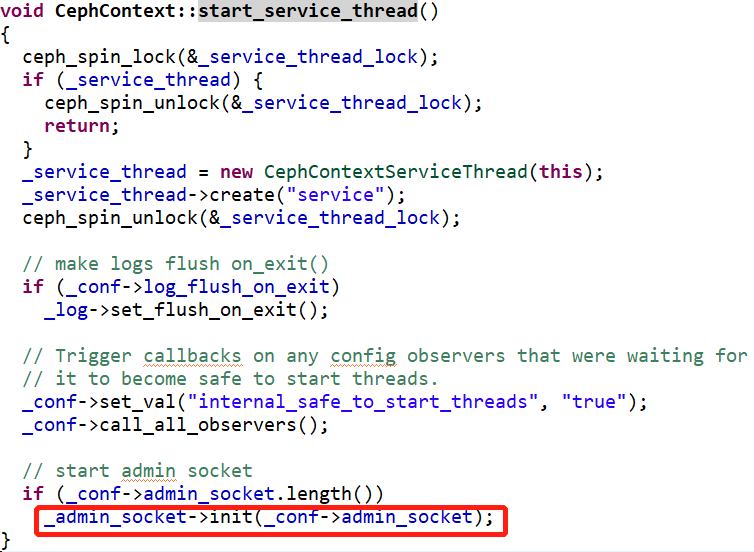
然后注册相应的命令和对应的hook：



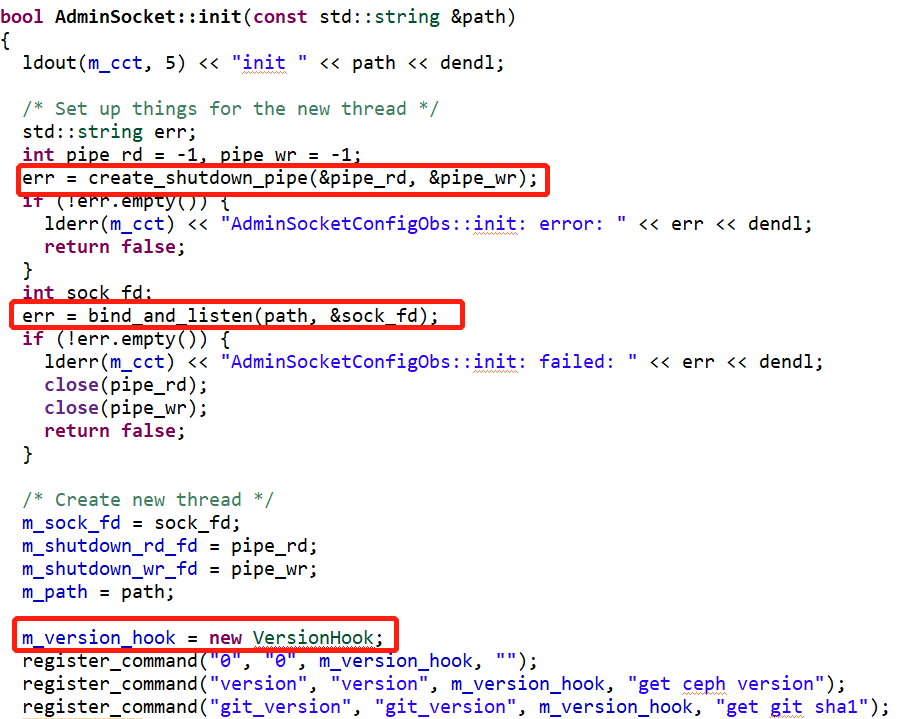
AdminSocket是一个线程，在monitor启动过程中函数common\_init\_finish(g\_ceph\_context)中启动。

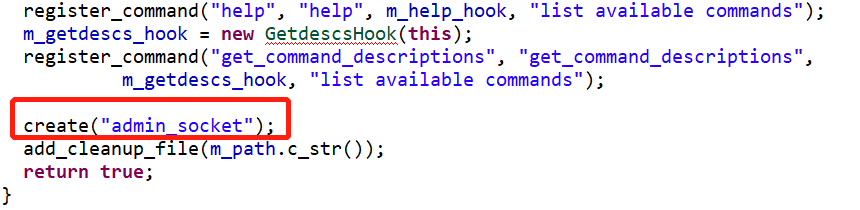


在start\_service\_thread()函数中会初始化admin socket线程

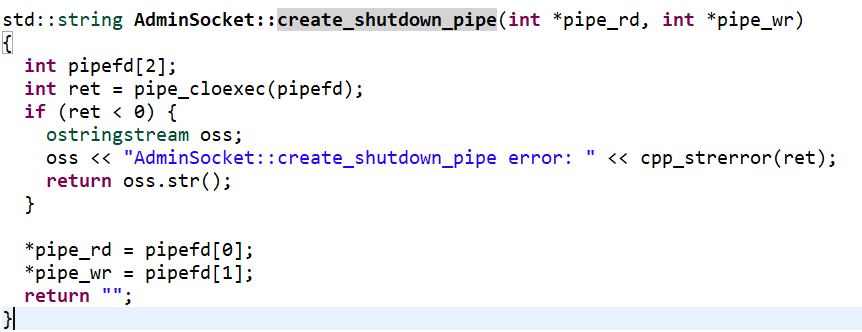


在初始化函数中，创建了管道，读取端的文件描述符记录在m\_shutdown\_rd\_fd中，写入端的文件描述符记录在m\_shutdown\_wr\_fd中。从变量名字也可以看出，该文件描述符的作用是收取关闭信息。因为adminsocket一旦创建，必须能够通知到该线程及时退出。  
退出的事情会写入管道的写入端，而线程会通过多路复用接口，监听读取端，一旦发现m\_shutdown\_rd\_fd中读出内容，线程就知道，可以退出了。

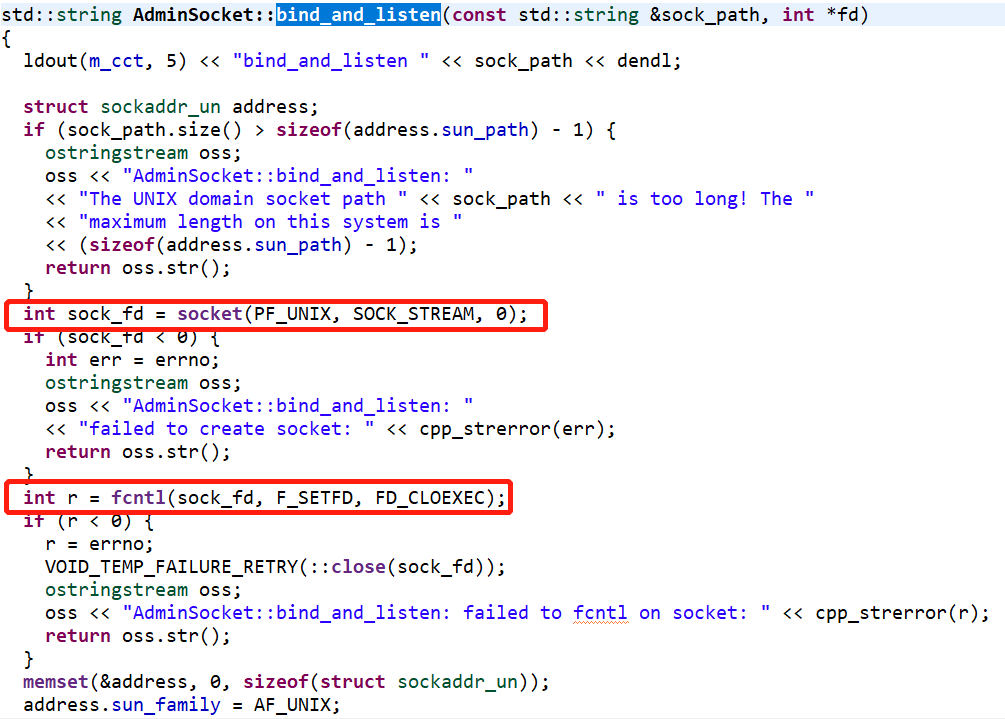


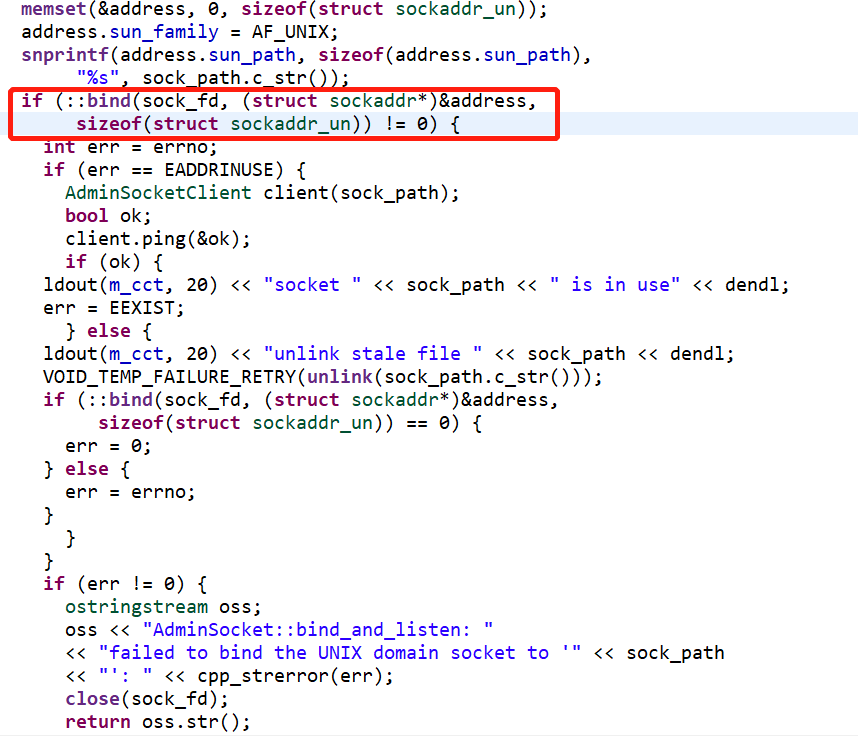


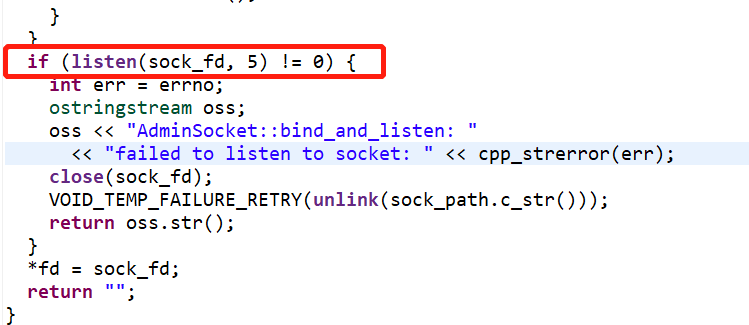
创建用于退出处理的管道：



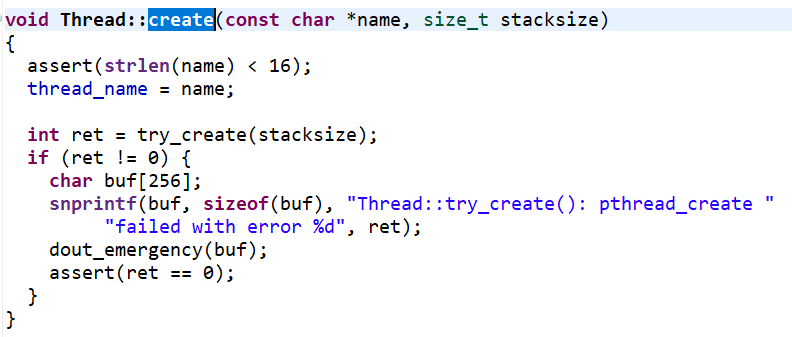
bind\_and\_listen(path, &sock\_fd)建立绑定监听socket，这个socket是unix socket。



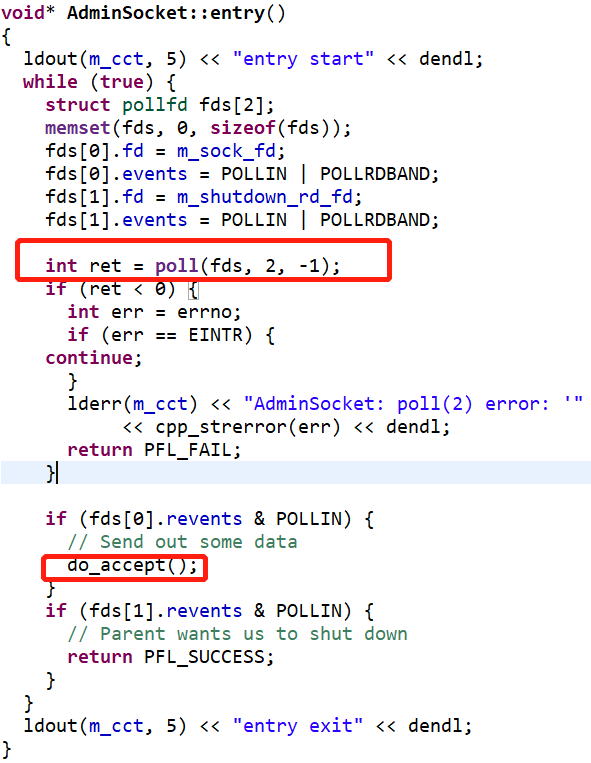




create（） 函数创建启动线程：



同系列一关于LOG的实现机制一样，这个线程调用AdminSocket::entry()函数作为线程的执行体



这个线程函数比较简单，它监听socket fd和管道的读取端。

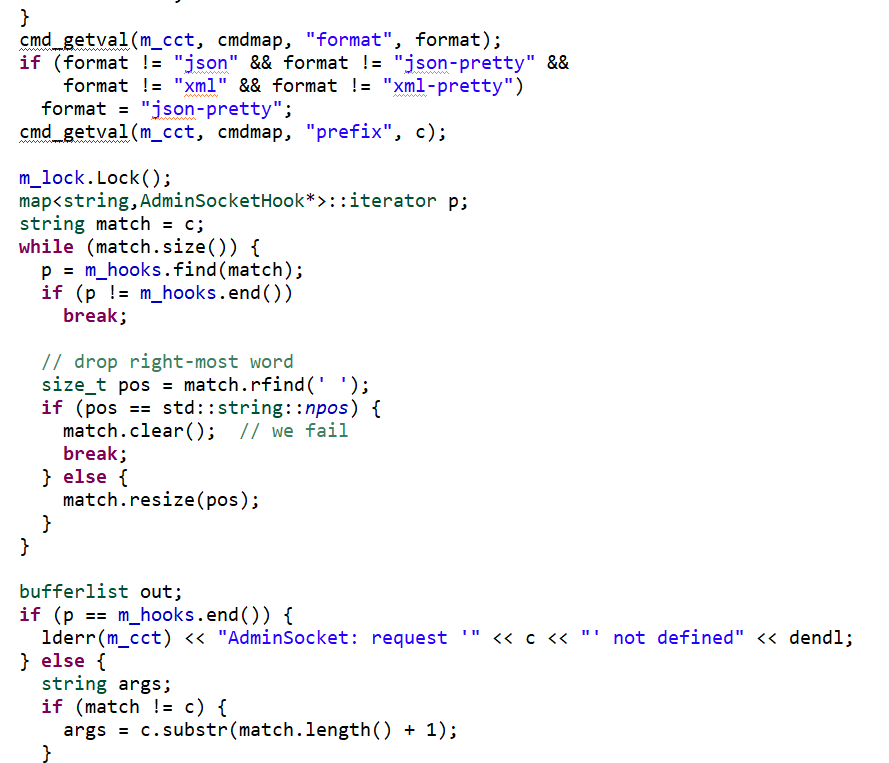
1 管道的读取端负责管理何时退出

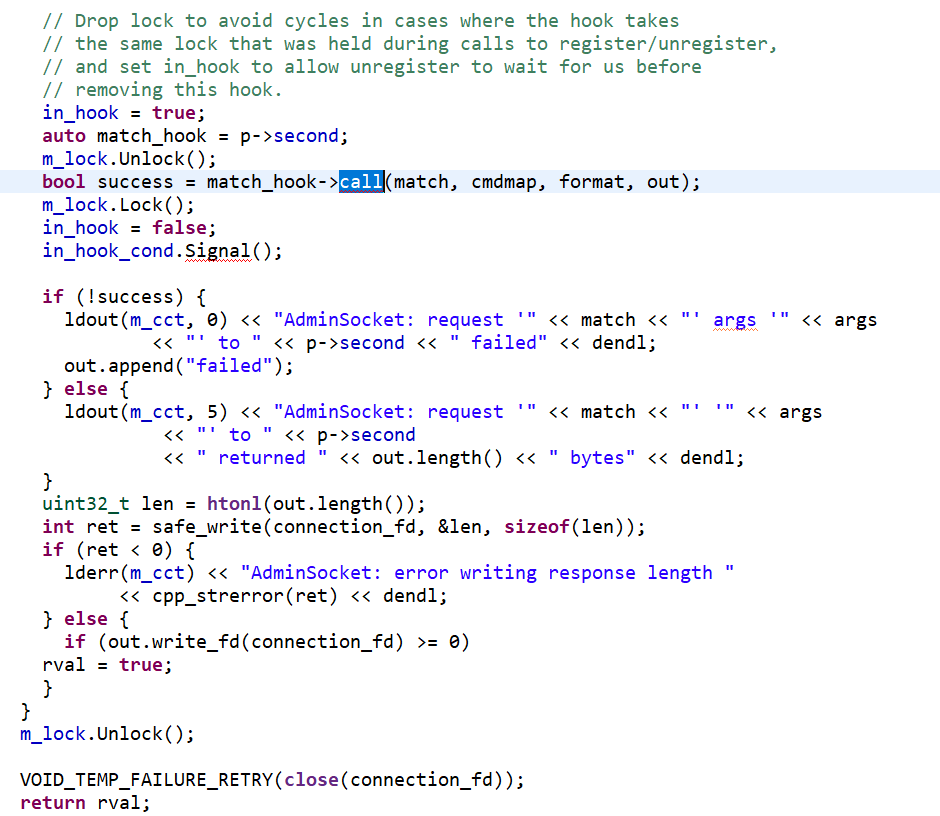
2 socket fd 负责监听用户发过来的指令。

处理用户发过来的命令，是do\_accept函数要处理的工作。









这个函数就是实现：

这个函数有点长，但是并不复杂。简单说，如果有个client尝试 connect ，该线程就poll就会感知到，然后进入do\_accept函数。

do\_accept首先执行accept，和client 搭上线，然后开始通信。

safe\_read负责 读取客户发过来的指令。前面已经提到过，AdminSocket支持的命令是有限的，初始化之前都已注册过了。

如果client 发来的指令时注册过的指令，就见招拆招，返回相应的结果给客户端。

每一个命令的字符串，都是和一个AdminSocketHook 的类型关联的，但是一个AdminSocketHook可以对应多个command

std::map<std::string,AdminSocketHook\*> m\_hooks

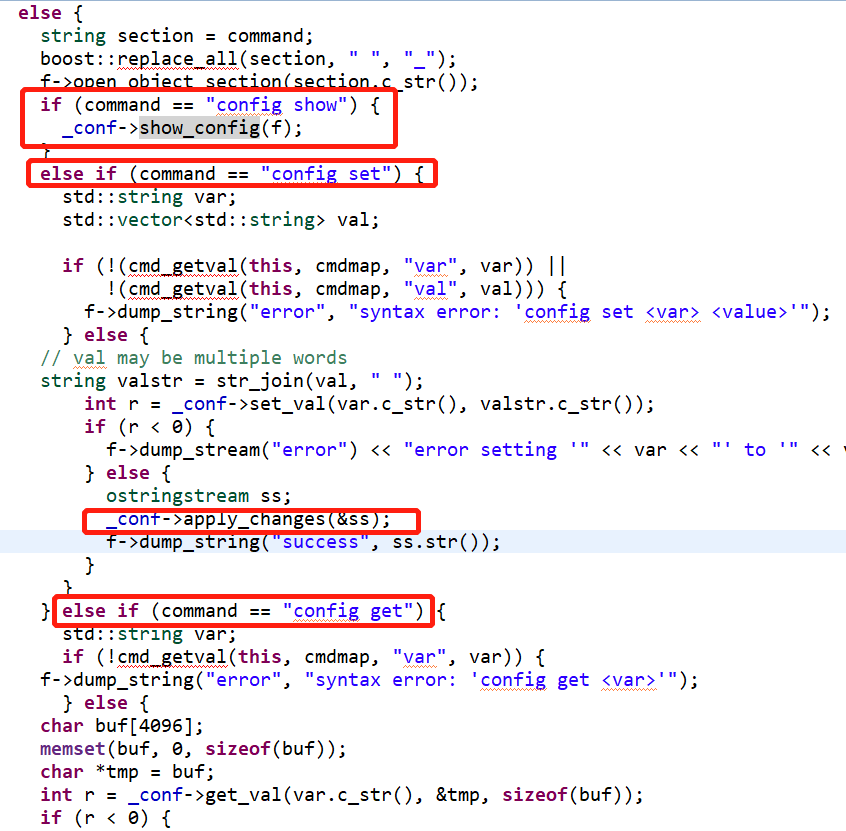
最终的执行则在相应的hook上：



即调用CephContext的do\_command函数：



在这个函数中解析命令，并调用相应的处理函数：截取部分命令分支的处理：



前面注册的所有命令都会在这里找到相应的分支进行处理。