android 6.0 对vold的改动比较大，在此记录学习过程。
在vold作为一个守护进程，一方面接受驱动的信息，并把信息传给应用层；另一方面接受上层的命令并完成相应操作
1. vold的启动
在android 6.0之前，vold在init.rc中启动的代码如下

```
service vold /system/bin/vold
    class core
    socket vold stream 0660 root mount
    ioprio be 2
```

但是在android 6.0中，vold的启动使用了更多的参数以及配置

```
service vold /system/bin/vold \
        --blkid_context=u:r:blkid:s0 --blkid_untrusted_context=u:r:blkid_untrusted:s0 \
        --fsck_context=u:r:fsck:s0 --fsck_untrusted_context=u:r:fsck_untrusted:s0
    class core
    socket vold stream 0660 root mount
    socket cryptd stream 0660 root mount
    ioprio be 2  //设置IO的优先级，0--7
```

启动时附件的参数暂时不明白是什么意思，但是启动的时候新创建了一个socket（cryptd），猜测是为分区加密锁准备的socket，后续在机器加密的过程中学习。
2. vold 的main函数分析

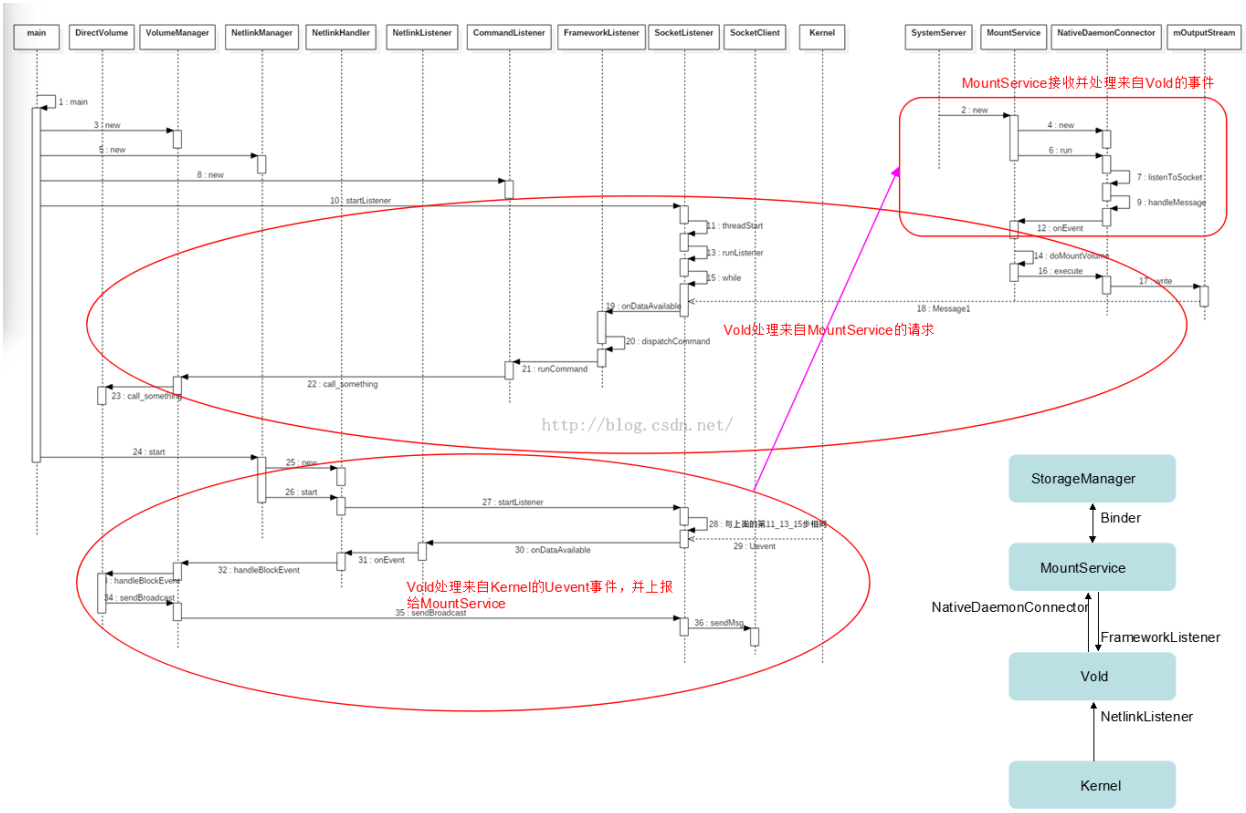
```

int main(int argc, char** argv) {
    setenv("ANDROID_LOG_TAGS", "*:v", 1);
    android::base::InitLogging(argv, android::base::LogdLogger(android::base::SYSTEM));
    LOG(VERBOSE) << "Detected support for:"
            << (android::vold::IsFilesystemSupported("ext4") ? " ext4" : "")
            << (android::vold::IsFilesystemSupported("f2fs") ? " f2fs" : "")
            << (android::vold::IsFilesystemSupported("vfat") ? " vfat" : "");
     //从这些信息中可以看到有些文件格式并不支持挂载，如果要挂载，需要一些其它的修改来配合
    VolumeManager *vm;
    CommandListener *cl;

    CryptCommandListener *ccl;
    NetlinkManager *nm;   //用来监听kernel上报的uevent事件
    parse_args(argc, argv);
    sehandle = selinux_android_file_context_handle();
    if (sehandle) {
        selinux_android_set_sehandle(sehandle);
    }
    // Quickly throw a CLOEXEC on the socket we just inherited from init
    fcntl(android_get_control_socket("vold"), F_SETFD, FD_CLOEXEC);
    fcntl(android_get_control_socket("cryptd"), F_SETFD, FD_CLOEXEC);
    mkdir("/dev/block/vold", 0755);
     //创建/dev/block/vold目录，在这个目录下存放sd卡的挂载点
    /* For when cryptfs checks and mounts an encrypted filesystem */
```

```
23.       klog_set_level(6);
24.       /* Create our singleton managers */
25.       if (!(vm = VolumeManager::Instance())) {
26.           LOG(ERROR) << "Unable to create VolumeManager";
27.           exit(1);
28.       }
29.       if (!(nm = NetlinkManager::Instance())) {
30.           LOG(ERROR) << "Unable to create NetlinkManager";
31.           exit(1);
32.       }
33.       if (property_get_bool("vold.debug", false)) {
34.           vm->setDebug(true);
35.       }
36.       cl = new CommandListener();
37.       ccl = new CryptCommandListener();
38.       vm->setBroadcaster((SocketListener *) cl);
39.       nm->setBroadcaster((SocketListener *) cl);
40.       if (vm->start()) {
41.           PLOG(ERROR) << "Unable to start VolumeManager";
42.           exit(1);
43.       }
44.       if (process_config(vm)) {                                                    //
      这个函数会去解析vold.fstab文件，创建需要的挂载点，
45.           PLOG(ERROR) << "Error reading configuration... continuing anyways";
46.       }
47.       if (nm->start()) {
48.           PLOG(ERROR) << "Unable to start NetlinkManager";
49.           exit(1);
50.       }
51.       coldboot("/sys/block");                                                       /
      /向这个目录下的所有文件写入add\n，这样就会触发uevent事件来挂载这个目录下的所有设备
52.       /*
53.        * Now that we're up, we can respond to commands
54.        */
55.       if (cl->startListener()) {
56.           PLOG(ERROR) << "Unable to start CommandListener";
57.           exit(1);
58.       }
59.       if (ccl->startListener()) {
60.           PLOG(ERROR) << "Unable to start CryptCommandListener";
61.           exit(1);
62.       }
63.       // Eventually we'll become the monitoring thread
64.       while(1) {
65.           sleep(1000);
66.       }
67.       exit(0);
68.   }
```

下面看一张vold的流程图：



从这张图中可以看到vold与framework的交过过程。