Vold(Volume Deamon) 外设存储设备管理

Vold启动：

Vold服务由volumeManager统一管控，它将具体任务分别分派给netlinkManager, commandListener, directVolume, Volume去完成。

Vold服务向下通过socket机制与底层驱动交互，向上通过JNI, intent, socket, doCommand等机制与Java Framework交互。

Android 启动时会启用Vold服务进程，代码从main.cpp开始执行。

vold启动在init.rc中：

service vold /system/bin/vold

class core

socket vold stream 0660 root mount//创建一个socket，与framework通信，在注册CommandListener的时候用到该socket

ioprio be 2

netlinkManager:与内核通信，接受事件消息。

commandListener:与framework通信，接受命令。

cl = new CommandListener();

vm->setBroadcaster((SocketListener \*) cl);

nm->setBroadcaster((SocketListener \*) cl);

socket（vold）实例化，分别赋值VM NM。

process\_config函数：解析vold.fstab

"Volume %s state changing %d (%s) -> %d (%s)"

DirectVolume：向framework层发送消息。

vm->addVolume(dv)添加到VM，由VM统一管理。

nm->start() ，mHandler->start()开始监听内核消息

(pthread\_create(&mThread, NULL, SocketListener::threadStart, this)) threadStart调用runListener，当有内核消息传来时会调用onDataAvailable(c)调用vm->handleBlockEvent(evt) -> DirectVolume -> setState

cl->startListener()，启动commandlistener，socketname为vold。

Mountservice启动：

new一个NativeDaemonConnector，传递了一个"vold"字符串，跟vold启动的时候传给CommandListener是一样的。调用connect函数连接到vold模块。

Mountservice和netlinkmanager的listener均启动完毕，等待usb挂载。。。。。。。。

接受内核信息：

当某个套接字上有数据到来时，首先看这个套接字是不是listen的那个套接字，如果是则接收，连接并加到mClients链表中，这时是我们注册到内核的那个套接字，调用onDataAvailable函数，这里由于多态调用的是NetlinkListener::onDataAvailable中的这个函数。

runListener：当有内核消息传来时会调用onDataAvailable(c)调用onevent，subsys是block所以调用vm->handleBlockEvent(evt) -> DirectVolume:: handleBlockEvent

/\* Lookup a volume to handle this device \*/

(\*it)->handleBlockEvent(evt);

之前有注册一个directvolume，add后，这里调用DirectVolume

:: handleBlockEvent,检测到disk插入，所以调用DirectVolume::handleDiskAdded。handlePartitionAdded,通过vm socketlistener （vold）发送广播，由MountService.java收到并处理广播的信息。再通过"vold"socket 到Vold中，在SocketListener 中会接收到，依次会调用到FrameworkListener.cpp中的onDataAvailable，dispatchCommand

再到CommandListener.cpp中的runCommand，VolumeManager.cpp中的mountVolume。调用到Volume的mountVol

handleBlockEvent()完成SD的挂载和卸载，具体交由DirectVolume完成。

handleSwitchEvent()完成由USB实现的U盘的连接（mstar没有handleSwitchEvent）。这里是vold部分从内核获取信息后的mount动作。

两者最后都是通过setBroadcast()将ResponsibleCode经过nativeDaemonConnector的socket监听机制，最终上传到MountService作统一规划。

MountService（framework）里的socketlistener -> runListener -> FrameworkListener::onDataAvailable -> FrameworkListener::dispatchCommand -> CommandListener::VolumeCmd::runCommand -> vm.mountVolume-> v->mountVol -> extractMetadata -> setUserLabel(value)

MountService得到vold解析后的信息，执行下面的任务：

-> 发送命令doCommand()通过commandListener传递给volumeManager

-> 将事件信息广播给相关服务，供上层应用使用。

**Framework mountservice.java**

NativeDaemonConnector(this, "vold", MAX\_CONTAINERS \* 2, VOLD\_TAG, 25);//创建 vold 的监听接收，用于接收system中Vold的socket消息

MountService下发Command:

就从上面doMountVolume来解析，MountService中对Volume的各种操作都是需要转换成符合Vold中socket command，这样Vold中才能正确的解析识别调用

/system/bin/blkid -c /dev/null /dev/block/vold/8:1