目录

[USB 1](#_Toc492900641)

[前言 1](#_Toc492900642)

[配置USB设备 1](#_Toc492900643)

[USB驱动开发 3](#_Toc492900644)

[Usb设备插入检测 4](#_Toc492900645)

[Android流程 4](#_Toc492900646)

[UsbHostManager 5](#_Toc492900647)

[UsbHostManager的hal层 6](#_Toc492900648)

[UsbManager中调用usb设备 15](#_Toc492900649)

[UsbDeviceManager 21](#_Toc492900650)

[参考资料 41](#_Toc492900651)

# USB

## 前言

**USB是英文"Universal Serial Bus"的缩写，意为"通用串行总线"**。是由Compaq(康柏)、DEC、IBM、Intel、NEC、微软以及Northern Telecom（北方电讯）等公司于1994年11月共同提出的，**主要目的就是为了解决接口标准太多的弊端**。USB使用一个4针插头作为标准插头，并通过这个标准接头，采用菊花瓣形式把所有外设连接起来，它**采用串行方式传输数据**，**支持多数据流和多个设备并行操作**，**允许外设热插拔**，**USB2.0**的最大传输带宽为**480**[Mbps](https://baike.baidu.com/item/Mbps)**（即60**[MB](https://baike.baidu.com/item/MB/5022)**/s）**，而**USB3.0**的最大传输带宽高达**5.0Gbps（640MB/s）**。

## 配置USB设备

**内核中配置**

make menuconfig—>Device Drivers-->USB support-->USB\_xxx

需要选择用于系统的正确 USB 主控制器驱动程序。选项是"EHCI" （对应模块为ehci-hcd.o）、"UHCI" （对应模块为usb-uhci.o）、"UHCI (alternate driver)"和"OHCI" （对应模块为usb-ohci.o）

**USB鼠标：**

键盘和鼠标属于低速的输入设备，对于已经为用户认可的PS/2接口，USB键盘和USB鼠标似乎并没有太多更优越的地方。现在的大部分鼠标采用了PS/2接口，不过USB接口的鼠标也越来越多，两者相比，各有优势：一般来说，USB的鼠标接口的带宽大于PS/2鼠标，也就是说在同样的时间内，USB鼠标扫描次数就要多于PS/2鼠标，这样在定位上USB鼠标就更为精确；同时USB接口鼠标的默认采样率也比较高，达到125HZ，而PS/2接口的鼠标仅有40HZ（Windows 9x/Me）或是60HZ（Windows NT/2000）。

对于USB设备你当然必须先插入相应的USB控制器模块：usb-uhci.o或usb-ohci.o

modprobe usb-uhci

USB鼠标为了使其正常工作，您必须先插入模块usbmouse.o和mousedev.o

modprobe usbmouse

modprobe mousedev

若你把HID input layer支持和input core 支持也作为模块方式安装，那么启动hid模块和input模块也是必要的。

modprobe hid

modprobe input

**USB键盘：**

一般的，我们现在使用的键盘大多是PS/2的，USB键盘还比较少见，但是下来的发展，键盘将向USB接口靠拢。使用USB键盘基本上没有太多的要求，只需在主板的BIOS设定对USB键盘的支持，就可以在各系统中完全无障碍的使用，而且更可以真正做到在即插即用和热插拔使用,并能提供两个USB连接埠：让您可以轻易地直接将具有USB接头的装置接在您的键盘上，而非计算机的后面。

同样你当然必须先插入相应的USB控制器模块：usb-uhci.o或usb-ohci.o

modprobe usb-uhci

然后您还必须插入键盘模块usbkbd.o，以及keybdev.o，这样usb键盘才能够正常工作。此时，运行的系统命令：

modprobe usbkbd

modprobe keybdev

同样若你把HID input layer支持和input core 支持也作为模块方式安装，那么启动hid模块和input模块也是必要的。

U盘和USB读卡器：

数码存储设备现在对我们来说已经是相当普遍的了。CF卡、SD卡、Memory Stick等存储卡已经遍及我们的身边，通常，他们的读卡器都是USB接口的。另外，很多MP3、数码相机也都是USB接口和计算机进行数据传递。更我们的U盘、USB硬盘，作为移动存储设备，已经成为我们的必须装备。

在Linux下这些设备通常都是以一种叫做usb-storage的方式进行驱动。要使用他们必须加载此模块

modprobe usb-storage

当然，usbcore.o 和usb-uhci.o或usb-ohci也肯定是不可缺少的。另外，若你系统中SCSI支持也是模块方式，那么下面的模块也要加载

modprobe scsi\_mod

modprobe sd\_mod

在加载完这些模块后，我们插入U盘或存储卡，就会发现系统中多了一个SCSI硬盘，通过正确地mount它，就可以使用了（SCSI硬盘一般为/dev/sd?，可参照文章后面的常见问题解答）。

mount /dev/sda1 /mnt

**Linux支持的其他USB设备。**

MODEM--（比较常见）

网络设备

摄像头--（比较常见）例如ov511.o

联机线--可以让你的两台电脑用USB线实现网络功能。usbnet.o

显示器--（我没见过）

游戏杆

电视盒--（比较常见）

手写板--（比较常见）

扫描仪--（比较常见）

刻录机--（比较常见）

打印机--（比较常见）

注意：**上面所说的每个驱动模块，并不是都要手动加载，有很多系统会在启动或你的应用需要时自动加载的，写明这些模块，是便于你在不能够使用USB设备时，可以自行检查。**只要用lsmod确保以上模块已经被系统加载，你的设备就应该可以正常工作了。当然注意有些模块已经以内核方式在kernel启动时存在了（这些模块文件在/lib/modules/2.4.XX中是找不到的）。

## USB驱动开发

**USB骨架**

**在Linux kernel源码目录中driver/usb/usb-skeleton.c为我们提供了一个最基础的USB驱动程序。**我们称为USB骨架。通过它我们仅需要修改极少的部分，就可以完成一个USB设备的驱动。

## Usb设备插入检测

### Android流程

#### SystemServer启动UsbService

./frameworks/base/services/java/com/android/server/SystemServer.java

UsbService usb = null;

......

|  |
| --- |
| try { |
|  | Slog.i(TAG, "USB Service"); |
|  | // Manage USB host and device support |
|  | usb = new UsbService(context); |
|  | ServiceManager.addService(Context.USB\_SERVICE, usb); |
|  | } catch (Throwable e) { |
|  | reportWtf("starting UsbService", e); |
|  | } |

**在SystemServer里面开启Usbservice服务**

./frameworks/base/services /java/com/android/server/usb/UsbService.java

[106](http://androidxref.com/7.1.1_r6/xref/frameworks/base/services/usb/java/com/android/server/usb/UsbService.java" \l "106) public [UsbService](http://androidxref.com/7.1.1_r6/s?refs=UsbService&project=frameworks)([Context](http://androidxref.com/7.1.1_r6/s?defs=Context&project=frameworks) [context](http://androidxref.com/7.1.1_r6/s?refs=context&project=frameworks)) {

[107](http://androidxref.com/7.1.1_r6/xref/frameworks/base/services/usb/java/com/android/server/usb/UsbService.java" \l "107) [mContext](http://androidxref.com/7.1.1_r6/xref/frameworks/base/services/usb/java/com/android/server/usb/UsbService.java#mContext) = [context](http://androidxref.com/7.1.1_r6/s?defs=context&project=frameworks);

[108](http://androidxref.com/7.1.1_r6/xref/frameworks/base/services/usb/java/com/android/server/usb/UsbService.java" \l "108)

[109](http://androidxref.com/7.1.1_r6/xref/frameworks/base/services/usb/java/com/android/server/usb/UsbService.java" \l "109) [mAlsaManager](http://androidxref.com/7.1.1_r6/xref/frameworks/base/services/usb/java/com/android/server/usb/UsbService.java#mAlsaManager) = new [UsbAlsaManager](http://androidxref.com/7.1.1_r6/s?defs=UsbAlsaManager&project=frameworks)([context](http://androidxref.com/7.1.1_r6/s?defs=context&project=frameworks));

[110](http://androidxref.com/7.1.1_r6/xref/frameworks/base/services/usb/java/com/android/server/usb/UsbService.java" \l "110)

[111](http://androidxref.com/7.1.1_r6/xref/frameworks/base/services/usb/java/com/android/server/usb/UsbService.java" \l "111) final [PackageManager](http://androidxref.com/7.1.1_r6/s?defs=PackageManager&project=frameworks) [pm](http://androidxref.com/7.1.1_r6/s?defs=pm&project=frameworks) = [mContext](http://androidxref.com/7.1.1_r6/xref/frameworks/base/services/usb/java/com/android/server/usb/UsbService.java#mContext).[getPackageManager](http://androidxref.com/7.1.1_r6/s?defs=getPackageManager&project=frameworks)();

[112](http://androidxref.com/7.1.1_r6/xref/frameworks/base/services/usb/java/com/android/server/usb/UsbService.java" \l "112) if ([pm](http://androidxref.com/7.1.1_r6/s?defs=pm&project=frameworks).[hasSystemFeature](http://androidxref.com/7.1.1_r6/s?defs=hasSystemFeature&project=frameworks)([PackageManager](http://androidxref.com/7.1.1_r6/s?defs=PackageManager&project=frameworks).[FEATURE\_USB\_HOST](http://androidxref.com/7.1.1_r6/s?defs=FEATURE_USB_HOST&project=frameworks))) {

[113](http://androidxref.com/7.1.1_r6/xref/frameworks/base/services/usb/java/com/android/server/usb/UsbService.java" \l "113) [**mHostManager**](http://androidxref.com/7.1.1_r6/xref/frameworks/base/services/usb/java/com/android/server/usb/UsbService.java#mHostManager) **= new** [**UsbHostManager**](http://androidxref.com/7.1.1_r6/s?defs=UsbHostManager&project=frameworks)**(**[**context**](http://androidxref.com/7.1.1_r6/s?defs=context&project=frameworks)**,** [**mAlsaManager**](http://androidxref.com/7.1.1_r6/xref/frameworks/base/services/usb/java/com/android/server/usb/UsbService.java#mAlsaManager)**);**

[114](http://androidxref.com/7.1.1_r6/xref/frameworks/base/services/usb/java/com/android/server/usb/UsbService.java" \l "114) }

[115](http://androidxref.com/7.1.1_r6/xref/frameworks/base/services/usb/java/com/android/server/usb/UsbService.java" \l "115) if (new [File](http://androidxref.com/7.1.1_r6/s?defs=File&project=frameworks)("/[sys](http://androidxref.com/7.1.1_r6/s?path=/sys/)/[class](http://androidxref.com/7.1.1_r6/s?path=/sys/class/)/[android\_usb](http://androidxref.com/7.1.1_r6/s?path=/sys/class/android_usb)").[exists](http://androidxref.com/7.1.1_r6/s?defs=exists&project=frameworks)()) {

[116](http://androidxref.com/7.1.1_r6/xref/frameworks/base/services/usb/java/com/android/server/usb/UsbService.java" \l "116) [**mDeviceManager**](http://androidxref.com/7.1.1_r6/xref/frameworks/base/services/usb/java/com/android/server/usb/UsbService.java#mDeviceManager) **= new** [**UsbDeviceManager**](http://androidxref.com/7.1.1_r6/s?defs=UsbDeviceManager&project=frameworks)**(**[**context**](http://androidxref.com/7.1.1_r6/s?defs=context&project=frameworks)**,** [**mAlsaManager**](http://androidxref.com/7.1.1_r6/xref/frameworks/base/services/usb/java/com/android/server/usb/UsbService.java#mAlsaManager)**);**

[117](http://androidxref.com/7.1.1_r6/xref/frameworks/base/services/usb/java/com/android/server/usb/UsbService.java" \l "117) }

[118](http://androidxref.com/7.1.1_r6/xref/frameworks/base/services/usb/java/com/android/server/usb/UsbService.java" \l "118) if ([mHostManager](http://androidxref.com/7.1.1_r6/xref/frameworks/base/services/usb/java/com/android/server/usb/UsbService.java#mHostManager) != [null](http://androidxref.com/7.1.1_r6/s?defs=null&project=frameworks) || [mDeviceManager](http://androidxref.com/7.1.1_r6/xref/frameworks/base/services/usb/java/com/android/server/usb/UsbService.java#mDeviceManager) != [null](http://androidxref.com/7.1.1_r6/s?defs=null&project=frameworks)) {

[119](http://androidxref.com/7.1.1_r6/xref/frameworks/base/services/usb/java/com/android/server/usb/UsbService.java" \l "119) [mPortManager](http://androidxref.com/7.1.1_r6/xref/frameworks/base/services/usb/java/com/android/server/usb/UsbService.java#mPortManager) = new [UsbPortManager](http://androidxref.com/7.1.1_r6/s?defs=UsbPortManager&project=frameworks)([context](http://androidxref.com/7.1.1_r6/s?defs=context&project=frameworks));

[120](http://androidxref.com/7.1.1_r6/xref/frameworks/base/services/usb/java/com/android/server/usb/UsbService.java" \l "120) }

**在UsbService.java的UsbService函数中new了**[**UsbHostManager**](http://androidxref.com/7.1.1_r6/s?defs=UsbHostManager&project=frameworks)**以及**[**UsbDeviceManager**](http://androidxref.com/7.1.1_r6/s?defs=UsbDeviceManager&project=frameworks)

./frameworks/base/services/usb/java/com/android/server/usb/UsbHostManager.java

./frameworks/base/services/usb/java/com/android/server/usb/UsbDeviceManager.java

UsbDeviceManager和UsbHostManager是一个相对的概念，UsbHostManager是手机作为一个host，比如键盘、鼠标通过usb连接手机。而UsbDeviceManager是手机与电脑连接。

**然后在UsbService的systemReady中调用了UsbHostManager以及UsbDeviceManager的systemReady函数**

1. **public** **void** systemReady() {
2. mAlsaManager.systemReady();
4. **if** (mDeviceManager != null) {
5. mDeviceManager.systemReady();
6. }
7. **if** (mHostManager != null) {
8. mHostManager.systemReady();
9. }
10. **if** (mPortManager != null) {
11. mPortManager.systemReady();
12. }
13. }

### UsbHostManager

UsbHostManager的构造函数就是新建一些对象，我们直接看**systemReady函数在新的线程中调用了monitorUsbHostBus函数**

1. **public** **void** systemReady() {
2. synchronized (mLock) {
3. // Create a thread to call into native code to wait for USB host events.
4. // This thread will call us back on usbDeviceAdded and usbDeviceRemoved.
5. Runnable runnable = **new** Runnable() {
6. **public** **void** run() {
7. monitorUsbHostBus();
8. }
9. };
10. **new** Thread(null, runnable, "UsbService host thread").start();
11. }
12. }

**monitorUsbHostBus函数是一个JNI函数**

1. **private** native **void** monitorUsbHostBus();

### UsbHostManager的hal层

**monitorUsbHostBus对应的JNI函数是在com\_android\_server\_UsbHostManager.cpp的android\_server\_UsbHostManager\_monitorUsbHostBus函数**，**在这个函数调用了usb\_host\_init函数，创建了一个INotify的fd，以及创建了一个usb\_host\_context对象。usb\_host\_run函数就是循环读取INotify的fd的事件，我们把usb\_device\_added, usb\_device\_removed两个回调函数也传入了usb\_host\_run函数**

1. **static** **void** android\_server\_UsbHostManager\_monitorUsbHostBus(JNIEnv\* /\* env \*/, jobject thiz)
2. {
3. **struct** usb\_host\_context\* context = usb\_host\_init();
4. **if** (!context) {
5. ALOGE("usb\_host\_init failed");
6. **return**;
7. }
8. // this will never return so it is safe to pass thiz directly
9. usb\_host\_run(context, usb\_device\_added, usb\_device\_removed, NULL, (**void** \*)thiz);
10. }

**usb\_host\_init是在system\core\libusbhost\usbhost.c文件中，这个函数中新建一个usb\_host\_context对象，还有新建了一个INotify，并且usb\_host\_context的fd就是INotify的fd。**

1. **struct** usb\_host\_context \*usb\_host\_init()
2. {
3. **struct** usb\_host\_context \*context = calloc(1, **sizeof**(**struct** usb\_host\_context));//新建一个usb\_host\_context对象
4. **if** (!context) {
5. fprintf(stderr, "out of memory in usb\_host\_context\n");
6. **return** NULL;
7. }
8. context->fd = inotify\_init();//新建一个INotify
9. **if** (context->fd < 0) {
10. fprintf(stderr, "inotify\_init failed\n");
11. free(context);
12. **return** NULL;
13. }
14. **return** context;
15. }

我们再来看看**usb\_host\_run函数。先调用了usb\_host\_load函数，这个函数主要把add和remove的回调，放到context相应的成员变量中，然后增加了dev目录放入INotify的观察。下面循环调用usb\_host\_read\_event函数去读取INotify fd的事件**。

1. **void** usb\_host\_run(**struct** usb\_host\_context \*context,
2. usb\_device\_added\_cb added\_cb,
3. usb\_device\_removed\_cb removed\_cb,
4. usb\_discovery\_done\_cb discovery\_done\_cb,
5. **void** \*client\_data)
6. {
7. **int** done;
9. done = usb\_host\_load(context, added\_cb, removed\_cb, discovery\_done\_cb, client\_data);
11. **while** (!done) {
13. done = usb\_host\_read\_event(context);
14. }
15. }

**usb\_host\_load函数先把add和remove的两个回调设置到usb\_host\_context 中，然后将dev目录放入INotify观察。然后调用watch\_existing\_subdirs将/dev/bus/usb下的目录都添加到INotify中观察，最后再调用find\_existing\_devices函数，把找到的设备调用added\_cb（增加的回调函数）**

1. **int** usb\_host\_load(**struct** usb\_host\_context \*context,
2. usb\_device\_added\_cb added\_cb,
3. usb\_device\_removed\_cb removed\_cb,
4. usb\_discovery\_done\_cb discovery\_done\_cb,
5. **void** \*client\_data)
6. {
7. **int** done = 0;
8. **int** i;
10. context->cb\_added = added\_cb;//回调赋值
11. context->cb\_removed = removed\_cb;
12. context->data = client\_data;
14. D("Created device discovery thread\n");
16. /\* watch for files added and deleted within USB\_FS\_DIR \*/
17. context->wddbus = -1;
18. **for** (i = 0; i < MAX\_USBFS\_WD\_COUNT; i++)
19. context->wds[i] = -1;
21. /\* watch the root for new subdirectories \*/
22. context->wdd = inotify\_add\_watch(context->fd, DEV\_DIR, IN\_CREATE | IN\_DELETE);//将dev目录放入INotify观察
23. **if** (context->wdd < 0) {
24. fprintf(stderr, "inotify\_add\_watch failed\n");
25. **if** (discovery\_done\_cb)
26. discovery\_done\_cb(client\_data);
27. **return** done;
28. }
30. watch\_existing\_subdirs(context, context->wds, MAX\_USBFS\_WD\_COUNT);
32. /\* check for existing devices first, after we have inotify set up \*/
33. done = find\_existing\_devices(added\_cb, client\_data);
34. **if** (discovery\_done\_cb)
35. done |= discovery\_done\_cb(client\_data);
37. **return** done;
38. }

**watch\_existing\_subdirs，就是将/dev/bus/usb下的设备目录（001开始到MAX\_USBFS\_WD\_COUNT值都inotify\_add\_watch）都添加到INotify中去。**

1. **static** **void** watch\_existing\_subdirs(**struct** usb\_host\_context \*context,
2. **int** \*wds, **int** wd\_count)
3. {
4. **char** path[100];
5. **int** i, ret;
7. wds[0] = inotify\_add\_watch(context->fd, USB\_FS\_DIR, IN\_CREATE | IN\_DELETE);
8. **if** (wds[0] < 0)
9. **return**;
11. /\* watch existing subdirectories of USB\_FS\_DIR \*/
12. **for** (i = 1; i < wd\_count; i++) {
13. snprintf(path, **sizeof**(path), USB\_FS\_DIR "/%03d", i);
14. ret = inotify\_add\_watch(context->fd, path, IN\_CREATE | IN\_DELETE);
15. **if** (ret >= 0)
16. wds[i] = ret;
17. }
18. }

**然后我们再来看find\_existing\_devices函数，就是遍历dev/bus/usb的目录然后再调用find\_existing\_devices\_bus函数**

1. **static** **int** find\_existing\_devices(usb\_device\_added\_cb added\_cb,
2. **void** \*client\_data)
3. {
4. **char** busname[32];
5. DIR \*busdir;
6. **struct** dirent \*de;
7. **int** done = 0;
9. busdir = opendir(USB\_FS\_DIR);
10. **if**(busdir == 0) **return** 0;
12. **while** ((de = readdir(busdir)) != 0 && !done) {
13. **if**(badname(de->d\_name)) **continue**;
15. snprintf(busname, **sizeof**(busname), USB\_FS\_DIR "/%s", de->d\_name);
16. done = find\_existing\_devices\_bus(busname, added\_cb,
17. client\_data);
18. } //end of busdir while
19. closedir(busdir);
21. **return** done;
22. }

**find\_existing\_devices\_bus函数，就是将dev/bus/usb下的目录比如001，然后001目录里面的文件，作为一个设备（组成一个devname），再调用added\_cb（增加设备的回调函数）**

1. **static** **int** find\_existing\_devices\_bus(**char** \*busname,
2. usb\_device\_added\_cb added\_cb,
3. **void** \*client\_data)
4. {
5. **char** devname[32];
6. DIR \*devdir;
7. **struct** dirent \*de;
8. **int** done = 0;
10. devdir = opendir(busname);
11. **if**(devdir == 0) **return** 0;
13. **while** ((de = readdir(devdir)) && !done) {
14. **if**(badname(de->d\_name)) **continue**;
16. snprintf(devname, **sizeof**(devname), "%s/%s", busname, de->d\_name);
17. done = added\_cb(devname, client\_data);
18. } // end of devdir while
19. closedir(devdir);
21. **return** done;
22. }

**因此总结下usb\_host\_load函数就是把增加、去除设备的回调赋值到usb\_host\_context 的相关变量中，然后增加相关目录的观察，最后查找已经存在的设备调用added\_cb（增加设备的回调函数）。**

然后我们再来看看usb\_host\_read\_event函数，这个函数就是去INotify中的fd读取相关的事件。具体分析我们就看注释，有一点我们要注意了，除了增加bus目录失败返回的done是1，其他的返回的done都是0.也就是我们的usb\_host\_run函数会在while循环中一直循环。

1. **int** usb\_host\_read\_event(**struct** usb\_host\_context \*context)
2. {
3. **struct** inotify\_event\* event;
4. **char** event\_buf[512];
5. **char** path[100];
6. **int** i, ret, done = 0;
7. **int** offset = 0;
8. **int** wd;
10. ret = read(context->fd, event\_buf, **sizeof**(event\_buf));
11. **if** (ret >= (**int**)**sizeof**(**struct** inotify\_event)) {
12. **while** (offset < ret && !done) {
13. event = (**struct** inotify\_event\*)&event\_buf[offset];
14. done = 0;
15. wd = event->wd;
16. **if** (wd == context->wdd) {
17. **if** ((event->mask & IN\_CREATE) && !strcmp(event->name, "bus")) {//增加bus目录，并且将bus目录也增加到watch中
18. context->wddbus = inotify\_add\_watch(context->fd, DEV\_BUS\_DIR, IN\_CREATE | IN\_DELETE);
19. **if** (context->wddbus < 0) {
20. done = 1;//增加bus目录失败
21. } **else** {
22. watch\_existing\_subdirs(context, context->wds, MAX\_USBFS\_WD\_COUNT);
23. done = find\_existing\_devices(context->cb\_added, context->data);
24. }
25. }
26. } **else** **if** (wd == context->wddbus) {
27. **if** ((event->mask & IN\_CREATE) && !strcmp(event->name, "usb")) {//如果是dev/bus目录的事件
28. watch\_existing\_subdirs(context, context->wds, MAX\_USBFS\_WD\_COUNT);//将dev/bus/usb目录再增加到watch
29. done = find\_existing\_devices(context->cb\_added, context->data);//然后再看看usb下面时候有设备，有就调用添加函数
30. } **else** **if** ((event->mask & IN\_DELETE) && !strcmp(event->name, "usb")) {
31. **for** (i = 0; i < MAX\_USBFS\_WD\_COUNT; i++) {
32. **if** (context->wds[i] >= 0) {
33. inotify\_rm\_watch(context->fd, context->wds[i]);//是删除事件，就把这个watch删除了
34. context->wds[i] = -1;
35. }
36. }
37. }
38. } **else** **if** (wd == context->wds[0]) {//是第一个wds，也就是dev/bus/usb目录的事件
39. i = atoi(event->name);
40. snprintf(path, **sizeof**(path), USB\_FS\_DIR "/%s", event->name);
41. D("%s subdirectory %s: index: %d\n", (event->mask & IN\_CREATE) ?
42. "new" : "gone", path, i);
43. **if** (i > 0 && i < MAX\_USBFS\_WD\_COUNT) {
44. **int** local\_ret = 0;
45. **if** (event->mask & IN\_CREATE) {
46. local\_ret = inotify\_add\_watch(context->fd, path,//我们需要把usb下新增的目录也增加到watch中
47. IN\_CREATE | IN\_DELETE);
48. **if** (local\_ret >= 0)
49. context->wds[i] = local\_ret;
50. done = find\_existing\_devices\_bus(path, context->cb\_added,//然后再看看是否有已经存在的设备
51. context->data);
52. } **else** **if** (event->mask & IN\_DELETE) {
53. inotify\_rm\_watch(context->fd, context->wds[i]);
54. context->wds[i] = -1;
55. }
56. }
57. } **else** {
58. **for** (i = 1; (i < MAX\_USBFS\_WD\_COUNT) && !done; i++) {//最后剩下的肯定是dev/bus/usb下目录的事件
59. **if** (wd == context->wds[i]) {
60. snprintf(path, **sizeof**(path), USB\_FS\_DIR "/%03d/%s", i, event->name);
61. **if** (event->mask == IN\_CREATE) {//这里就直接调用增加设备或者删除设备的回调函数。
62. D("new device %s\n", path);
63. done = context->cb\_added(path, context->data);
64. } **else** **if** (event->mask == IN\_DELETE) {
65. D("gone device %s\n", path);
66. done = context->cb\_removed(path, context->data);
67. }
68. }
69. }
70. }
72. offset += **sizeof**(**struct** inotify\_event) + event->len;//读取的字节数增加
73. }
74. }
76. **return** done;
77. }

usb\_device\_added函数会调用UsbHostManager的beginUsbDeviceAdded和endUsbDeviceAdded函数，在UsbHostManager中会新建一个UsbDevice，然后放入mDevices中。

1. **static** **int** usb\_device\_added(**const** **char** \*devname, **void**\* client\_data) {
2. **struct** usb\_descriptor\_header\* desc;
3. **struct** usb\_descriptor\_iter iter;
5. **struct** usb\_device \*device = usb\_device\_open(devname);
6. **if** (!device) {
7. ALOGE("usb\_device\_open failed\n");
8. **return** 0;
9. }
11. JNIEnv\* env = AndroidRuntime::getJNIEnv();
12. jobject thiz = (jobject)client\_data;
13. **const** usb\_device\_descriptor\* deviceDesc = usb\_device\_get\_device\_descriptor(device);
15. **char** \*manufacturer = usb\_device\_get\_manufacturer\_name(device);
16. **char** \*product = usb\_device\_get\_product\_name(device);
17. **int** version = usb\_device\_get\_version(device);
18. **char** \*serial = usb\_device\_get\_serial(device);
20. jstring deviceName = env->NewStringUTF(devname);
21. jstring manufacturerName = AndroidRuntime::NewStringLatin1(env, manufacturer);
22. jstring productName = AndroidRuntime::NewStringLatin1(env, product);
23. jstring serialNumber = AndroidRuntime::NewStringLatin1(env, serial);
25. jboolean result = env->CallBooleanMethod(thiz, method\_beginUsbDeviceAdded,
26. deviceName, usb\_device\_get\_vendor\_id(device), usb\_device\_get\_product\_id(device),
27. deviceDesc->bDeviceClass, deviceDesc->bDeviceSubClass, deviceDesc->bDeviceProtocol,
28. manufacturerName, productName, version, serialNumber);
30. env->DeleteLocalRef(serialNumber);
31. env->DeleteLocalRef(productName);
32. env->DeleteLocalRef(manufacturerName);
33. env->DeleteLocalRef(deviceName);
34. free(manufacturer);
35. free(product);
36. free(serial);
38. **if** (!result) **goto** fail;
40. usb\_descriptor\_iter\_init(device, &iter);
42. **while** ((desc = usb\_descriptor\_iter\_next(&iter)) != NULL) {
43. **if** (desc->bDescriptorType == USB\_DT\_CONFIG) {
44. **struct** usb\_config\_descriptor \*config = (**struct** usb\_config\_descriptor \*)desc;
45. **char** \*name = usb\_device\_get\_string(device, config->iConfiguration);
46. jstring configName = AndroidRuntime::NewStringLatin1(env, name);
48. env->CallVoidMethod(thiz, method\_addUsbConfiguration,
49. config->bConfigurationValue, configName, config->bmAttributes,
50. config->bMaxPower);
52. env->DeleteLocalRef(configName);
53. free(name);
54. } **else** **if** (desc->bDescriptorType == USB\_DT\_INTERFACE) {
55. **struct** usb\_interface\_descriptor \*interface = (**struct** usb\_interface\_descriptor \*)desc;
56. **char** \*name = usb\_device\_get\_string(device, interface->iInterface);
57. jstring interfaceName = AndroidRuntime::NewStringLatin1(env, name);
59. env->CallVoidMethod(thiz, method\_addUsbInterface,
60. interface->bInterfaceNumber, interfaceName, interface->bAlternateSetting,
61. interface->bInterfaceClass, interface->bInterfaceSubClass,
62. interface->bInterfaceProtocol);
64. env->DeleteLocalRef(interfaceName);
65. free(name);
66. } **else** **if** (desc->bDescriptorType == USB\_DT\_ENDPOINT) {
67. **struct** usb\_endpoint\_descriptor \*endpoint = (**struct** usb\_endpoint\_descriptor \*)desc;
69. env->CallVoidMethod(thiz, method\_addUsbEndpoint,
70. endpoint->bEndpointAddress, endpoint->bmAttributes,
71. \_\_le16\_to\_cpu(endpoint->wMaxPacketSize), endpoint->bInterval);
72. }
73. }
75. env->CallVoidMethod(thiz, method\_endUsbDeviceAdded);
77. fail:
78. usb\_device\_close(device);
79. checkAndClearExceptionFromCallback(env, \_\_FUNCTION\_\_);
81. **return** 0;
82. }

去除函数，也会调用UsbHostManager的usbDeviceRemoved函数，注意这个函数返回就是0

1. **static** **int** usb\_device\_removed(**const** **char** \*devname, **void**\* client\_data) {
2. JNIEnv\* env = AndroidRuntime::getJNIEnv();
3. jobject thiz = (jobject)client\_data;
5. jstring deviceName = env->NewStringUTF(devname);
6. env->CallVoidMethod(thiz, method\_usbDeviceRemoved, deviceName);
7. env->DeleteLocalRef(deviceName);
8. checkAndClearExceptionFromCallback(env, \_\_FUNCTION\_\_);
9. **return** 0;
10. }

UsbHostManager的usbDeviceRemoved函数，会把mDevices相关的设备去除。

1. **private** **void** usbDeviceRemoved(String deviceName) {
2. synchronized (mLock) {
3. UsbDevice device = mDevices.remove(deviceName);
4. **if** (device != null) {
5. mUsbAlsaManager.usbDeviceRemoved(device);
6. getCurrentSettings().deviceDetached(device);
7. }
8. }
9. }

### UsbManager中调用usb设备

我们来看USBManager中关于获取Usb设备，以及使用它们的函数。getDeviceList可以得到一个设备名，以及对应的UsbDevice。而openDevice会得到一个UsbDeviceConnection对象。

1. **public** HashMap<String,UsbDevice> getDeviceList() {
2. Bundle bundle = **new** Bundle();
3. **try** {
4. mService.getDeviceList(bundle);
5. HashMap<String,UsbDevice> result = **new** HashMap<String,UsbDevice>();
6. **for** (String name : bundle.keySet()) {
7. result.put(name, (UsbDevice)bundle.get(name));
8. }
9. **return** result;
10. } **catch** (RemoteException e) {
11. Log.e(TAG, "RemoteException in getDeviceList", e);
12. **return** null;
13. }
14. }
16. /\*\*
17. \* Opens the device so it can be used to send and receive
18. \* data using {@link android.hardware.usb.UsbRequest}.
19. \*
20. \* @param device the device to open
21. \* @return a {@link UsbDeviceConnection}, or {@code null} if open failed
22. \*/
23. **public** UsbDeviceConnection openDevice(UsbDevice device) {
24. **try** {
25. String deviceName = device.getDeviceName();
26. ParcelFileDescriptor pfd = mService.openDevice(deviceName);
27. **if** (pfd != null) {
28. UsbDeviceConnection connection = **new** UsbDeviceConnection(device);
29. boolean result = connection.open(deviceName, pfd);
30. pfd.close();
31. **if** (result) {
32. **return** connection;
33. }
34. }
35. } **catch** (Exception e) {
36. Log.e(TAG, "exception in UsbManager.openDevice", e);
37. }
38. **return** null;
39. }

UsbHostManager的getDeviceList函数，就是把mDevices中的数据返回，然后是通过Bundle对象来传输的。

1. **public** **void** getDeviceList(Bundle devices) {
2. synchronized (mLock) {
3. **for** (String name : mDevices.keySet()) {
4. devices.putParcelable(name, mDevices.get(name));
5. }
6. }
7. }

而当用户进程通过UsbManger获取到设备名后，可以通过openDevice来获取ParcelFileDescriptor ，在UsbManager中又会新建一个UsbDeviceConnection，然后把ParcelFileDescriptor 作为参数调用这个UsbDeviceConnection的open函数。

1. **public** ParcelFileDescriptor openDevice(String deviceName) {
2. synchronized (mLock) {
3. **if** (isBlackListed(deviceName)) {
4. **throw** **new** SecurityException("USB device is on a restricted bus");
5. }
6. UsbDevice device = mDevices.get(deviceName);
7. **if** (device == null) {
8. // if it is not in mDevices, it either does not exist or is blacklisted
9. **throw** **new** IllegalArgumentException(
10. "device " + deviceName + " does not exist or is restricted");
11. }
12. getCurrentSettings().checkPermission(device);
13. **return** nativeOpenDevice(deviceName);
14. }
15. }

我们来看下UsbHostManager的openDevice函数，最后也是调用了JNI函数nativeOpenDevice

1. **public** ParcelFileDescriptor openDevice(String deviceName) {
2. synchronized (mLock) {
3. **if** (isBlackListed(deviceName)) {
4. **throw** **new** SecurityException("USB device is on a restricted bus");
5. }
6. UsbDevice device = mDevices.get(deviceName);
7. **if** (device == null) {
8. // if it is not in mDevices, it either does not exist or is blacklisted
9. **throw** **new** IllegalArgumentException(
10. "device " + deviceName + " does not exist or is restricted");
11. }
12. getCurrentSettings().checkPermission(device);
13. **return** nativeOpenDevice(deviceName);
14. }
15. }

android\_server\_UsbHostManager\_openDevice就是对应的JNI函数，这里主要调用了usb\_device\_open函数，并且返回一个usb\_device，最后我们通过usb\_device来获取其fd，并且把它封装在[Java](http://lib.csdn.net/base/java)层ParcelFileDescriptor类中。

1. **static** jobject android\_server\_UsbHostManager\_openDevice(JNIEnv \*env, jobject /\* thiz \*/,
2. jstring deviceName)
3. {
4. **const** **char** \*deviceNameStr = env->GetStringUTFChars(deviceName, NULL);
5. **struct** usb\_device\* device = usb\_device\_open(deviceNameStr);
6. env->ReleaseStringUTFChars(deviceName, deviceNameStr);
8. **if** (!device)
9. **return** NULL;
11. **int** fd = usb\_device\_get\_fd(device);
12. **if** (fd < 0) {
13. usb\_device\_close(device);
14. **return** NULL;
15. }
16. **int** newFD = dup(fd);
17. usb\_device\_close(device);
19. jobject fileDescriptor = jniCreateFileDescriptor(env, newFD);
20. **if** (fileDescriptor == NULL) {
21. **return** NULL;
22. }
23. **return** env->NewObject(gParcelFileDescriptorOffsets.mClass,
24. gParcelFileDescriptorOffsets.mConstructor, fileDescriptor);
25. }

usb\_device\_open函数就是打开对应的设备

1. **struct** usb\_device \*usb\_device\_open(**const** **char** \*dev\_name)
2. {
3. **int** fd, did\_retry = 0, writeable = 1;
5. D("usb\_device\_open %s\n", dev\_name);
7. retry:
8. fd = open(dev\_name, O\_RDWR);
9. **if** (fd < 0) {
10. /\* if we fail, see if have read-only access \*/
11. fd = open(dev\_name, O\_RDONLY);
12. D("usb\_device\_open open returned %d errno %d\n", fd, errno);
13. **if** (fd < 0 && (errno == EACCES || errno == ENOENT) && !did\_retry) {
14. /\* work around race condition between inotify and permissions management \*/
15. sleep(1);
16. did\_retry = 1;
17. **goto** retry;
18. }
20. **if** (fd < 0)
21. **return** NULL;
22. writeable = 0;
23. D("[ usb open read-only %s fd = %d]\n", dev\_name, fd);
24. }
26. **struct** usb\_device\* result = usb\_device\_new(dev\_name, fd);
27. **if** (result)
28. result->writeable = writeable;
29. **return** result;
30. }

然后我们再来看看UsbManager的openDevice使用的UsbDeviceConnection对象。

1. **public** UsbDeviceConnection openDevice(UsbDevice device) {
2. **try** {
3. String deviceName = device.getDeviceName();
4. ParcelFileDescriptor pfd = mService.openDevice(deviceName);
5. **if** (pfd != null) {
6. UsbDeviceConnection connection = **new** UsbDeviceConnection(device);
7. boolean result = connection.open(deviceName, pfd);
8. pfd.close();
9. **if** (result) {
10. **return** connection;
11. }
12. }
13. } **catch** (Exception e) {
14. Log.e(TAG, "exception in UsbManager.openDevice", e);
15. }
16. **return** null;
17. }

UsbDeviceConnection的构造函数就是把UsbDevice保存到mDevice，然后调用了native\_open函数

1. **public** UsbDeviceConnection(UsbDevice device) {
2. mDevice = device;
3. }
5. /\* package \*/ boolean open(String name, ParcelFileDescriptor pfd) {
6. **return** native\_open(name, pfd.getFileDescriptor());
7. }

native\_open函数是JNI函数，对应函数如下。根据之前打开设备的fd新建了一个usb\_device，然后保存在了UsbDeviceConnection的mNativeContext对象中了。

1. **static** jboolean
2. android\_hardware\_UsbDeviceConnection\_open(JNIEnv \*env, jobject thiz, jstring deviceName,
3. jobject fileDescriptor)
4. {
5. **int** fd = jniGetFDFromFileDescriptor(env, fileDescriptor);
6. // duplicate the file descriptor, since ParcelFileDescriptor will eventually close its copy
7. fd = dup(fd);
8. **if** (fd < 0)
9. **return** JNI\_FALSE;
11. **const** **char** \*deviceNameStr = env->GetStringUTFChars(deviceName, NULL);
12. **struct** usb\_device\* device = usb\_device\_new(deviceNameStr, fd);
13. **if** (device) {
14. env->SetLongField(thiz, field\_context, (jlong)device);
15. } **else** {
16. ALOGE("usb\_device\_open failed for %s", deviceNameStr);
17. close(fd);
18. }
20. env->ReleaseStringUTFChars(deviceName, deviceNameStr);
21. **return** (device != NULL) ? JNI\_TRUE : JNI\_FALSE;
22. }

最后用户进程获取的对象就是UsbDeviceConnection，我们可以调用其getFileDescriptor，来获取设备的fd。我们直接看下这个函数

1. **public** **int** getFileDescriptor() {
2. **return** native\_get\_fd();
3. }

而这个函数就是一个JNI函数，最后就是通过之前保存在UsbDeviceConnection的mNativeContext，把它变成一个usb\_device,然后再获取其fd。

1. **static** jint
2. android\_hardware\_UsbDeviceConnection\_get\_fd(JNIEnv \*env, jobject thiz)
3. {
4. **struct** usb\_device\* device = get\_device\_from\_object(env, thiz);
5. **if** (!device) {
6. ALOGE("device is closed in native\_get\_fd");
7. **return** -1;
8. }
9. **return** usb\_device\_get\_fd(device);
10. }

get\_device\_from\_object就是获取UsbDeviceConnection的mNativeContext对象（实际是之前保存的usb\_device对象）

1. **struct** usb\_device\* get\_device\_from\_object(JNIEnv\* env, jobject connection)
2. {
3. **return** (**struct** usb\_device\*)env->GetLongField(connection, field\_context);
4. }

这样通过UsbDeviceConnection的getFileDescriptor函数，用户进程拿到了自己需要的usb设备的fd了。

### UsbDeviceManager

接下来先讲UsbDeviceManager路径

先来看下构造函数：

1. **public** UsbDeviceManager(Context context) {
2. Log.d(TAG,"UsbDeviceManager start , persist.sys.usb.config =" + SystemProperties.get("persist.sys.usb.config", ""));
3. mContext = context;
4. mContentResolver = context.getContentResolver();
5. mMassStorageEnable = SystemProperties.get("persist.mass\_storage.enable", "true");//大容量属性
6. PackageManager pm = mContext.getPackageManager();
7. mHasUsbAccessory = pm.hasSystemFeature(PackageManager.FEATURE\_USB\_ACCESSORY);
8. initRndisAddress();//初始化usb共享的地址，用手机序列码作为地址，序列码是存在系统属性中的
10. readOemUsbOverrideConfig();//读取厂家设置
12. mHandler = **new** UsbHandler(FgThread.get().getLooper());//构造UsbHandler，后面详细看
14. **if** (nativeIsStartRequested()) {
15. **if** (DEBUG) Slog.d(TAG, "accessory attached at boot");
16. startAccessoryMode();
17. }
19. boolean secureAdbEnabled = SystemProperties.getBoolean("ro.adb.secure", **false**);
20. boolean dataEncrypted = "1".equals(SystemProperties.get("vold.decrypt"));
21. **if** (secureAdbEnabled && !dataEncrypted) {
22. mDebuggingManager = **new** UsbDebuggingManager(context);
23. }
24. }

接下来看UsbHandler这个内部类：

先看usbHandler构造函数的前半部分：

1. **public** UsbHandler(Looper looper) {
2. super(looper);
3. **try** {
4. // persist.sys.usb.config should never be unset.  But if it is, set it to "adb"
5. // so we have a chance of debugging what happened.
6. Log.d(TAG,"UsbHandler , persist.sys.usb.config :" + SystemProperties.get("persist.sys.usb.config"));
7. **if**(SystemProperties.get("persist.sys.usb.config", "") == null || SystemProperties.get("persist.sys.usb.config", "") == "none"){//永久性属性里面没有就设置
8. **if**("false".equals(mMassStorageEnable)){
9. SystemProperties.set("persist.sys.usb.config", "mtp,adb");
10. }**else**{
11. SystemProperties.set("persist.sys.usb.config", "mtp,mass\_storage,adb");
12. }
13. }
14. **if**("false".equals(mMassStorageEnable)){//有的话就直接将获取的属性放在mDefaultFunctions
15. mDefaultFunctions = SystemProperties.get("persist.sys.usb.config", "mtp,adb");
16. }**else**{
17. mDefaultFunctions = SystemProperties.get("persist.sys.usb.config", "mtp,mass\_storage,adb");
18. }
19. Log.d(TAG,"mMassStorageEnable ="+ mMassStorageEnable +", mDefaultFunctions :" + mDefaultFunctions);
20. // Check if USB mode needs to be overridden depending on OEM specific bootmode.
21. mDefaultFunctions = processOemUsbOverride(mDefaultFunctions);//厂家定制的
23. // sanity check the sys.usb.config system property
24. // this may be necessary if we crashed while switching USB configurations
25. String config = SystemProperties.get("sys.usb.config", "none");
26. **if** (!config.equals(mDefaultFunctions)) {//如果mDefaultFunctions与"sys.usb.config“属性不一样，就重新设置属性
27. Slog.w(TAG, "resetting config to persistent property: " + mDefaultFunctions);
28. SystemProperties.set("sys.usb.config", mDefaultFunctions);
29. }

上面理论上persist.sys.usb.config 与sys.usb.config属性一样的

因为在init.usb.rc中设置persist.sys.usb.config 属性，也会将sys.usb.config一同设置了。

1. # Used to set USB configuration at boot and to **switch** the configuration
2. # when changing the **default** configuration
3. on property:persist.sys.usb.config=\*
4. setprop sys.usb.config $persist.sys.usb.config

继续分析usbHandler的构造函数

1. mCurrentFunctions = getDefaultFunctions();
2. String state = FileUtils.readTextFile(**new** File(STATE\_PATH), 0, **null**).trim();
3. Log.d(TAG,"UsbHandler updateState  state:" + state);
4. mAdbEnabled = containsFunction(mCurrentFunctions, UsbManager.USB\_FUNCTION\_ADB);//属性中如果有adb，就置位true
5. updateState(state);
6. Log.d(TAG," UsbHandler charging\_value:" + SystemProperties.get("sys.usb.charging", "")
7. + "  ro.debuggable:" + SystemProperties.get("ro.debuggable", ""));
8. **if**(SystemProperties.get("sys.usb.charging", "").equals("yes") && SystemProperties.get("ro.debuggable","").equals("1")){

下面再看看updateState函数，state从下面这个节点读取状态

1. **private** **static** **final** String STATE\_PATH =
2. "/sys/class/android\_usb/android0/state"

updateState函数如下：

1. **public** **void** updateState(String state) {
2. Log.d(TAG, "updateState : state = " + state );
3. **int** connected, configured;
5. **if** ("DISCONNECTED".equals(state)) {
6. connected = 0;
7. configured = 0;
8. SystemProperties.set("sys.lc.usb.state", "0");
9. } **else** **if** ("CONNECTED".equals(state)) {
10. connected = 1;
11. configured = 0;
12. } **else** **if** ("CONFIGURED".equals(state)) {
13. connected = 1;
14. configured = 1;
15. SystemProperties.set("sys.lc.usb.state", "1");
16. } **else** {
17. Slog.e(TAG, "unknown state " + state);
18. **return**;
19. }
20. removeMessages(MSG\_UPDATE\_STATE);
21. Message msg = Message.obtain(**this**, MSG\_UPDATE\_STATE);
22. msg.arg1 = connected;
23. msg.arg2 = configured;
24. // debounce disconnects to avoid problems bringing up USB tethering
25. sendMessage(msg);//参数配置后发送消息

收到消息后处理

1. **public** **void** handleMessage(Message msg) {
2. Log.d(TAG," handleMessage msg:" + msg + " msg.arg1:" + msg.arg1 + "  msg.arg2  "+ msg.arg2);
3. **switch** (msg.what) {
4. **case** MSG\_UPDATE\_STATE:
5. mConnected = (msg.arg1 == 1);
6. mConfigured = (msg.arg2 == 1);
7. updateUsbNotification();
8. updateAdbNotification();
9. **if** (containsFunction(mCurrentFunctions,
10. UsbManager.USB\_FUNCTION\_ACCESSORY)) {
11. updateCurrentAccessory();
12. } **else** **if** (!mConnected) {//usb断开连接后，属性设置成默认的
13. // restore defaults when USB is disconnected
14. setEnabledFunctions(getDefaultFunctions(), **false**);
15. }
16. **if** (mBootCompleted) {
17. updateUsbState();//发送广播
18. updateAudioSourceFunction();
19. }
20. **break**;

继续分析

1. String value = SystemProperties.get("persist.service.adb.enable", "");
2. **if** (value.length() > 0) {
3. **char** enable = value.charAt(0);
4. **if** (enable == '1') {
5. setAdbEnabled(**true**);
6. } **else** **if** (enable == '0') {
7. setAdbEnabled(**false**);
8. }
9. SystemProperties.set("persist.service.adb.enable", "");
10. }

分析setAdbEnabled函数

1. **private** **void** setAdbEnabled(**boolean** enable) {
2. **if** (DEBUG) Slog.d(TAG, "setAdbEnabled: " + enable + " mCurrentFunctions:" + mCurrentFunctions);
3. **if** (enable != mAdbEnabled) {//当mAdbEnabled改变时
4. mAdbEnabled = enable;
5. // Due to the persist.sys.usb.config property trigger, changing adb state requires
6. // persisting default function
7. **if**(!mCurrentFunctions.equals(UsbManager.USB\_FUNCTION\_CHARGING)){
8. setEnabledFunctions(mDefaultFunctions, **true**);//先设置永久属性
9. // After persisting them use the lock-down aware function set
10. setEnabledFunctions(getDefaultFunctions(), **false**);//再设置usb.config属性
11. updateAdbNotification();
12. }
13. }
14. **if** (mDebuggingManager != **null**) {
15. mDebuggingManager.setAdbEnabled(mAdbEnabled);
16. }
17. }

再来看看setEnabledFunctions函数

1. **private** **void** setEnabledFunctions(String functions, **boolean** makeDefault) {
2. **if** (DEBUG) Slog.d(TAG, "setEnabledFunctions " + functions
3. + " makeDefault: " + makeDefault);
4. // Do not update persystent.sys.usb.config if the device is booted up
5. // with OEM specific mode.
6. **if** (functions != **null** && makeDefault && !needsOemUsbOverride()) {//设置永久属性
7. **if** (mAdbEnabled && !functions.equals(UsbManager.USB\_FUNCTION\_CHARGING)) {
8. functions = addFunction(functions, UsbManager.USB\_FUNCTION\_ADB);//根据mAdbEnabled设置系统属性
9. } **else** {
10. functions = removeFunction(functions, UsbManager.USB\_FUNCTION\_ADB);
11. }
12. **if** (!mDefaultFunctions.equals(functions)) {//当functions改变时
13. **if** (!setUsbConfig("none")) {//设置none，直接断开usb，也就是差不多每次设置不一样属性。都要先断开usb。稍后详细看看setUsbConfig这个函数
14. Slog.e(TAG, "Failed to disable USB");
15. // revert to previous configuration if we fail
16. setUsbConfig(mCurrentFunctions);//
17. **return**;
18. }
19. // setting this property will also change the current USB state
20. // via a property trigger
21. SystemProperties.set("persist.sys.usb.config", functions);// 设置永久属性
22. **if** (waitForState(functions)) {//就是设置的属性，去读取属性是否和设置的一样
23. **if**(mCurrentFunctions.equals(UsbManager.USB\_FUNCTION\_CHARGING) && !mCurrentFunctions.equals(functions)){
24. **if** (DEBUG) Slog.d(TAG, " change charging functions to new functions: " + functions);
25. SystemProperties.set("sys.usb.charging", "no");
26. }
27. mCurrentFunctions = functions;
28. mDefaultFunctions = functions;
29. } **else** {//不成功，设置层默认的
30. Slog.e(TAG, "Failed to switch persistent USB config to " + functions);
31. // revert to previous configuration if we fail
32. SystemProperties.set("persist.sys.usb.config", mDefaultFunctions);
33. }
34. }
35. }

非永久属性的，和永久属性的差不多。只是一个是设置永久属性。一个是设置usb.config这个属性

1. **else** {
2. **if** (functions == **null**) {
3. functions = mDefaultFunctions;
4. }
6. // Override with bootmode specific usb mode if needed
7. functions = processOemUsbOverride(functions);
9. **if** (mAdbEnabled && !functions.equals(UsbManager.USB\_FUNCTION\_CHARGING)) {
10. functions = addFunction(functions, UsbManager.USB\_FUNCTION\_ADB);
11. } **else** {
12. functions = removeFunction(functions, UsbManager.USB\_FUNCTION\_ADB);
13. }
14. **if** (!mCurrentFunctions.equals(functions)) {
15. **if** (!setUsbConfig("none")) {//断开usb
16. Slog.e(TAG, "Failed to disable USB");
17. // revert to previous configuration if we fail
18. setUsbConfig(mCurrentFunctions);
19. **return**;
20. }
21. **if** (setUsbConfig(functions)) {
22. **if**(mCurrentFunctions.equals(UsbManager.USB\_FUNCTION\_CHARGING) && !mCurrentFunctions.equals(functions)){
23. **if** (DEBUG) Slog.d(TAG, " change charging functions to new functions: " + functions);
24. SystemProperties.set("sys.usb.charging", "no");
25. }
26. mCurrentFunctions = functions;
27. } **else** {
28. Slog.e(TAG, "Failed to switch USB config to " + functions);
29. // revert to previous configuration if we fail
30. setUsbConfig(mCurrentFunctions);
31. }
32. }
33. }

下面我们再来看看setUsbConfig这个函数

1. **private** **boolean** setUsbConfig(String config) {
2. **if** (DEBUG) Slog.d(TAG, "setUsbConfig(" + config + ")");
3. // set the new configuration
4. SystemProperties.set("sys.usb.config", config);
5. **return** waitForState(config);
6. }

waitForState函数

1. **private** **boolean** waitForState(String state) {
2. // wait for the transition to complete.
3. // give up after 1 second.
4. **for** (**int** i = 0; i < 40; i++) {
5. // State transition is done when sys.usb.state is set to the new configuration
6. **if** (state.equals(SystemProperties.get("sys.usb.state"))){//设置的属性和读取的属性相同
7. Log.i(TAG,"Setting SystemProperties ,i =  "+i);
8. **if**(state.equals("none")){
9. **if**(SystemProperties.get("sys.lc.usb.state").equals("0")){
10. **return** **true**;
11. }**else**{
12. Log.i(TAG,"sys.lc.usb.state ="+SystemProperties.get("sys.lc.usb.state"));
13. }
14. }**else**{
15. **return** **true**;
16. }
17. }
18. SystemClock.sleep(50);//不相同，睡50ms 也就是最多会睡2s
19. }
20. Slog.e(TAG, "waitForState(" + state + ") FAILED");
21. **return** **false**;
22. }

继续分析usbhandler

1. mContentResolver.registerContentObserver(
2. Settings.Global.getUriFor(Settings.Global.ADB\_ENABLED),
3. **false**, **new** AdbSettingsObserver());//注册一个adb的观察者，会发送消息。
5. // Watch for USB configuration changes
6. mUEventObserver.startObserving(USB\_STATE\_MATCH);//开启两个节点的监控。
7. mUEventObserver.startObserving(ACCESSORY\_START\_MATCH);
9. IntentFilter filter = **new** IntentFilter(Intent.ACTION\_BOOT\_COMPLETED);
10. filter.setPriority(IntentFilter.SYSTEM\_HIGH\_PRIORITY);
11. mContext.registerReceiver(mBootCompletedReceiver, filter);
12. mContext.registerReceiver(
13. mUserSwitchedReceiver, **new** IntentFilter(Intent.ACTION\_USER\_SWITCHED));
14. mContext.registerReceiver(
15. mBatteryChangedReceiver, **new** IntentFilter(Intent.ACTION\_BATTERY\_CHANGED));

AdbSettingsObserver 发送消息

1. **private** **class** AdbSettingsObserver **extends** ContentObserver {
2. **public** AdbSettingsObserver() {
3. **super**(**null**);
4. }
5. @Override
6. **public** **void** onChange(**boolean** selfChange) {
7. **boolean** enable = (Settings.Global.getInt(mContentResolver,
8. Settings.Global.ADB\_ENABLED, 0) > 0);
9. mHandler.sendMessage(MSG\_ENABLE\_ADB, enable);
10. }
11. }

消息处理，调用setAdbEnabled

1. **case** MSG\_ENABLE\_ADB:
2. setAdbEnabled(msg.arg1 == 1);
3. **break**;

UEventObserver 分析

1. **private** **final** UEventObserver mUEventObserver = **new** UEventObserver() {
2. @Override
3. **public** **void** onUEvent(UEventObserver.UEvent event) {
4. **if** (DEBUG) Slog.v(TAG, "USB UEVENT: " + event.toString());
6. String state = event.get("USB\_STATE");
7. String accessory = event.get("ACCESSORY");
8. **if** (state != **null**) {//state对应一个节点
9. mHandler.updateState(state);//updateState函数前面分析过了
10. } **else** **if** ("START".equals(accessory)) {//accessory对应一个节点
11. **if** (DEBUG) Slog.d(TAG, "got accessory start");
12. startAccessoryMode();
13. }
14. }
15. }

startAccessoryMode函数

1. **private** **void** startAccessoryMode() {
2. **if** (!mHasUsbAccessory) **return**;
4. mAccessoryStrings = nativeGetAccessoryStrings();
5. **boolean** enableAudio = (nativeGetAudioMode() == AUDIO\_MODE\_SOURCE);
6. // don't start accessory mode if our mandatory strings have not been set
7. **boolean** enableAccessory = (mAccessoryStrings != **null** &&
8. mAccessoryStrings[UsbAccessory.MANUFACTURER\_STRING] != **null** &&
9. mAccessoryStrings[UsbAccessory.MODEL\_STRING] != **null**);
10. String functions = **null**;
12. **if** (enableAccessory && enableAudio) {
13. functions = UsbManager.USB\_FUNCTION\_ACCESSORY + ","
14. + UsbManager.USB\_FUNCTION\_AUDIO\_SOURCE;
15. } **else** **if** (enableAccessory) {
16. functions = UsbManager.USB\_FUNCTION\_ACCESSORY;
17. } **else** **if** (enableAudio) {
18. functions = UsbManager.USB\_FUNCTION\_AUDIO\_SOURCE;
19. }
21. **if** (functions != **null**) {
22. mAccessoryModeRequestTime = SystemClock.elapsedRealtime();
23. setCurrentFunctions(functions, **false**);//设置属性
24. }
25. }

还有就是用户调用usbmanger中的setCurrentFunction

1. **public** **void** setCurrentFunction(String function, **boolean** makeDefault) {
2. **try** {
3. mService.setCurrentFunction(function, makeDefault);
4. } **catch** (RemoteException e) {
5. Log.e(TAG, "RemoteException in setCurrentFunction", e);
6. }
7. }

接着调用usbservice中的setCurrentFunction函数

1. **public** **void** setCurrentFunction(String function, **boolean** makeDefault) {
2. mContext.enforceCallingOrSelfPermission(android.Manifest.permission.MANAGE\_USB, **null**);
4. // If attempt to change USB function while file transfer is restricted, ensure that
5. // the current function is set to "none", and return.
6. UserManager userManager = (UserManager) mContext.getSystemService(Context.USER\_SERVICE);
7. **if** (userManager.hasUserRestriction(UserManager.DISALLOW\_USB\_FILE\_TRANSFER)) {
8. **if** (mDeviceManager != **null**) mDeviceManager.setCurrentFunctions("none", **false**);
9. **return**;
10. }
12. **if** (mDeviceManager != **null**) {
13. mDeviceManager.setCurrentFunctions(function, makeDefault);
14. } **else** {
15. **throw** **new** IllegalStateException("USB device mode not supported");
16. }
17. }

最后调用usbdevicemanager中的setCurrentFunctions函数

1. **public** **void** setCurrentFunctions(String functions, **boolean** makeDefault) {
2. String currentUsbConfig = SystemProperties.get("sys.usb.config", "none");// 先从属性里面读出来，主要看有没有adb
3. Log.i(TAG, "begin to setCurrentFunction : sys.usb.config = " + currentUsbConfig + " , mMassStorageEnable:"
4. + mMassStorageEnable + " , functions =" + functions +" , makeDefault =" + makeDefault);
5. **if**(functions != **null**){
6. **if**(containsFunction(functions,UsbManager.USB\_FUNCTION\_ENABLE\_SERIAL)){
7. functions = "serial";
8. }**else** **if**(containsFunction(functions,UsbManager.USB\_FUNCTION\_DISABLE\_SERIAL)){
9. **if**("false".equals(mMassStorageEnable)){
10. functions = "mtp";
11. }**else**{
12. functions = "mtp,mass\_storage";
13. }
14. }**else** **if**(containsFunction(functions,UsbManager.USB\_FUNCTION\_RNDIS)){// usb共享要看，串口有没有开，
15. **if**(containsFunction(currentUsbConfig,UsbManager.USB\_FUNCTION\_SERIAL)){
16. functions = "rndis,serial";//开了串口才能把这两项都设置到属性中去
17. }
18. }
19. }
20. **if**(containsFunction(currentUsbConfig,UsbManager.USB\_FUNCTION\_ADB)){
21. **if**(functions != **null**)addFunction(functions,UsbManager.USB\_FUNCTION\_ADB);
22. }
23. Log.i(TAG, "setCurrentFunction  finally  function:"  + fixFunction(functions) + " makeDefault:" + makeDefault);
24. mHandler.sendMessage(MSG\_SET\_CURRENT\_FUNCTIONS, fixFunction(functions), makeDefault);
25. }

fixFunction的话是将一些其他的属性加入，因为设置属性每次都要设置所有的

1. **private** String fixFunction(String function) {
2. Log.d(TAG," fixFunction: function:" + function);
3. String ret = function;
4. **if** (**null** == function) {
5. **return** **null**;
6. }
8. **if** (containsFunction(function, UsbManager.USB\_FUNCTION\_MTP)){
9. **if**("false".equals(mMassStorageEnable)){
10. ret = "mtp";
11. } **else** {
12. ret = "mtp,mass\_storage";
13. }
14. } **else** **if** (containsFunction(function, UsbManager.USB\_FUNCTION\_PTP)) {
15. **if**("false".equals(mMassStorageEnable)){
16. ret = "ptp";
17. } **else** {
18. ret = "ptp,mass\_storage";
19. }
20. } **else** {
21. **return** ret;
22. }
24. **if** (containsFunction(function, UsbManager.USB\_FUNCTION\_ADB)) {
25. addFunction(ret, UsbManager.USB\_FUNCTION\_ADB);
26. }
28. **return** ret;
29. }

发送消息之后，消息处理。主要还是调用setEnabledFunctions设置属性。当属性中含有rndis和accessory经常导致不能及时获取的属性值一样。而waitforstate时会等待，所有这边就起了一个线程。

1. **case** MSG\_SET\_CURRENT\_FUNCTIONS:
2. **final** String functions = (String)msg.obj;
3. **final** **boolean** makeDefault = (msg.arg1 == 1);
4. Log.d(TAG, "set current funtion to " + functions);
5. **if** (**null** != functions){
6. **if**(!functions.contains(UsbManager.USB\_FUNCTION\_CHARGING)){
7. mConnected\_InChargeOnlyMode = **false**;
8. }**else**{
9. mConnected\_InChargeOnlyMode = **true**;
10. }
11. }
12. **if** (**null** != functions && (functions.contains("rndis")||functions.contains("accessory"))) {
13. **new** Thread() {
14. **public** **void** run() {
15. Log.d(TAG, "new Thread setEnabledFunctions to " + functions);
16. setEnabledFunctions(functions, makeDefault);
17. }
18. }.start();
19. } **else** {
20. setEnabledFunctions(functions, makeDefault);
21. }
22. **break**;

下面大致说下各个属性。

1. **public** **static** **final** String USB\_FUNCTION\_ADB = "adb";
3. /\*\*
4. \* Name of the RNDIS ethernet USB function.
5. \* Used in extras for the {@link #ACTION\_USB\_STATE} broadcast
6. \*
7. \* {@hide}
8. \*/
9. **public** **static** **final** String USB\_FUNCTION\_RNDIS = "rndis";//usb共享
11. /\*\*
12. \* Name of the MTP USB function.
13. \* Used in extras for the {@link #ACTION\_USB\_STATE} broadcast
14. \*
15. \* {@hide}
16. \*/
17. **public** **static** **final** String USB\_FUNCTION\_MTP = "mtp";
19. /\*\*
20. \* Name of the PTP USB function.
21. \* Used in extras for the {@link #ACTION\_USB\_STATE} broadcast
22. \*
23. \* {@hide}
24. \*/
25. **public** **static** **final** String USB\_FUNCTION\_PTP = "ptp";
27. /\*\*
28. \* Name of the audio source USB function.
29. \* Used in extras for the {@link #ACTION\_USB\_STATE} broadcast
30. \*
31. \* {@hide}
32. \*/
33. **public** **static** **final** String USB\_FUNCTION\_AUDIO\_SOURCE = "audio\_source";
35. /\*\*
36. \* Name of the Accessory USB function.
37. \* Used in extras for the {@link #ACTION\_USB\_STATE} broadcast
38. \*
39. \* {@hide}
40. \*/
41. **public** **static** **final** String USB\_FUNCTION\_ACCESSORY = "accessory";
43. /\*\*
44. \* Name of the charging USB function.
45. \* Used in extras for the {@link #ACTION\_USB\_STATE} broadcast
46. \*
47. \* {@hide}
48. \*/
49. **public** **static** **final** String USB\_FUNCTION\_CHARGING = "charging";
51. /\*\*
52. \* Name of the serial USB function.
53. \* Used in extras for the {@link #ACTION\_USB\_STATE} broadcast
54. \*
55. \* {@hide}
56. \*/
57. **public** **static** **final** String USB\_FUNCTION\_SERIAL = "serial";//开启串口的

还有整个usb.config属性设置后。会触发init.usb.rc中的一些触发器。这在init的博客中分析过了。

下面我们看下整个init.usb.rc，是各个排列组合都列出来了。最后一个是设置persist.sys.usb.config永久系统属性的时候也会设置 sys.usb.config属性。

1. # Used to disable USB when switching states
2. on property:sys.usb.config=none//设置属性none
3. stop adbd
4. write /sys/**class**/android\_usb/android0/enable 0//写enable为0
5. write /sys/**class**/android\_usb/android0/bDeviceClass 0
6. setprop sys.usb.state $sys.usb.config
8. #mtp,mass\_storage,adb
9. on property:sys.usb.config=mtp,mass\_storage,adb
10. write /sys/**class**/android\_usb/android0/enable 0
11. write /sys/**class**/android\_usb/android0/idVendor 18D1
12. write /sys/**class**/android\_usb/android0/idProduct 181C
13. write /sys/**class**/android\_usb/android0/functions $sys.usb.config
14. write /sys/**class**/android\_usb/android0/enable 1
15. start adbd
16. setprop sys.usb.state $sys.usb.config
18. #mtp,mass\_storage
19. on property:sys.usb.config=mtp,mass\_storage
20. write /sys/**class**/android\_usb/android0/enable 0
21. write /sys/**class**/android\_usb/android0/idVendor 18D1
22. write /sys/**class**/android\_usb/android0/idProduct 181C
23. write /sys/**class**/android\_usb/android0/functions $sys.usb.config
24. write /sys/**class**/android\_usb/android0/enable 1
25. setprop sys.usb.state $sys.usb.config
27. #ptp,mass\_storage,adb
28. on property:sys.usb.config=ptp,mass\_storage,adb
29. write /sys/**class**/android\_usb/android0/enable 0
30. write /sys/**class**/android\_usb/android0/idVendor 18D1
31. write /sys/**class**/android\_usb/android0/idProduct 181D
32. write /sys/**class**/android\_usb/android0/functions $sys.usb.config
33. write /sys/**class**/android\_usb/android0/enable 1
34. start adbd
35. setprop sys.usb.state $sys.usb.config
37. #ptp,mass\_storage
38. on property:sys.usb.config=ptp,mass\_storage
39. write /sys/**class**/android\_usb/android0/enable 0
40. write /sys/**class**/android\_usb/android0/idVendor 18D1
41. write /sys/**class**/android\_usb/android0/idProduct 181D
42. write /sys/**class**/android\_usb/android0/functions $sys.usb.config
43. write /sys/**class**/android\_usb/android0/enable 1
44. setprop sys.usb.state $sys.usb.config
46. #rndis,adb,serial
47. on property:sys.usb.config=rndis,adb,serial
48. write /sys/**class**/android\_usb/android0/enable 0
49. write /sys/**class**/android\_usb/android0/idVendor 18D1
50. write /sys/**class**/android\_usb/android0/idProduct 181F
51. write /sys/**class**/android\_usb/android0/functions rndis,serial,adb
52. write /sys/**class**/android\_usb/android0/f\_rndis/wceis 1
53. write /sys/**class**/android\_usb/android0/f\_serial/serial\_port\_num 4
54. write /sys/**class**/android\_usb/android0/enable 1
55. start adbd
56. setprop sys.usb.state rndis,serial,adb
58. #rndis,serial,adb
59. on property:sys.usb.config=rndis,serial,adb
60. write /sys/**class**/android\_usb/android0/enable 0
61. write /sys/**class**/android\_usb/android0/idVendor 18D1
62. write /sys/**class**/android\_usb/android0/idProduct 181F
63. write /sys/**class**/android\_usb/android0/functions $sys.usb.config
64. write /sys/**class**/android\_usb/android0/f\_rndis/wceis 1
65. write /sys/**class**/android\_usb/android0/f\_serial/serial\_port\_num 4
66. write /sys/**class**/android\_usb/android0/enable 1
67. start adbd
68. setprop sys.usb.state $sys.usb.config
70. #rndis,serial
71. on property:sys.usb.config=rndis,serial
72. write /sys/**class**/android\_usb/android0/enable 0
73. write /sys/**class**/android\_usb/android0/idVendor 18D1
74. write /sys/**class**/android\_usb/android0/idProduct 181F
75. write /sys/**class**/android\_usb/android0/functions $sys.usb.config
76. write /sys/**class**/android\_usb/android0/f\_rndis/wceis 1
77. write /sys/**class**/android\_usb/android0/f\_serial/serial\_port\_num 4
78. write /sys/**class**/android\_usb/android0/enable 1
79. setprop sys.usb.state $sys.usb.config
81. #rndis,adb
82. on property:sys.usb.config=rndis,adb
83. write /sys/**class**/android\_usb/android0/enable 0
84. write /sys/**class**/android\_usb/android0/idVendor 18D1
85. write /sys/**class**/android\_usb/android0/idProduct 181E
86. write /sys/**class**/android\_usb/android0/functions $sys.usb.config
87. write /sys/**class**/android\_usb/android0/f\_rndis/wceis 1
88. write /sys/**class**/android\_usb/android0/enable 1
89. start adbd
90. setprop sys.usb.state $sys.usb.config
92. #rndis
93. on property:sys.usb.config=rndis
94. write /sys/**class**/android\_usb/android0/enable 0
95. write /sys/**class**/android\_usb/android0/idVendor 18D1
96. write /sys/**class**/android\_usb/android0/idProduct 181E
97. write /sys/**class**/android\_usb/android0/functions $sys.usb.config
98. write /sys/**class**/android\_usb/android0/f\_rndis/wceis 1
99. write /sys/**class**/android\_usb/android0/enable 1
100. setprop sys.usb.state $sys.usb.config
102. #mass\_storage,serial,adb
103. on property:sys.usb.config=mass\_storage,serial,adb
104. write /sys/**class**/android\_usb/android0/enable 0
105. write /sys/**class**/android\_usb/android0/idVendor 18D1
106. write /sys/**class**/android\_usb/android0/idProduct 181B
107. write /sys/**class**/android\_usb/android0/functions $sys.usb.config
108. write /sys/**class**/android\_usb/android0/f\_serial/serial\_port\_num 5
109. write /sys/**class**/android\_usb/android0/enable 1
110. start adbd
111. setprop sys.usb.state $sys.usb.config

114. #accessory,adb
115. on property:sys.usb.config=accessory,adb
116. write /sys/**class**/android\_usb/android0/enable 0
117. write /sys/**class**/android\_usb/android0/idVendor 18D1
118. write /sys/**class**/android\_usb/android0/idProduct 2D00
119. write /sys/**class**/android\_usb/android0/functions $sys.usb.config
120. write /sys/**class**/android\_usb/android0/enable 1
121. start adbd
122. setprop sys.usb.state $sys.usb.config
124. #accessory
125. on property:sys.usb.config=accessory
126. write /sys/**class**/android\_usb/android0/enable 0
127. write /sys/**class**/android\_usb/android0/idVendor 18D1
128. write /sys/**class**/android\_usb/android0/idProduct 2D01
129. write /sys/**class**/android\_usb/android0/functions $sys.usb.config
130. write /sys/**class**/android\_usb/android0/enable 1
131. setprop sys.usb.state $sys.usb.config
133. #adb
134. on property:sys.usb.config=adb
135. write /sys/**class**/android\_usb/android0/enable 0
136. write /sys/**class**/android\_usb/android0/idVendor 18D1
137. write /sys/**class**/android\_usb/android0/idProduct 1819
138. write /sys/**class**/android\_usb/android0/functions $sys.usb.config
139. write /sys/**class**/android\_usb/android0/enable 1
140. start adbd
141. setprop sys.usb.state $sys.usb.config
143. #mtp,adb
144. on property:sys.usb.config=mtp,adb
145. write /sys/**class**/android\_usb/android0/enable 0
146. write /sys/**class**/android\_usb/android0/idVendor 18D1
147. write /sys/**class**/android\_usb/android0/idProduct 1818
148. write /sys/**class**/android\_usb/android0/functions $sys.usb.config
149. write /sys/**class**/android\_usb/android0/enable 1
150. start adbd
151. setprop sys.usb.state $sys.usb.config
153. #mtp
154. on property:sys.usb.config=mtp
155. write /sys/**class**/android\_usb/android0/enable 0
156. write /sys/**class**/android\_usb/android0/idVendor 18D1
157. write /sys/**class**/android\_usb/android0/idProduct 1818
158. write /sys/**class**/android\_usb/android0/functions $sys.usb.config
159. write /sys/**class**/android\_usb/android0/enable 1
160. start adbd
161. setprop sys.usb.state $sys.usb.config
163. #ptp,adb
164. on property:sys.usb.config=ptp,adb
165. write /sys/**class**/android\_usb/android0/enable 0
166. write /sys/**class**/android\_usb/android0/idVendor 18D1
167. write /sys/**class**/android\_usb/android0/idProduct 1817
168. write /sys/**class**/android\_usb/android0/functions $sys.usb.config
169. write /sys/**class**/android\_usb/android0/enable 1
170. start adbd
171. setprop sys.usb.state $sys.usb.config
173. #ptp
174. on property:sys.usb.config=ptp
175. write /sys/**class**/android\_usb/android0/enable 0
176. write /sys/**class**/android\_usb/android0/idVendor 18D1
177. write /sys/**class**/android\_usb/android0/idProduct 1817
178. write /sys/**class**/android\_usb/android0/functions $sys.usb.config
179. write /sys/**class**/android\_usb/android0/enable 1
180. setprop sys.usb.state $sys.usb.config
182. #serial,adb
183. on property:sys.usb.config=serial,adb
184. write /sys/**class**/android\_usb/android0/enable 0
185. write /sys/**class**/android\_usb/android0/idVendor 18D1
186. write /sys/**class**/android\_usb/android0/idProduct 1816
187. write /sys/**class**/android\_usb/android0/functions $sys.usb.config
188. write /sys/**class**/android\_usb/android0/f\_serial/serial\_port\_num 4
189. write /sys/**class**/android\_usb/android0/enable 1
190. start adbd
191. setprop sys.usb.state $sys.usb.config

194. #serial
195. on property:sys.usb.config=serial
196. write /sys/**class**/android\_usb/android0/enable 0
197. write /sys/**class**/android\_usb/android0/idVendor 18D1
198. write /sys/**class**/android\_usb/android0/idProduct 1816
199. write /sys/**class**/android\_usb/android0/functions $sys.usb.config
200. write /sys/**class**/android\_usb/android0/f\_serial/serial\_port\_num 4
201. write /sys/**class**/android\_usb/android0/enable 1
202. setprop sys.usb.state $sys.usb.config

205. #USB Charging Only configuration
206. #**for** android\_usb device should not been none,so set a **default** function in **this** mode
207. on property:sys.usb.config=charging
208. write /sys/**class**/android\_usb/android0/enable 0
209. write /sys/**class**/android\_usb/android0/idVendor 18D1
210. write /sys/**class**/android\_usb/android0/idProduct 181C
211. write /sys/**class**/android\_usb/android0/functions mtp,mass\_storage,adb
212. write /sys/**class**/android\_usb/android0/enable 1
213. setprop sys.usb.state $sys.usb.config
214. setprop sys.usb.charging yes
216. on property:sys.usb.charging=yes
217. write /sys/devices/platform/comip-u2d/charge\_mode 1
219. on property:sys.usb.charging=no
220. write /sys/devices/platform/comip-u2d/charge\_mode 0
222. # Used to set USB configuration at boot and to **switch** the configuration
223. # when changing the **default** configuration
224. on property:persist.sys.usb.config=\*
225. setprop sys.usb.config $persist.sys.usb.config

usbdevicemanager差不多分析完了。usb这块暂时先这样后续还会继续深入分析，比如UsbHostManager和UsbSettingsManager那块。

## 参考资料

<https://www.ibm.com/developerworks/cn/linux/l-usb/index1.html>

<https://www.ibm.com/developerworks/cn/linux/l-usb/index2.html>

<http://blog.csdn.net/aaronychen/article/details/2192147>

<http://blog.csdn.net/fanqipin/article/details/8480972>

<http://blog.csdn.net/u013686019/article/details/50409421#comments>

<https://simowce.github.io/study/2016/03/20/android-usb-framework-usbhostmanager/>

<http://www.jianshu.com/p/51809085e9cc>

<http://blog.csdn.net/kc58236582/article/details/54691334>

<http://blog.csdn.net/kc58236582/article/details/47810987>

<http://blog.csdn.net/u011279649/article/details/17419473>