# Todo

发送消息弹窗，我们直接弹窗

Device/google/accessory/Arduino/AndroidAccessory

<https://www.jianshu.com/p/3dc561555f12>

# USB

USB 是 Universal Serial Bus 的缩写，从字面上看，就是通用串行总线的意思。从物理上看，其实就是一对差分线，连接两台设备后，相互间进行数据传输。加上另外两路供电（ 5V 和 GND）线，一共是 4 根线。

那么，既然是只有一对差分线，那么该如何决定由谁传给谁呢（如果两边同时在线上建立电平，线路上的电平会是不确定态的，以致无法通信）？这就要说到 USB 传输的一个重要基础：“询问-应答” 机制—— Device（slave） 设备通常是处在等待状态，只有 HOST 侧设备发起询问、请求，它才会在接下来的时间片中使用数据线向 HOST 发送数据。

那么，谁是 HOST，谁是 SLAVE 又是由什么来决定的呢？答案是硬件。也就是说，你 USB 后面的那块驱动芯片如果是 HOST，那么，这个 USB 只能做 HOST 用了。反之，SLAVE 亦然。比如我们经常见到的，PC 上的 USB HOST 连接到 U盘、鼠标、键盘这些 SLAVE 设备。

后来有人觉得这样一个设备只能是 HOST 或者只能是 SLAVE 太死板了，所以又发明了 USB OTG。USB OTG（on-the-go，大意为在使用时切换身份）是在原来 4 根线的基础上，又加了一根线，ID。那块 USB 后面的驱动芯片，就可以根据这根线，来选择自己到底该扮演 HOST 还是 SLAVE 的角色。后面我们单独介绍。

另外，因为使用一对差分线进行数据传输，所以，USB 又采用了基于 HUB 的星形拓扑结构（包括根控制器，最多7 层拓扑，且7层已不具备挂载 HUB 能力，只能是功能设备）。所以，更确切来说，“HOST-SLAVE“ 是在由 HUB 支持的物理链路之上的传输机制。同时，HUB 本身也是一个 USB SLAVE 设备。

root@gl300k:/ # getprop | grep usb

[persist.sys.usb.config]: [mass\_storage,adb,acm]

[prop.usb.connected]: [1]

[prop.usb.state]: [1]

[ro.kom\_wifi.mode]: [usb]

[ro.product.usbfactory]: [rockchip\_usb]

[sys.status.hideStatusbar\_enable]: [true]

[sys.usb.config]: [mass\_storage,adb,acm]

[sys.usb.state]: [mass\_storage,adb,acm]

[sys.usb.umsavailible]: [true]

init.xxx.usb.rc这里定义了所有usb device协议的组合。当usb device的协议发生变化的时候，会设置sys.usb.config这个属性，init.xxx.usb.rc中定义的某种组合会被触发，通过sys节点来通知kernel切换USB总线协议。

USB Device的功能很丰富，其支持的协议越来越多包括：MTP、ADB、rndis、mass storage、accessory、audio\_source、CDROOM等。

## 功能概述

### [协议切换](https://www.jianshu.com/p/b267c5cedfa9)

常用到的有device协议有ADB、MTP、PTP、MassStorage这几个，这些都是可以在Setting中开关或者是切换的。

在切换协议的时候是调用UsbDeviceManager中的setCurrentFunctions(String functions, boolean makeDefault)最终设置sys.usb.config这个属性，从而触发init.xx.usb.rc去通知kernel切换usb协议。

UsbDeviceManager.java中同时也监听usb事件的uevent，并通过updateUsbState（）发出UsbManager.ACTION\_USB\_STATE这个广播来通知MtpReceiver和MountService。其中MtpReceiver负责根据所选择的usb协议，启动或者关闭MtpService。

### USB各种模式 解释

1、MTP:  
通过MTP这种技术，可以把音乐传到手机里。有了U盘功能为什么还要多此一举呢？因为版权问题，MTP可以把权限文件从电脑上导过去；如果只使用手机的U盘功能，把歌的文件拷过去之后，没有权限文件还是放不了。微软搞这个，就是想控制版权。  
    对于在中国使用，有没有MTP基本上没什么区别，因为我们没有版权环境，网上下的歌在哪都能放。

MTP是微软免费向数码相机，媒体设备等厂商公开的连接技术，这些厂商可以将其写入自己设备的“固件” 当中。MTP的基础是“照片传输协议”(PTP)。 MTP支持将通过升级或者操作系统换代整合进未来的微软操作系统当中。数码相机用户不用再额外安装驱动程序就能够将自己的数码设备连接至微软操作系统的电脑。 利用MTP，数码相机将被Windows识别为和USB闪存驱动器一样的设备

2、PTP  
PTP（Paper Tape Printer）纸带打印机  mtp  媒体传输协议，是基于PTP(Picture Transfer Protocol)协议的扩展，主要用于传输媒体文件，其中有价值的应用就是同步DRM文件的license。目前支持MTP协议的只有 WMP10(Windows Media Player 10)和WMP11(Windows Media Player 11)两个版本，WMP11加入了对Playlist和Album art的支持，在获取媒体文件信息的时候GetObjectPropList代替了WMP10的GetObjectInfo命令。  
3、RNDIS  
RNDIS是指Remote NDIS，基于USB实现RNDIS实际上就是TCP/IP over USB，就是在USB设备上跑TCP/IP，让USB设备看上去像一块网卡。  
4、DM  
手机DM就是将广告客户的打折、促销等产品信息通过短信、彩信、WAP网页以及手机电视信号的形式传送到消费者手中。  
Desktop Manager  同步软件，连接电脑用的，刷机装软件。  
增强售后服务的意思  
5、RMNET  
PPP 和 RMNET 都是上网协议  
RMNET 是老版,速度慢,兼容性高  
PPP      是新版.速度快,兼容性低  
当你在PPP下无法连接GPRS时,可选用RMNET  
信息来源：起点手机论坛   
原文链接：<http://www.qdppc.com/thread-26502-1-1.html>  
5、ADB  
ADB全称Android Debug Bridge, 是android sdk里的一个工具, 用这个工具可以直接操作管理android模拟器或者真实的andriod设备(如G1手机).  
它的主要功能有:  
\* 运行设备的shell(命令行)  
\* 管理模拟器或设备的端口映射  
\* 计算机和设备之间上传/下载文件  
\* 将本地apk软件安装至模拟器或android设备

### 源码目录

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

usbService.java用来管理usb协议，其通过property系统与init.xxx.usb.rc通讯。其中UsbDeviceManager.java以及HostManager.java分别管理device和host的设备。

## USB Device源码分析

### [Accessory模式](https://www.jianshu.com/p/b267c5cedfa9)

在accessory模式下，PAD是作为Device设备的，通常需要一个支持Accessory的Host设备（ADK2012等）配合才能工作，可以参考如下谷歌文档[1](1.%09http:/developer.android.com/guide/topics/connectivity/usb/index.html),[2](1.%09http:/developer.android.com/guide/topics/connectivity/usb/accessory.html),[3](1.%09http:/developer.android.com/guide/topics/connectivity/usb/host.html)



#### Host端

Accessory模式下Host端代码可以参考cts/apps/cts-usb-accessory/cts-usb-accessory.c。这里面模拟了一个Host端的设备。其思路是main()调用system/core/libusbhost/usbhost.c中的usb\_host\_run（）函数，这个函数的主要作用就是去监控/dev/bus/usb/这个目录。

static int usb\_device\_removed(const char \*devname, void\* client\_data)

static int usb\_device\_added(const char \*devname, void\* client\_data)

usb\_host\_run(context, usb\_device\_added, usb\_device\_removed, NULL, NULL);

调用如下接口去查询/dev/bus/usb其中的设备是否支持accessory协议

usb\_device\_control\_transfer(device, USB\_DIR\_IN | USB\_TYPE\_VENDOR, ACCESSORY\_GET\_PROTOCOL, 0, 0, &protocol, sizeof(protocol), 0);

如果支持就调用如下接口尝试将其切换到accessory模式。usb\_device\_control\_transfer(device, USB\_DIR\_OUT | USB\_TYPE\_VENDOR,ACCESSORY\_START, 0, 0, 0, 0, 0);

#### Device端

/kernel/drivers/usb/gadget/f\_accessory.c中收到ACCESSORY\_START这个ioctl后（其实是由usb中断传递上来的）就会发送ACCESSORY=START的uevent。

|  |
| --- |
| static void acc\_start\_work(struct work\_struct \*data)  { |
| char \*envp[2] = { "ACCESSORY=START", NULL }; |
| kobject\_uevent\_env(&acc\_device.this\_device->kobj, KOBJ\_CHANGE, envp); |
| } |

/UsbDeviceManager.java中，接收到uevent后调用startAccessoryMode();--->setCurrentFunctions(xxx)-->设置sys.usb.config这个属性后，就触发init.xxx.usb.rc去通知kernel切换到accessory模式。

这个3399设备的节点为dev/usb\_accessory

### [Mass\_Storage模式](https://www.jianshu.com/p/b267c5cedfa9)

#### UsbDeviceManager收到插入事件

1）UsbDeviceManager监听DEVPATH=/devices/virtual/android\_usb/android0"这个路径的UEVENT，

public UsbHandler(Looper looper){

mUEventObserver.startObserving(USB\_STATE\_MATCH);}

2）在收到状态改变的时候会发出UsbManager.ACTION\_USB\_STATE这个broadcast。其中包含connect，configuration状态以及当前的usb配置的function。

|  |
| --- |
| private void updateUsbState() {  // send a sticky broadcast containing current USB state |
| Intent intent = new Intent(UsbManager.ACTION\_USB\_STATE); |
| intent.addFlags(Intent.FLAG\_RECEIVER\_REPLACE\_PENDING); |
| intent.putExtra(UsbManager.USB\_CONNECTED, mConnected); |
| intent.putExtra(UsbManager.USB\_CONFIGURED, mConfigured); |
|  |
| if (mCurrentFunctions != null) { |
| String[] functions = mCurrentFunctions.split(","); |
| for (int i = 0; i < functions.length; i++) { |
| intent.putExtra(functions[i], true); |
| } |
| } |
|  |
| mContext.sendStickyBroadcastAsUser(intent, UserHandle.ALL); |
| } |

#### MountService

frameworks/base/[services](https://android.googlesource.com/platform/frameworks/base/+/483f3b06ea84440a082e21b68ec2c2e54046f5a6/services) / [java](https://android.googlesource.com/platform/frameworks/base/+/483f3b06ea84440a082e21b68ec2c2e54046f5a6/services/java) / [com](https://android.googlesource.com/platform/frameworks/base/+/483f3b06ea84440a082e21b68ec2c2e54046f5a6/services/java/com) /[android](https://android.googlesource.com/platform/frameworks/base/+/483f3b06ea84440a082e21b68ec2c2e54046f5a6/services/java/com/android?autodive=0) / [server](https://android.googlesource.com/platform/frameworks/base/+/483f3b06ea84440a082e21b68ec2c2e54046f5a6/services/java/com/android/server) /

在MountService收到ACTION\_USB\_STATE这个广播的时候，notifyShareAvailabilityChange（）会调用所有注册的listener的bl.mListener.onUsbMassStorageConnectionChanged(avail);

同时在这里还要处理usb拔出的事件，这里必须把已经shared的盘重新Mount回系统中

#### StorageManager

StorageManager向Mountservice注册了listener

|  |
| --- |
| public StorageManager(Looper tgtLooper) throws RemoteException {  mMountService = IMountService.Stub.asInterface(ServiceManager.getService("mount")); |
| if (mMountService == null) { |
| Log.e(TAG, "Unable to connect to mount service! - is it running yet?"); |
| return; |
| } |
| mTgtLooper = tgtLooper; |
| mBinderListener = new MountServiceBinderListener(); |
| mMountService.registerListener(mBinderListener); |
| } |

#### UI监听事件

其他应用又向StorageManager注册listener，主要有如下地方：

UsbStorageActivity.java ---UMS开关界面UI切换

StorageNotification.java----实现状态栏通知（在onUsbMassStorageConnectionChange（）中实现，这个函数中可以实现自动弹出usbStorageActivity，关键字POP\_UMS\_ACTIVITY\_ON\_CONNECT）

TabletStatusBar.java---------向StorageManager注册listener，用来显示UMS状态栏通知

MtpService.java--------------Mtp状态变化

### 互斥配置

1.多用户和UMS不能共存

----谷歌默认的方式是采用fuse将/data/media模拟成用户盘，这种模式下支持多用户，但是不能支持UMS。如果要支持UMS那么就不能使用fuse，需要划出USER分区，通过Vold来管理。

目前Android4.4的SDK中通过BoradConfig.mk中的BUILD\_WITH\_UMS这个宏来在二者中切换。

BUILD\_WITH\_UMS = true即支持UMS不支持多用户

BUILD\_WITH\_UMS = false即支持多用户但是不支持UMS

2.CDROOM和UMS不能共存

----CDROOM和UMS在kernel中的实现是类似的，都往/sys/class/android\_usb/f\_mass\_storage/lun/file中写入内容来与kernel通讯。

目前Android4.4的SDK中通过BoradConfig.mk中的BUILD\_WITH\_CDROM来控制是否打开CDROOM，BUILD\_WITH\_CDROM\_PATH来设置iso的路径。注意BUILD\_WITH\_UMS和BUILD\_WITH\_CDROM两者应该是互斥的，不能同时设置成true。

## USB Host分析

当usb口作为host使用时，可以连接u盘，鼠标/键盘，usb音响等设备，针对不同的设备由不同的子系统来处理。

### 输入设备

连接鼠标/键盘/手柄等输入设备时，这些外设被当成是输入设备，归输入子系统管理。设备节点在/dev/input下，输入事件由InputReader调用EventHub来读取，具体请看EventHub的分析。

### 音频设备

外接usb音响等音频设备，这些外设被识别成音频设备，设备节点在/dev/snd/下，归音频系统管理。

### [块设备](https://www.jianshu.com/p/b267c5cedfa9)

连接usb存储设备（u盘，硬盘等）时，设备节点在/dev/bus/usb下，由UsbHostManager.java来管理，简单分析如下：

1)frameworks/base/services/java/com/android/server/usb/UsbService.java中的systemReady()调用mHostManager.systemReady()。

2)frameworks/base/services/java/com/android/server/usb/UsbHostManager.java的systemReady中启动一个线程来运行monitorUsbHostBus();

|  |
| --- |
| static void android\_server\_UsbHostManager\_monitorUsbHostBus(JNIEnv \*env, jobject thiz)  { |
| struct usb\_host\_context\* context = usb\_host\_init(); |
| if (!context) { |
| ALOGE("usb\_host\_init failed"); |
| return; |
| } |
| // this will never return so it is safe to pass thiz directly |
| usb\_host\_run(context, usb\_device\_added, usb\_device\_removed, NULL, (void \*)thiz); |
| } |

其中分别调用到了system/core/libusbhost/usbhost.c中的usb\_host\_init(...)和usb\_host\_run(...)

在usb\_host\_init()中，最主要的是初始化context->fd = inotify\_init();，这个会在后面用来监听**/dev/bus/usb**目录的创建和删除

在usb\_host\_run中，主要是添加监控的目录ret = inotify\_add\_watch(context->fd, path, IN\_CREATE | IN\_DELETE);如果发现目录有create或者是delete操作，通知回调函数.

3）在usb\_device\_added（）中

主要是获取usb设备的属性,然后调用UsbHostManager.java中的usbDeviceAdded（），并将这些usb属性传递上去

**method\_usbDeviceAdded** = env->GetMethodID(clazz, "usbDeviceAdded", "(Ljava/lang/String;IIIII[I[I)V");

|  |
| --- |
| env->CallVoidMethod(thiz, **method\_usbDeviceAdded,**  deviceName, vendorId, productId, deviceClass, |
| deviceSubClass, protocol, interfaceArray, endpointArray); |

4）在UsbHostManager.java中的usbDeviceAdded（）中，

主要是创建UsbDevice，如下：

|  |
| --- |
| UsbDevice device = new UsbDevice(deviceName, vendorID, productID,  deviceClass, deviceSubclass, deviceProtocol, interfaces); |
| mDevices.put(deviceName, device); |

getCurrentSettings().deviceAttached(device);

5）UsbSettingsManager. deviceAttached通知app

frameworks/base/services/java/com/android/server/usb/UsbSettingsManager.java中的deviceAttached（）函数，主要是检查系统中是否有安装能处理UsbManager.ACTION\_USB\_DEVICE\_ATTACHED这个广播的activity，并转到该activity.

|  |
| --- |
| public void deviceAttached(UsbDevice device) {  Intent intent = new Intent(UsbManager.ACTION\_USB\_DEVICE\_ATTACHED); |
| intent.putExtra(UsbManager.EXTRA\_DEVICE, device); |
| intent.addFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK); |
|  |
| ArrayList<ResolveInfo> matches; |
| String defaultPackage; |
| synchronized (mLock) { |
| matches = getDeviceMatchesLocked(device, intent); |
| // Launch our default activity directly, if we have one. |
| // Otherwise we will start the UsbResolverActivity to allow the user to choose. |
| defaultPackage = mDevicePreferenceMap.get(new DeviceFilter(device)); |
| } |
|  |
| // Send broadcast to running activity with registered intent |
| mUserContext.sendBroadcast(intent); |
|  |
| // Start activity with registered intent |
| resolveActivity(intent, matches, defaultPackage, device, null); |
| } |

#### [libusbhost](https://www.jianshu.com/p/b267c5cedfa9)

libusbhost主要提供与usb设备通信的接口

struct usb\_device \*usb\_device\_open(const char \*dev\_name)

---打开一个usb设备，在/dev/bus/usb/下

void usb\_device\_close(struct usb\_device \*device)

void usb\_descriptor\_iter\_init(struct usb\_device \*device, struct usb\_descriptor\_iter \*iter)

struct usb\_descriptor\_header \*usb\_descriptor\_iter\_next(struct usb\_descriptor\_iter \*iter)

--获取descriptor

int usb\_device\_claim\_interface(struct usb\_device \*device, unsigned int interface)

----claim一个interface用于通讯

int usb\_device\_release\_interface(struct usb\_device \*device, unsigned int interface)

int usb\_device\_bulk\_transfer(struct usb\_device \*device, --------传输数据

int endpoint,

void\* buffer,

int length,

unsigned int timeout)

int usb\_device\_control\_transfer(struct usb\_device \*device, ----------控制指令

int requestType,

int request,

int value,

int index,

void\* buffer,

int length,

unsigned int timeout)

在java代码中可以通过一下文件中提供的接口来访问usb设备。

frameworks/base/core/java/android/hardware/usb/UsbDeviceConnection.java

### [rndis](https://baike.baidu.com/item/rndis)

上至下解析，android5.0以上的系统自带了ethernet service，默认开机就会启动，默认ip获取方式是动态分配

#### EthernetService启动

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

|  |
| --- |
| public final class SystemServer {  private static final String ETHERNET\_SERVICE\_CLASS =  **"com.android.server.ethernet.EthernetService**";  public static void main(String[] args) {  new SystemServer().run();  }  private void run() {  startOtherServices();  }  private void startOtherServices() {  if (mPackageManager.hasSystemFeature(PackageManager.FEATURE\_ETHERNET)) {  mSystemServiceManager.startService(ETHERNET\_SERVICE\_CLASS);  }  }  } |

#### EthernetService

[frameworks/opt/net/ethernet/java/com/android/server/ethernet/EthernetService.java](https://link.jianshu.com/?t=http://androidxref.com/5.1.1_r6/xref/frameworks/opt/net/ethernet/java/com/android/server/ethernet/EthernetService.java)

|  |
| --- |
| public final class EthernetService extends SystemService {  private static final String TAG = "EthernetService";  final EthernetServiceImpl mImpl;  public EthernetService(Context context) {  super(context);  mImpl = new EthernetServiceImpl(context);  }  @Override  public void onStart() {  Log.i(TAG, "Registering service " + Context.ETHERNET\_SERVICE);  publishBinderService(Context.ETHERNET\_SERVICE, mImpl);  }  @Override  public void onBootPhase(int phase) {  if (phase == SystemService.PHASE\_SYSTEM\_SERVICES\_READY) {  mImpl.start();  }  }  } |

#### EthernetServiceImpl

frameworks/opt/net/ethernet/java/com/android/server/ethernet/EthernetServiceImpl.java

|  |
| --- |
| public class EthernetServiceImpl extends IEthernetManager.Stub {  public EthernetServiceImpl(Context context) {  mContext = context;  Log.i(TAG, "Creating EthernetConfigStore");  mEthernetConfigStore = new EthernetConfigStore();  mIpConfiguration = mEthernetConfigStore.readIpAndProxyConfigurations();  Log.i(TAG, "Read stored IP configuration: " + mIpConfiguration);  IBinder b = ServiceManager.getService(Context.NETWORKMANAGEMENT\_SERVICE);  mNMService = INetworkManagementService.Stub.asInterface(b);  **mTracker = new EthernetNetworkFactory();**  }  public void start() {  mCM = (ConnectivityManager) mContext.getSystemService(Context.CONNECTIVITY\_SERVICE);  HandlerThread handlerThread = new HandlerThread("EthernetServiceThread");  handlerThread.start();  mHandler = new Handler(handlerThread.getLooper());  mEnabled = getPersistedState();  Log.i(TAG, "Ethernet Persisted Enabled " + mEnabled);  setState(mEnabled); //重要  }  public synchronized void setState(int state) {  enforceChangePermission();  Log.i(TAG, "setState from mState=" + mState + " to state=" + state);  if (mState != state) {  mState = state;  if (state == EthernetManager.ETHERNET\_STATE\_DISABLED) {  setPersistedState(EthernetManager.ETHERNET\_STATE\_DISABLED);  mTracker.stopInterface();  mStarted.set(false);  } else {  setPersistedState(EthernetManager.ETHERNET\_STATE\_ENABLED);  mTracker.stop();  mTracker.start(mContext, mHandler);  mStarted.set(true);  }  }  }  } |

#### EthernetNetworkFactory

frameworks/opt/net/ethernet/java/com/android/server/ethernet/EthernetNetworkFactory.java

|  |
| --- |
| class EthernetNetworkFactory {  EthernetNetworkFactory() {  mNetworkInfo = new NetworkInfo(ConnectivityManager.TYPE\_ETHERNET, 0, NETWORK\_TYPE, "");  mLinkProperties = new LinkProperties();  initNetworkCapabilities();  }  public synchronized void start(Context context, Handler target) {  IBinder b = ServiceManager.getService(Context.NETWORKMANAGEMENT\_SERVICE);  mNMService = INetworkManagementService.Stub.asInterface(b);  mEthernetManager = (EthernetManager) context.getSystemService(Context.ETHERNET\_SERVICE);  mFactory = new LocalNetworkFactory(NETWORK\_TYPE, context, target.getLooper());  mFactory.setCapabilityFilter(mNetworkCapabilities);  mFactory.setScoreFilter(-1); // this set high when we have an iface  mFactory.register();  /\*  public void register() {  if (DBG) log("Registering NetworkFactory");  if (mMessenger == null) {  mMessenger = new Messenger(this);  ConnectivityManager.from(mContext).registerNetworkFactory(mMessenger, LOG\_TAG);  }  }  frameworks/base/services/core/java/com/android/server/ConnectivityService.java  public void registerNetworkFactory(Messenger messenger, String name) {  NetworkFactoryInfo nfi = new NetworkFactoryInfo(name, messenger, new AsyncChannel());  mHandler.sendMessage(mHandler.obtainMessage(EVENT\_REGISTER\_NETWORK\_FACTORY, nfi));  }  private class InternalHandler extends Handler {  public void handleMessage(Message msg) {  case EVENT\_REGISTER\_NETWORK\_FACTORY: {  handleRegisterNetworkFactory((NetworkFactoryInfo)msg.obj);  break;  }  }  }  private void handleRegisterNetworkFactory(NetworkFactoryInfo nfi) {  if (DBG) log("Got NetworkFactory Messenger for " + nfi.name);  mNetworkFactoryInfos.put(nfi.messenger, nfi);  nfi.asyncChannel.connect(mContext, mTrackerHandler, nfi.messenger);  }  \*/  mInterfaceObserver = new InterfaceObserver();  try {  mNMService.registerObserver(mInterfaceObserver);  } catch (RemoteException e) {  Log.e(TAG, "Could not register InterfaceObserver " + e);  }  updateInterfaceState(iface, true); //注册  }  private void updateInterfaceState(String iface, boolean up) {  updateAgent();  mFactory.setScoreFilter(up ? NETWORK\_SCORE : -1); //设置scroe值；这个是网络优先级判断依据  }  public void updateAgent() {  mNetworkAgent.sendNetworkInfo(mNetworkInfo);  }  } |

#### ETHERNET监听NETD进程的socket

##### init.rc

service netd /system/bin/netd

class main

socket netd stream 0660 root system

socket dnsproxyd stream 0660 root inet

socket mdns stream 0660 root system

socket fwmarkd stream 0660 root inet

##### [NetworkManagementService.java](https://link.jianshu.com/?t=http://androidxref.com/5.1.1_r6/xref/frameworks/base/services/core/java/com/android/server/NetworkManagementService.java)

|  |
| --- |
| public static NetworkManagementService create(Context context) throws InterruptedException {  return create(context, NETD\_SOCKET\_NAME);  /\*  private static final String NETD\_SOCKET\_NAME = "netd";  \*/  }  private NetworkManagementService(Context context, String socket) {  mConnector = new NativeDaemonConnector(  new NetdCallbackReceiver(), socket, 10, NETD\_TAG, 160, wl,  FgThread.get().getLooper());  /\*  frameworks/base/services/core/java/com/android/server/NativeDaemonConnector.java  public void run() {  mCallbackHandler = new Handler(mLooper, this);  while (true) {  try {  listenToSocket();  } catch (Exception e) {  loge("Error in NativeDaemonConnector: " + e);  SystemClock.sleep(5000);  }  }  }  private void listenToSocket() throws IOException {  LocalSocketAddress address = determineSocketAddress();  mCallbackHandler.sendMessage();  }  public boolean handleMessage(Message msg) {  mCallbacks.onEvent  }  \*/  }  private class NetdCallbackReceiver implements INativeDaemonConnectorCallbacks {  public boolean onEvent(int code, String raw, String[] cooked) {  notifyInterfaceAdded(cooked[3]);  ......  notifyInterfaceRemoved(cooked[3]);  ......  notifyInterfaceStatusChanged(cooked[3], cooked[4].equals("up"));  ......  notifyInterfaceLinkStateChanged(cooked[3], cooked[4].equals("up"));  }  private void notifyInterfaceLinkStateChanged(String iface, boolean up) {  mObservers.getBroadcastItem(i).interfaceLinkStateChanged(iface, up);  }  } |

##### EthernetNetworkFactory.InterfaceObserver

frameworks/opt/net/ethernet/java/com/android/server/ethernet/EthernetNetworkFactory.java

|  |
| --- |
| private class InterfaceObserver extends BaseNetworkObserver {  public void interfaceLinkStateChanged(String iface, boolean up) {  updateInterfaceState(iface, up);  }  }  private void updateInterfaceState(String iface, boolean up) {  mFactory.setScoreFilter(up ? NETWORK\_SCORE : -1);  } |

##### NetworkFactory

frameworks/base/core/java/android/net/NetworkFactory.java

|  |
| --- |
| public void setScoreFilter(int score) {  sendMessage(obtainMessage(CMD\_SET\_SCORE, score, 0));  }  public void handleMessage(Message msg) {  case CMD\_SET\_SCORE: {  handleSetScore(msg.arg1);  break;  }  }  private void handleSetScore(int score) {  mScore = score;  evalRequests();  }  private void evalRequests() {  for (int i = 0; i < mNetworkRequests.size(); i++) {  NetworkRequestInfo n = mNetworkRequests.valueAt(i);  evalRequest(n);  }  }  private void evalRequest(NetworkRequestInfo n) {  needNetworkFor(n.request, n.score);  }  protected void needNetworkFor(NetworkRequest networkRequest, int score) {  if (++mRefCount == 1) startNetwork();  } |

##### EthernetNetworkFactory. LocalNetworkFactory()

frameworks/opt/net/ethernet/java/com/android/server/ethernet/EthernetNetworkFactory.java

|  |
| --- |
| private class LocalNetworkFactory extends NetworkFactory {  LocalNetworkFactory(String name, Context context, Looper looper) {  super(looper, context, name, new NetworkCapabilities());  }  protected void startNetwork() {  onRequestNetwork();  }  protected void stopNetwork() {  }  }  public void onRequestNetwork() {  Thread dhcpThread = new Thread(new Runnable() {  public void run() {  DhcpResults dhcpResults = new DhcpResults(); //DHCP相关  if (!NetworkUtils.runDhcp(mIface, dhcpResults)) {  /\*  frameworks/base/core/java/android/net/NetworkUtils.java  public native static boolean runDhcp(String interfaceName, DhcpResults dhcpResults);  frameworks/base/core/jni/android\_net\_NetUtils.cp  static jboolean android\_net\_utils\_runDhcp(JNIEnv\* env, jobject clazz, jstring ifname, jobject info)  {  return android\_net\_utils\_runDhcpCommon(env, clazz, ifname, info, false);  }  static jboolean android\_net\_utils\_runDhcpCommon(JNIEnv\* env, jobject clazz, jstring ifname,  jobject dhcpResults, bool renew)  {  if (renew) {  result = ::dhcp\_do\_request\_renew(nameStr, ipaddr, gateway, &prefixLength,  dns, server, &lease, vendorInfo, domains, mtu);  } else {  result = ::dhcp\_do\_request(nameStr, ipaddr, gateway, &prefixLength,  dns, server, &lease, vendorInfo, domains, mtu);  }  }  system/core/libnetutils/dhcp\_utils.c  DHCP Client和DHCP server（system/bin/dhcpd进程）通过property\_get/set 共享内存来共享信息  \*/  Log.e(TAG, "DHCP request error:" + NetworkUtils.getDhcpError());  // set our score lower than any network could go  // so we get dropped.  mFactory.setScoreFilter(-1);  return;  }  mNetworkAgent = new NetworkAgent(mFactory.getLooper(), mContext,  NETWORK\_TYPE, mNetworkInfo, mNetworkCapabilities, mLinkProperties,  NETWORK\_SCORE)  }  });  dhcpThread.start();  } |

配置完成以后ConnectivityService向EthernetManager发送CONNECTIVITY\_ACTION\_IMMEDIATE的广播；EthernetManager接收到该广播以后向应用程序发送ETHERNET\_INTERFACE\_CONF\_CHANGED广播。否则；应用程序将TIMEOUT。

|  |
| --- |
| public static final String EXTRA\_ETHERNET\_STATE = "ETHERNET\_state";  public static final String ETHERNET\_INTERFACE\_CONF\_CHANGED =  "android.net.ethernet.ETHERNET\_INTERFACE\_CONF\_CHANGED"; //add by tank  private void sendEthBroadcast(String action, boolean state) {  String bootStr = SystemProperties.get("sys.boot\_completed");  Log.d(TAG, "sendEthBroadcast -->: " + bootStr);  if(bootStr.equals("1")) { //boot complete  Intent intent = new Intent(action);  intent.putExtra(EXTRA\_ETHERNET\_STATE, state);  Log.d(TAG, "sendEthBroadcast --> action= " + action + " state=" + state);  mContext.sendBroadcast(intent);  }  }  //连接成功调用如下：  sendEthBroadcast(ETHERNET\_INTERFACE\_CONF\_CHANGED, true);  //连接失败调用如下：  sendEthBroadcast(ETHERNET\_INTERFACE\_CONF\_CHANGED, false); |

#### UI监听

网络监听一：设置 packages/apps/Settings/src/com/android/settings/ethernet/EthernetEnabler.java

设置项网络按钮类定义

网络监听二：statusbar frameworks/base/packages/SystemUI/src/com/android/systemui/statusbar/policy/NetworkController.java  
NetworkController本身是个BroadcastReceiver，其中关于网络状态变化的监听消息为EthernetManager.NETWORK\_STATE\_CHANGED\_ACTION，可以猜测这个消息是framework发出来的，往下看。

# Kernel层

<https://android.googlesource.com/kernel/tegra/+/050d171d5e47a539c616ffea9e2da82ef16bb5bc/drivers/usb/gadget/f_accessory.c>

/kernel/drivers/usb/gadget/f\_accessory.c中收到ACCESSORY\_START这个ioctl后（其实是由usb中断传递上来的）就会发送ACCESSORY=START的uevent。

## acc\_ctrlrequest

static int acc\_ctrlrequest(struct usb\_composite\_dev \*cdev,

const struct usb\_ctrlrequest \*ctrl)

{

struct acc\_dev \*dev = \_acc\_dev;

int value = -EOPNOTSUPP;

struct acc\_hid\_dev \*hid;

int offset;

u8 b\_requestType = ctrl->bRequestType;

u8 b\_request = ctrl->bRequest;

u16 w\_index = le16\_to\_cpu(ctrl->wIndex);

u16 w\_value = le16\_to\_cpu(ctrl->wValue);

u16 w\_length = le16\_to\_cpu(ctrl->wLength);

unsigned long flags;

if (b\_requestType == (USB\_DIR\_OUT | USB\_TYPE\_VENDOR)) {

if (b\_request == ACCESSORY\_START) {

dev->start\_requested = 1;

schedule\_delayed\_work(

&dev->start\_work, msecs\_to\_jiffies(10));

value = 0;

}

## acc\_start\_work

|  |
| --- |
| static void acc\_start\_work(struct work\_struct \*data)  { |
| char \*envp[2] = { "ACCESSORY=START", NULL }; |
| kobject\_uevent\_env(&acc\_device.this\_device->kobj, KOBJ\_CHANGE, envp); |
| } |

UsbDeviceManager.java中，接收到uevent后调用startAccessoryMode();--->setCurrentFunctions(xxx)-->设置sys.usb.config这个属性后，就触发init.xxx.usb.rc去通知kernel切换到accessory模式。

这个3399设备的节点为dev/usb\_accessory

# FW层

相关实例：

frameworks/base/libs/usb/tests/AccessoryChat/AndroidManifest.xml

\frameworks\base\tests\AccessoryDisplay\source\src\com\android\accessorydisplay\source\SourceActivity.java

cts\apps\CtsVerifier\src\com\android\cts\verifier\usb\UsbAccessoryTestActivity.java

api定义：

android/hardware/usb/UsbManager.java

**public static final** String ***ACTION\_USB\_ACCESSORY\_ATTACHED*** =**"android.hardware.usb.action.USB\_ACCESSORY\_ATTACHED"**;

相关源码：

UsbSettingsManager.java

**mHandler** = **new** UsbHandler(FgThread.*get*().getLooper());

## 注册usb事件

### UsbHandler

**private static final** String ***USB\_STATE\_MATCH*** =  
 **"DEVPATH=/devices/virtual/android\_usb/android0"**;  
**private static final** String ***ACCESSORY\_START\_MATCH*** =  
 **"DEVPATH=/devices/virtual/misc/usb\_accessory"**;

*// Watch for USB configuration changes***mUEventObserver**.startObserving(***USB\_STATE\_MATCH***);  
**mUEventObserver**.startObserving(***ACCESSORY\_START\_MATCH***);

### mUEventObserver

*/\*  
 \* Listens for uevent messages from the kernel to monitor the USB state  
 \*/***private final** UEventObserver **mUEventObserver** = **new** UEventObserver() {  
 @Override  
 **public void** onUEvent(UEventObserver.UEvent event) {  
 **if** (***DEBUG***) Slog.*v*(***TAG***, **"USB UEVENT: "** + event.toString());  
  
 String state = event.get(**"USB\_STATE"**);  
 String accessory = event.get(**"ACCESSORY"**);  
 **if** (state != **null**) {  
 **mHandler**.updateState(state);  
 } **else if** (**"START"**.equals(accessory)) {  
 **if** (***DEBUG***) Slog.*d*(***TAG***, **"got accessory start"**);  
 startAccessoryMode();  
 }  
 }  
};

#### startAccessoryMode

UsbDeviceManager.java中，接收到uevent后调用startAccessoryMode();--->setCurrentFunctions(xxx)-->设置sys.usb.config这个属性后，就触发init.xxx.usb.rc去通知kernel切换到accessory模式。

这个3399设备的节点为dev/usb\_accessory

**private void** startAccessoryMode() {  
 **if** (!**mHasUsbAccessory**) **return**;  
  
 **mAccessoryStrings** = nativeGetAccessoryStrings();  
 **boolean** enableAudio = (nativeGetAudioMode() == ***AUDIO\_MODE\_SOURCE***);  
 *// don't start accessory mode if our mandatory strings have not been set* **boolean** enableAccessory = (**mAccessoryStrings** != **null** &&  
 **mAccessoryStrings**[UsbAccessory.MANUFACTURER\_STRING] != **null** &&  
 **mAccessoryStrings**[UsbAccessory.MODEL\_STRING] != **null**);  
 String functions = **null**;  
  
 **if** (enableAccessory && enableAudio) {  
 functions = UsbManager.USB\_FUNCTION\_ACCESSORY + **","** + UsbManager.USB\_FUNCTION\_AUDIO\_SOURCE;  
 } **else if** (enableAccessory) {  
 functions = UsbManager.USB\_FUNCTION\_ACCESSORY;  
 } **else if** (enableAudio) {  
 functions = UsbManager.USB\_FUNCTION\_AUDIO\_SOURCE;  
 }  
  
 **if** (functions != **null**) {  
 **mAccessoryModeRequestTime** = SystemClock.*elapsedRealtime*();  
 setCurrentFunctions(functions);  
 }  
}

#### setCurrentFunctions

**public void** setCurrentFunctions(String functions) {  
 **if** (***DEBUG***) Slog.*d*(***TAG***, **"setCurrentFunctions("** + functions + **")"**);  
 **mHandler**.sendMessage(***MSG\_SET\_CURRENT\_FUNCTIONS***, functions);  
}

**case** MSG\_SET\_CURRENT\_FUNCTIONS:  
 String functions = (String)msg.obj;  
 setEnabledFunctions(functions, **false**);  
 **break**;

#### setEnabledFunctions

**private void** setEnabledFunctions(String functions, **boolean** forceRestart) {  
 **if** (***DEBUG***) Slog.*d*(***TAG***, **"setEnabledFunctions functions="** + functions + **", "** + **"forceRestart="** + forceRestart);  
  
 *// Try to set the enabled functions.* **final** String oldFunctions = **mCurrentFunctions**;  
 **final boolean** oldFunctionsApplied = **mCurrentFunctionsApplied**;  
 **if** (trySetEnabledFunctions(functions, forceRestart)) {  
 **return**;  
 }  
  
 *// Didn't work. Try to revert changes.  
 // We always reapply the policy in case certain constraints changed such as  
 // user restrictions independently of any other new functions we were  
 // trying to activate.* **if** (oldFunctionsApplied && !oldFunctions.equals(functions)) {  
 Slog.*e*(***TAG***, **"Failsafe 1: Restoring previous USB functions."**);  
 **if** (trySetEnabledFunctions(oldFunctions, **false**)) {  
 **return**;  
 }  
 }

#### trySetEnabledFunctions

**private boolean** trySetEnabledFunctions(String functions, **boolean** forceRestart) {  
 **if** (functions == **null**) {  
 functions = getDefaultFunctions();  
 }  
 functions = applyAdbFunction(functions);  
 functions = applyOemOverrideFunction(functions);  
  
 **if** (!**mCurrentFunctions**.equals(functions) || !**mCurrentFunctionsApplied** || forceRestart) {  
 Slog.*i*(***TAG***, **"Setting USB config to "** + functions);  
 **mCurrentFunctions** = functions;  
 **mCurrentFunctionsApplied** = **false**;  
  
 *// Kick the USB stack to close existing connections.* setUsbConfig(UsbManager.USB\_FUNCTION\_NONE);  
  
 *// Set the new USB configuration.* **if** (!setUsbConfig(functions)) {  
 Slog.*e*(***TAG***, **"Failed to switch USB config to "** + functions);  
 **return false**;  
 }  
  
 **mCurrentFunctionsApplied** = **true**;  
 }  
 **return true**;  
}

#### etUsbConfig

**private boolean** setUsbConfig(String config) {  
 **if** (***DEBUG***) Slog.*d*(***TAG***, **"setUsbConfig("** + config + **")"**);  
 *// set the new configuration  
 // we always set it due to b/23631400, where adbd was getting killed  
 // and not restarted due to property timeouts on some devices* SystemProperties.*set*(***USB\_CONFIG\_PROPERTY***, config);  
 **return** waitForState(config);  
}

## 内部路由事件

### updateState(String state)

Message msg = Message.*obtain*(**this**, ***MSG\_UPDATE\_STATE***);  
msg.**arg1** = connected;  
msg.**arg2** = configured;  
*// debounce disconnects to avoid problems bringing up USB tethering*sendMessageDelayed(msg, (connected == 0) ? ***UPDATE\_DELAY*** : 0);

### updateCurrentAccessory

**mCurrentAccessory** = **new** UsbAccessory(**mAccessoryStrings**);  
Slog.*d*(***TAG***, **"entering USB accessory mode: "** + **mCurrentAccessory**);  
*// defer accessoryAttached if system is not ready***if** (**mBootCompleted**) {  
 getCurrentSettings().accessoryAttached(**mCurrentAccessory**);  
} *// else handle in boot completed*

com/android/server/usb/UsbSettingsManager.java

## 查找接收者UsbSettingsManager .accessoryAttached

**public void** accessoryAttached(UsbAccessory accessory) {  
 Intent **intent** = **new** Intent(UsbManager.***ACTION\_USB\_ACCESSORY\_ATTACHED***);  
 intent.putExtra(UsbManager.***EXTRA\_ACCESSORY***, accessory);  
 intent.addFlags(Intent.***FLAG\_ACTIVITY\_NEW\_TASK***);  
  
 ArrayList<ResolveInfo> matches;  
 String defaultPackage;  
 **synchronized** (**mLock**) {  
 matches = getAccessoryMatchesLocked(accessory, intent);  
 *// Launch our default activity directly, if we have one.  
 // Otherwise we will start the UsbResolverActivity to allow the user to choose.* defaultPackage = **mAccessoryPreferenceMap**.get(**new** AccessoryFilter(accessory));  
 }  
  
 resolveActivity(**intent**, matches, defaultPackage, **null**, accessory);  
}

### getAccessoryMatchesLocked

**private final** ArrayList<ResolveInfo> getAccessoryMatchesLocked(  
 UsbAccessory accessory, Intent intent) {  
 ArrayList<ResolveInfo> matches = **new** ArrayList<ResolveInfo>();  
 List<ResolveInfo> resolveInfos = **mPackageManager**.queryIntentActivities(intent,  
 PackageManager.***GET\_META\_DATA***);  
 **int** count = resolveInfos.size();  
 **for** (**int** i = 0; i < count; i++) {  
 ResolveInfo resolveInfo = resolveInfos.get(i);  
 **if** (packageMatchesLocked(resolveInfo, intent.getAction(), **null**, accessory)) {  
 matches.add(resolveInfo);  
 }  
 }  
 **return** matches;  
}

### packageMatchesLocked

**if** (accessory != **null** && **"usb-accessory"**.equals(tagName)) {  
 AccessoryFilter filter = AccessoryFilter.*read*(parser);  
 **if** (filter.matches(accessory)) {  
 **return true**;  
 }  
}

### AccessoryFilter

**public static** AccessoryFilter read(XmlPullParser parser)  
 **throws** XmlPullParserException, IOException {  
 String manufacturer = **null**;  
 String model = **null**;  
 String version = **null**;  
  
 **int** count = parser.getAttributeCount();  
 **for** (**int** i = 0; i < count; i++) {  
 String name = parser.getAttributeName(i);  
 String value = parser.getAttributeValue(i);  
  
 **if** (**"manufacturer"**.equals(name)) {  
 manufacturer = value;  
 } **else if** (**"model"**.equals(name)) {  
 model = value;  
 } **else if** (**"version"**.equals(name)) {  
 version = value;  
 }  
 }  
 **return new** AccessoryFilter(manufacturer, model, version);  
}

**public boolean** matches(AccessoryFilter f) {  
 **if** (**mManufacturer** != **null** && !f.**mManufacturer**.equals(**mManufacturer**)) **return false**;  
 **if** (**mModel** != **null** && !f.**mModel**.equals(**mModel**)) **return false**;  
 **if** (**mVersion** != **null** && !f.**mVersion**.equals(**mVersion**)) **return false**;  
 **return true**;  
}

## 接收者转发

### FW层路由UsbSettingsManager. resolveActivity

*// don't show the resolver activity if there are no choices available***if** (count == 0) {  
 **if** (accessory != **null**) {  
 String uri = accessory.getUri();  
 **if** (uri != **null** && uri.length() > 0) {  
 *// display URI to user  
 // start UsbResolverActivity so user can choose an activity* Intent dialogIntent = **new** Intent();  
 dialogIntent.setClassName(**"com.android.systemui"**,  
 **"com.android.systemui.usb.UsbAccessoryUriActivity"**);  
 dialogIntent.addFlags(Intent.***FLAG\_ACTIVITY\_NEW\_TASK***);  
 dialogIntent.putExtra(UsbManager.***EXTRA\_ACCESSORY***, accessory);  
 dialogIntent.putExtra(**"uri"**, uri);  
 **try** {  
 **mUserContext**.startActivityAsUser(dialogIntent, **mUser**);  
 } **catch** (ActivityNotFoundException e) {  
 Slog.*e*(***TAG***, **"unable to start UsbAccessoryUriActivity"**);  
 }  
 }  
 }  
  
 *// do nothing* **return**;  
}

**if** (defaultRI != **null**) {  
 *// grant permission for default activity* **if** (device != **null**) {  
 grantDevicePermission(device, defaultRI.**activityInfo**.**applicationInfo**.**uid**);  
 } **else if** (accessory != **null**) {  
 grantAccessoryPermission(accessory, defaultRI.**activityInfo**.**applicationInfo**.**uid**);  
 }  
  
 *// start default activity directly* **try** {  
 intent.setComponent(  
 **new** ComponentName(defaultRI.**activityInfo**.**packageName**,  
 defaultRI.**activityInfo**.**name**));  
 **mUserContext**.startActivityAsUser(intent, **mUser**);  
 } **catch** (ActivityNotFoundException e) {  
 Slog.*e*(***TAG***, **"startActivity failed"**, e);  
 }  
} **else** {  
 Intent resolverIntent = **new** Intent();  
 resolverIntent.addFlags(Intent.***FLAG\_ACTIVITY\_NEW\_TASK***);  
  
  **if (count == 1) {**  
 *// start UsbConfirmActivity if there is only one choice* resolverIntent.setClassName(**"com.android.systemui"**,  
 **"com.android.systemui.usb.UsbConfirmActivity"**);  
 resolverIntent.putExtra(**"rinfo"**, matches.get(0));  
  
 **if** (device != **null**) {  
 resolverIntent.putExtra(UsbManager.***EXTRA\_DEVICE***, device);  
 } **else** {  
 resolverIntent.putExtra(UsbManager.***EXTRA\_ACCESSORY***, accessory);  
 }  
 } **else** {  
 *// start UsbResolverActivity so user can choose an activity* resolverIntent.setClassName(**"com.android.systemui"**,  
 **"com.android.systemui.usb.UsbResolverActivity"**);  
  **resolverIntent.putParcelableArrayListExtra("rlist", matches);**  
 resolverIntent.putExtra(Intent.***EXTRA\_INTENT***, intent);  
 }  
 **try** {  
 **mUserContext**.startActivityAsUser(resolverIntent, **mUser**);  
 } **catch** (ActivityNotFoundException e) {  
 Slog.*e*(***TAG***, **"unable to start activity "** + resolverIntent);  
 }  
}

### 系统APP层

Systemui.UsbResolverActivity

### 第三方app层

## 架构图

时序图

## TODO

**public class** UsbConfirmActivity **extends** AlertActivity

## 实践

<**activity  
 android:name=".main.activity.DJIAoaActivity"  
 android:configChanges="orientation|screenSize|keyboardHidden|keyboard"** >  
 <**intent-filter**>  
 <**action android:name="android.hardware.usb.action.USB\_ACCESSORY\_ATTACHED"** />  
 </**intent-filter**>  
  
 <**meta-data  
 android:name="android.hardware.usb.action.USB\_ACCESSORY\_ATTACHED"  
 android:resource="@xml/accessory\_filter"** />  
</**activity**>

<**resources**>  
 <**usb-accessory manufacturer="Google, Inc." model="AccessoryChat" type="Sample Program" version="1.0"** />  
</**resources**>

*<?***xml version="1.0" encoding="utf-8"***?>*<**resources**>  
 <**usb-accessory model="T600" manufacturer="DJI"**/>  
</**resources**>

# Usb Debug启动流程

代码在/framework/base/service/com/android/server/usb/UsbService.java里去初始化一个叫UsbDeviceManager的类，相关文件也在同一个目录。

在UsbService.java中

public UsbService(Context context) {

mContext = context;

final PackageManager pm = mContext.getPackageManger();

....

if (new File("/sys/class/android\_usb").exists())

{

mDeviceManager = new UsbDeviceManager();

}

...

}

中的new UsbDeviceManager()就是关键代码。我的项目中由于/sys/class目录下没有android\_usb所以UsbDeviceManager初始化函数没有执行，所以会无法启动ADB。刚才/sys/class/android\_usb应该是一个和驱动有关的目录，这里不做详细介绍。　为什么UsbDeviceManager能够启动ADB呢？额。。。那是因为他监听了settings的数据库，然后做了响应动作。我们来看下UsbDeviceManager.java中做了神马事情。

private class AdbSettingsObserver extends ContentObserver {

...

@Override

boolean enable = (Settings.Global.getInt(mContextResolver, Settings.Global.ADB\_ENABLED, 0) > 0)

mHandler.sendMessage(MSG\_ENABLE\_ADB, enable);

}

而后续代码中注册了数据库的监听mContentResolver.registerContentResolver(Settings.Global.getUriFor(Settings.Global.ADB\_ENABLED), false, new AdbSettingsObserver);

来监听最上面提到的Settings.Global\_ADB\_ENABLED这个数据库字段的改变来产生相应动作。

具体动作就是对android properties属性进行设置，关键字段是persist.sys.usb.config，如果启动ADB，则通过SystemProperties.get("persisit.sys.usb.config", "adb");来启动ADB。

你可以手动通过setprop在终端中设置相应的值先试试看看。前提是你必须有root权限。

## Ref

[USB各种模式 解释](http://www.cnblogs.com/zxc2man/p/5610812.html)

[Android的USB系统简单分析之一](https://www.jianshu.com/p/b267c5cedfa9)

[Linux下使用USB模拟ACM串口设备](http://blog.csdn.net/mapeng892020/article/details/54095037)

[Android Ethernet从上至下解析一](http://www.bkjia.com/Androidjc/1045168.html)

[android 5.1 Ethernet开发相关](https://www.jianshu.com/p/b52cda6c0b58)

[android5.1 增加ethernet设置(DHCP与Static ip)-UI修改](http://blog.csdn.net/hclydao/article/details/50972932)