DLAN Platinum 适配DMR应用

|  |  |  |
| --- | --- | --- |
| 序号 | 描述 | 姓名/时间 |
| 01 | DLAN Platinum 适配DMR应用 | 刘志保 / 2019-03-21 |
|  |  |  |
|  |  |  |
| 参考 | https://www.cnblogs.com/pngcui/p/7153933.html |  |

前面的已经将platinum 库流程编译通了,但是离使用还很远.这里先做适配DMR终端应用.

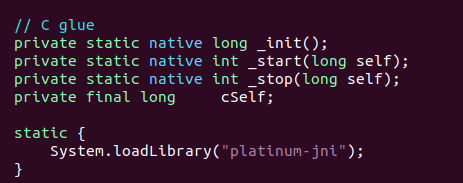
1、DMS， Digital Media Server的缩写，把本设备内的多媒体文件（自己控制需要共享哪些文件）到DLNA服务端，  
2、DMP，Digital Media Player的缩写，可以搜索局域网内，所有DMS提供的多媒体内容，并在本设备播放。  
3、DMC，Digital MediaController的缩写，作为控制DMP搜索播放DMS的内容，与DMP不同，DMC可以控制局域网内的其他设备的DMS和DMP之间的交互，比如，pad作为DMS，机顶盒做DMP，手机用做DMC，三个设备在同一局域网内，手机可以直接控制让机顶盒播放pad里面共享出来的多媒体。  
　　DMR，Digital Render的缩写，作用：DMP+DMR。.

通俗一点的说,DMR即获取共享媒体资源端的音源并且播放,还增加控制播放方式(下一首下一首等);比如: QQ音乐APP是DMS端,开发一个DMR端去获取QQ音乐资源,并且控制播放过程(暂停,快进等).这个DMR端可以应用到智能音箱,PAD,或者电视等设备.

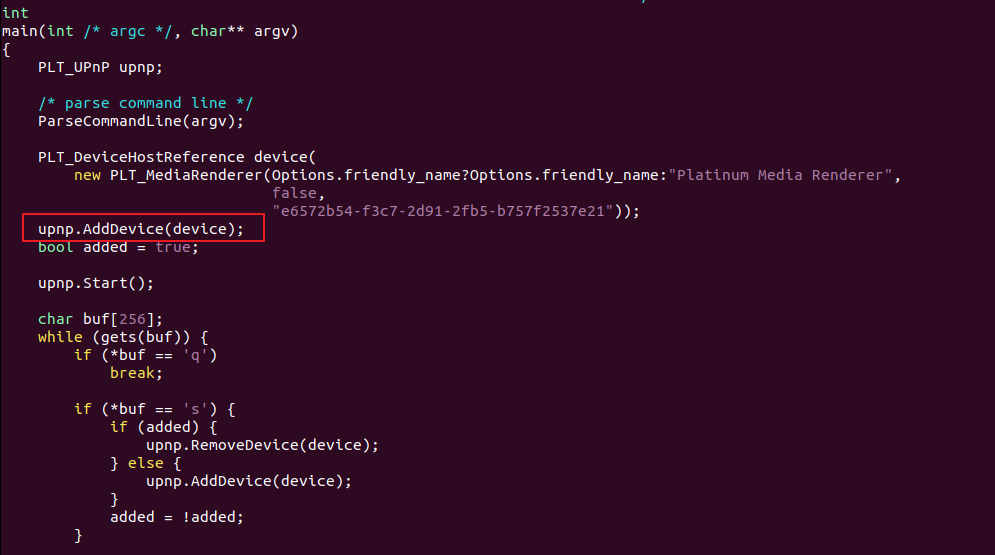
回到程序,我们编译了so,看看,jni的Java接口在:



它提供的接口如下:



这个就尴尬了,实际项目如果仅仅调用这个就能够使用,那就太简单了,但是实际不行,很遗憾,也觉得挺奇怪的,这个开源库为什么不能够提供完整的.但是,它提供了C++版本的测试使用样例,如下:



实际上它在操作之前还添加了设备,但是给android的jni却没有添加设备这个接口操作.

这个地方涉及到一个工作过程:

DMR 1

DMS

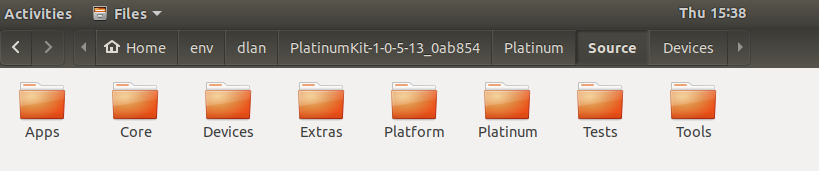
DMR 2

DMR 3

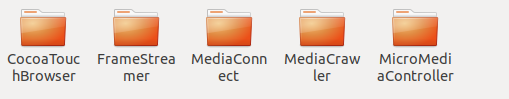
DMR端设备作为UPNP设备,要把自己的NAME和UUID 两个参数设置到UPNP服务中,DMS然后通过多点网络广播向整个局域网内进行广播,进行搜索UPNP设备(这里即DMR设备);

所以从样例代码和工作原理,作为UPNP设备必须要将自己的NAME和UUID公开被用于DMS搜索,并且建立连接.

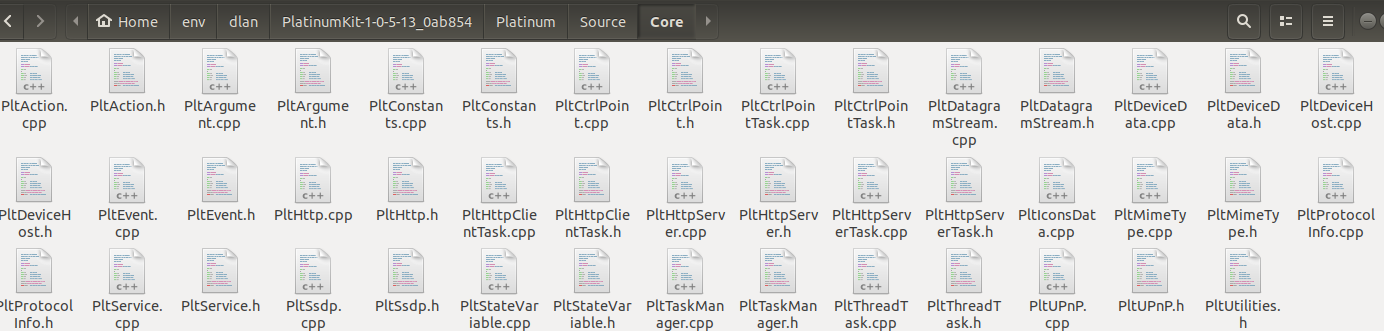
综合上述 : UPNP.java的jni接口显然不能够满足开发,需要开发者自行进行开发接口,这就重新做jni接口.重新做接口就需要我们熟悉platinum的源码了.



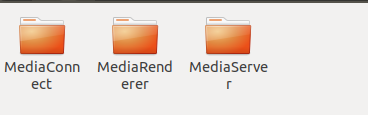
Apps : 这部分也是各自细小模块的测试单元用例.



Core : 这个是核心类



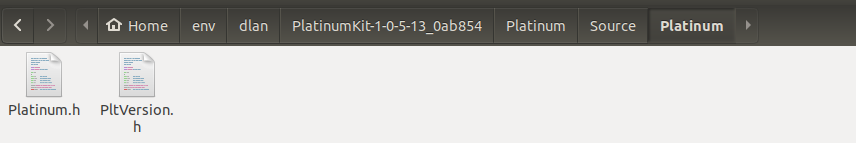
Devices : 相当于核心类的封装.



而这篇文章需要先看熟MediaRenderer文件夹里面的源码,其他的两个后面再看.

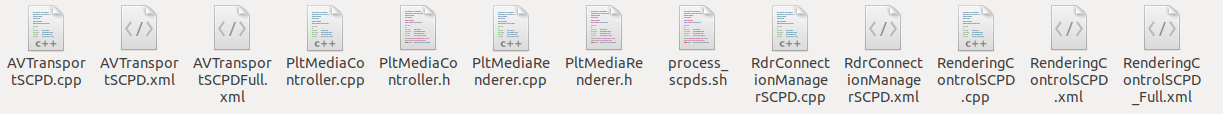
Platform : 这个Android应用dlan的样例,Upnp.java即是从这里来的.

Platinum : 这个文件主要是platinum 静态库的头文件,jni只需要引用这个头文件即可.



其他文件夹都是样例或者测试工具.

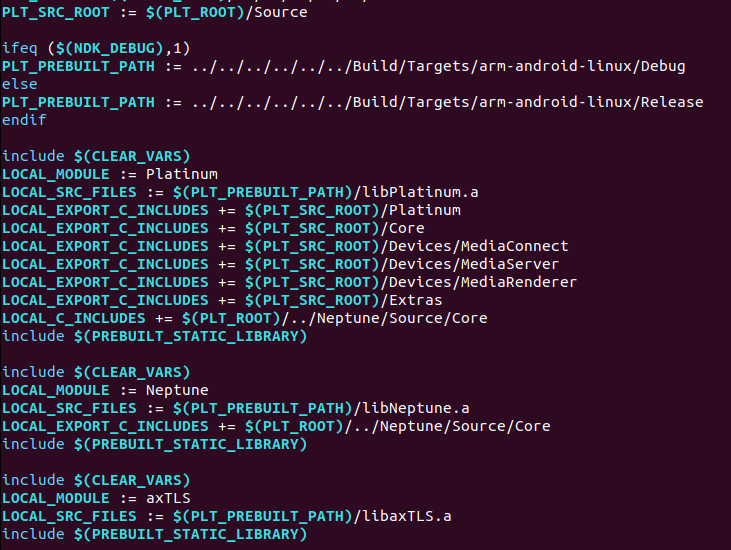
现在先看看MediaRenderer目录下的文档:



文件很少,几个xml是传输协议需要内容.把这个文件里面的源码看一下,基本上还是比较清楚.都是基本的C++封装的,没有特别”混乱的”语法和一堆的指针带指针的.

按照下面的步骤开始:

<0> : 先用Android Studio新建一个AngryPandaDMR工程(https://github.com/MMLoveMeMM/AngryPandaDMR.git),然后创建jni目录,并且配置好NDK编译环境.我们先分析platinum源码中编译so的Android.mk需要哪些目录:

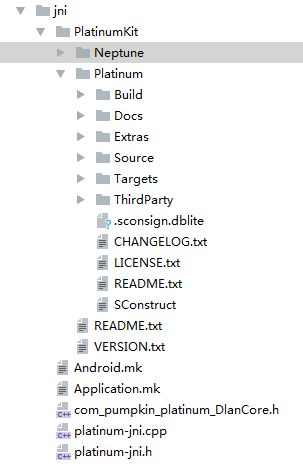


需要几个静态库和刚才熟悉的那个source目录即可.这样好办把需要的全部直接拷贝出来放到jni目录下,重新配置Android.mk文件.

为什么要这么麻烦呢?

这样做是为了后面开发编译调试更方便.

Jni的目录结果先定:



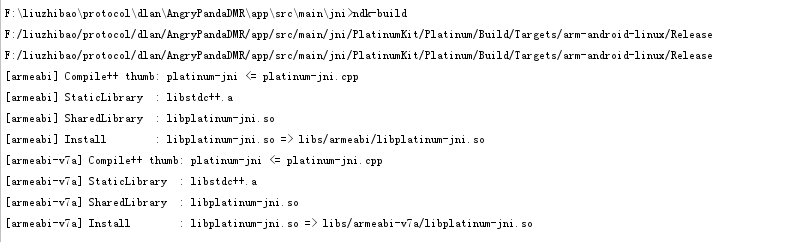
先用原有的代码编译成功再进行修改.

需要注意的是PlatinumKit=源码+编译过的带静态库.a的文件.

即最简单的操作是把Platinum源码按照前面一篇文章完整编译成功,并且可以生产so库,再把他的整个目录拷贝到jni目录下.

然后我们修改Android.mk里面的路径.

修改完以后执行ndk-build开始编译:



<1> : 先增加添加设备的接口,原有的接口就不使用了.

先java层下一个native对应的java类,如下:



添加设备,并且启动UPNP设备

**public static native int** startMediaRender(**byte**[] friendname ,**byte**[] uuid);

停止UPNP设备  
**public static native int** stopMediaRender();

<1> : DMS端发送到DMR的状态变化数据;

<2> : DMR发送数据到DMS端;  
**public static native boolean** respActionEvent(**int** cmd, **byte**[] value ,**byte**[] data);

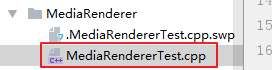
使能打印日志,这个在Android.mk宏(LOCAL\_CFLAGS += -DNPT\_CONFIG\_ENABLE\_LOGGING)中一定定义打印日志,所以这个作用忽略.  
**public static native boolean** enableLogPrint(**boolean** flag);

然后build一下工程,然后使用javah生成对应的.h文件,同时对应写一个.cpp文件



在.cpp文件中需要对应实现上面接口.

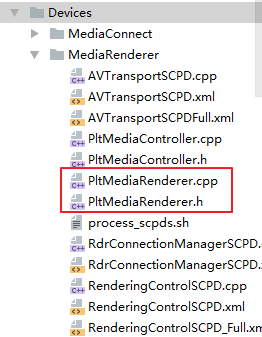
实现之前先看一下:



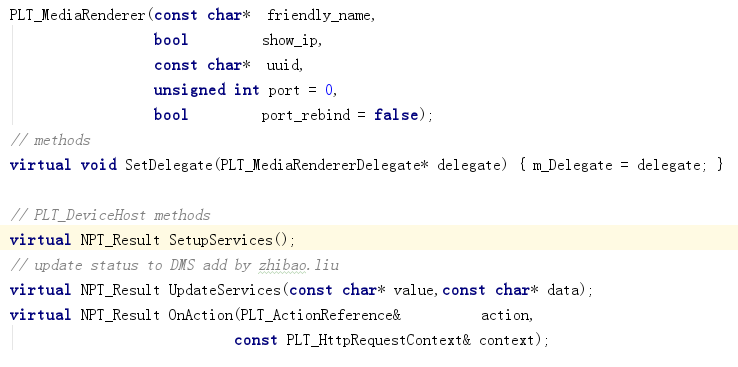
注册设备:

PLT\_DeviceHostReference device(  
 **new** PLT\_MediaRenderer(Options.friendly\_name?Options.friendly\_name:**"Platinum Media Renderer"**,  
 **false**,  
 **"e6572b54-f3c7-2d91-2fb5-b757f2537e21"**));  
upnp.AddDevice(device);

是通过PLT\_MediaRenderer类完成的.再看看PltMediaRenderer.cpp类

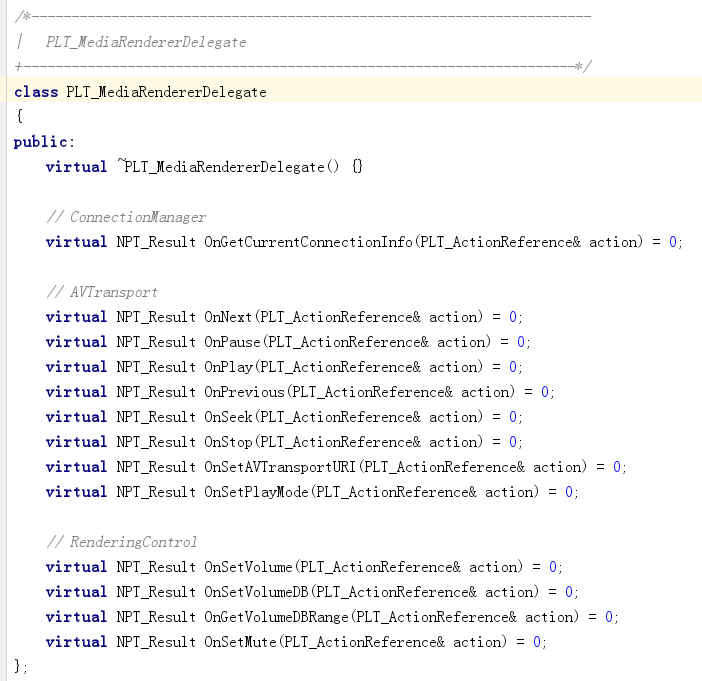


阅读代码:

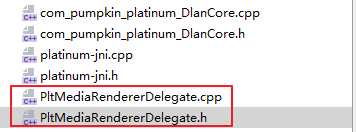


第一个构造方法,即添加设备对象必要的.

SetDelegate(…)这个方法设计是使用代理模式,即MDS那边有任何状态变化,都会触发OnAction方法.但是OnAction源码中并没有继续处理触发事件,而是给了一个代理类.



这个是虚类,具体业务流程需要开发继承这个类自己去完成.然后在jni下面自己写一个继承于这个代理类的具体实现类.



重新梳理一下流程:大致的逻辑关系如下

DMR

PltMediaRenderer

OnAction

OnNext

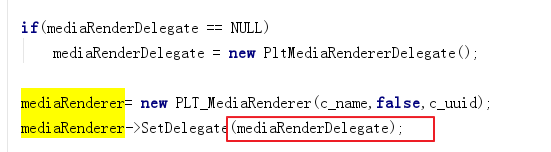
OnPlay

OnStop

代理类

Jni/Java

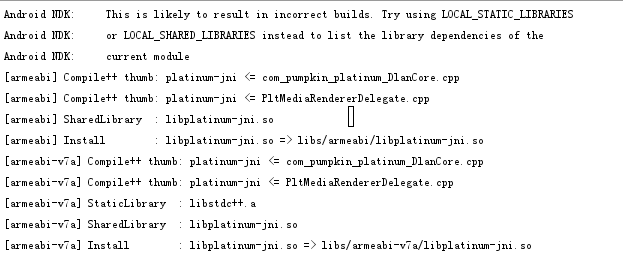
设置代理类如下:



完整代码看项目工程.

然后,修改Android.mk文件路径参数即可.

修改完成上面所有的过程后,执行NDK编译:



全过程和使用只有三个结果.Java层可以建立一个服务,来运作这三个接口和事务处理.

Git源码已经包含所有的代码,可以直接执行编译.

后面继续介绍MDS 适配应用.