Mst6A628 软件架构

时间	版本	内容	作者	备注
2015-7-31	V0.0.1	初版	钟斌	

目录

一、基本知识阐述

1、AIDL

AIDL 是 Android Interface Definition Language 的缩写，即 Android 接口定义语言。

Android 系统中的进程之间不能共享内存，因此，需要提供一些机制在不同进程之间进行数据通信。

为了使其他的应用程序也可以访问本应用程序提供的服务，Android 系统采用了远程过程调用（Remote Procedure Call，RPC）方式来实现。与很多其他的基于 RPC 的解决方案一样，Android 使用一种接口定义语言（Interface Definition Language，IDL）来公开服务的接口。我们知道 4 个 Android 应用程序组件中的 3 个（Activity、BroadcastReceiver 和 ContentProvider）都可以进行跨进程访问，另外一个 Android 应用程序组件 Service 同样可以。因此，可以将这种可以跨进程访问的服务称为 AIDL（Android Interface Definition Language）服务。

更多关于 AIDL 的使用说明可以参考之前的培训文档《Android 的 Service 和 AIDL.doc》。

2、JNI

JNI 是 Java Native Interface 的缩写，它提供了若干的 API 实现了 Java 和其他语言的通信（主要是 C&C++）。从 Java1.1 开始，JNI 标准成为 java 平台的一部分，它允许 Java 代码和其他语言写的代码进行交互。JNI 一开始是为了本地已编译语言，尤其是 C 和 C++ 而设计的，但是它并不妨碍你使用其他编程语言，只要调用约定接受支持就可以了。使用 java 与本地已编译的代码交互，通常会丧失平台可移植性。但是，有些情况下这样做是可以接受的，甚至是必须的。例如，使用一些旧的库，与硬件、操作系统进行交互，或者为了提高程序的性能。JNI 标准至少要保证本地代码能工作在任何 Java 虚拟机环境下。

更多关于 JNI 的使用说明可以参考之前的培训文档《JNI 的使用.ppt》。

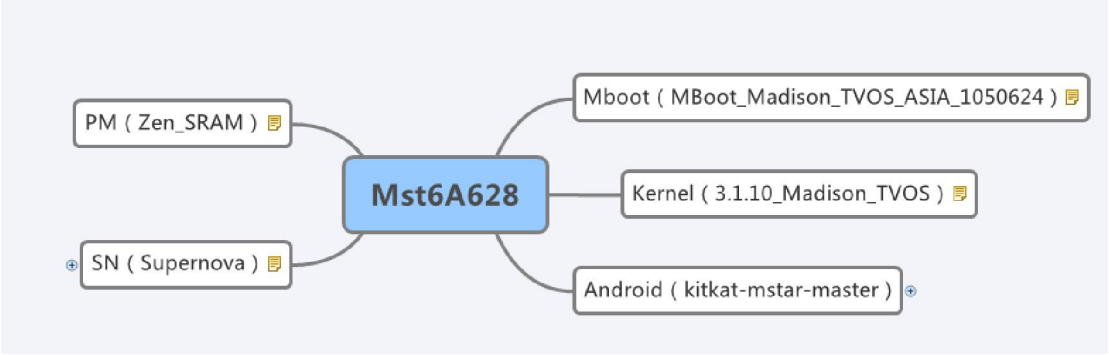
3、Binder 机制

Binder 是一种进程间通信机制，它是一种类似于 COM 和 CORBA 分布式组件架构，通俗一点，其实是提供远程过程调用（RPC）功能。从英文字面上意思看，Binder 具有粘结剂的意思，那么它把什么东西粘结在一起呢？在 Android 系统的 Binder 机制中，由一系统组件组成，分别是 Client、Server、Service Manager 和 Binder 驱动程序，其中 Client、Server 和 Service Manager 运行在用户空间，Binder 驱动程序运行内核空间。Binder 就是一种把这四个组件粘合在一起的粘结剂了，其中，核心组件便是 Binder 驱动程序了，Service Manager 提供了辅助管理的功能，Client 和 Server 正是在 Binder 驱动和 Service Manager 提供的基础设施上，进行 Client-Server 之间的通信。Service Manager 和 Binder 驱动已经在 Android 平台中实现好，开发者只要按照规范实现自己的 Client 和 Server 组件就可以了。

更多关于 binder 的使用说明可以参考老罗的博客：
http://blog.csdn.net/luoshengyang/article/details/6618363

二、模块阐述

1、软件组成



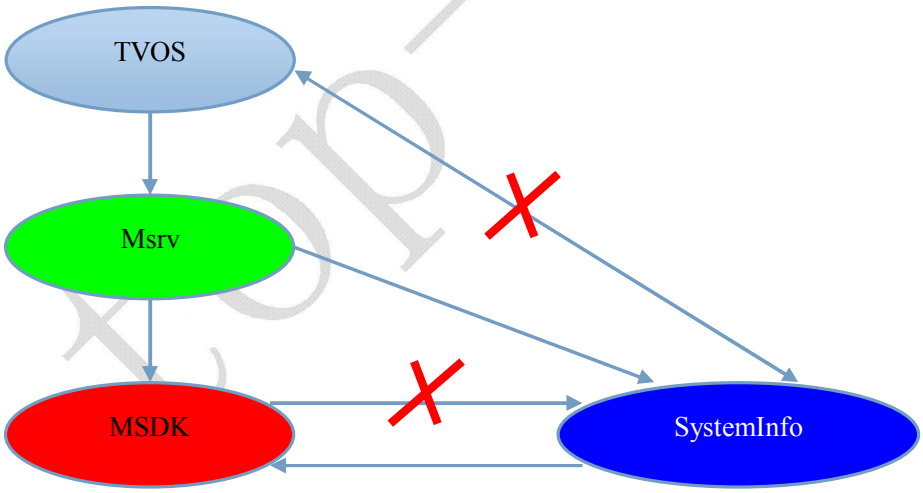
Mboot：初始化一些基本外设，比如说USB、有线网络、Flash等；升级程序、选择启动模式；显示开机logo、播放开机音乐等。
PM：在机子处于待机状态下运行的一个小程序；实时检测外部中断，比如按键板、遥控器等。
Kernel：初始化一些基本外设，比如说USB、有线网络、Flash、遥控器等。
Android：与标准的android系统区别不大。
Supernova：它相当于Android系统中的一个很大的本地Service，把电视相关的所有功能都集中在这里实现，它是由android的init.madison.rc启动起来的。

2、Mstar Java API

主要是在\kitkat-mstar-master\device\mstar\common\libraries\tvapi 文件夹下，它只是起到一个桥梁的作用，用于 Java Service 与 Native Service 之间的通讯。

3、Mstar SN 简介

1)、各模块的调用关系



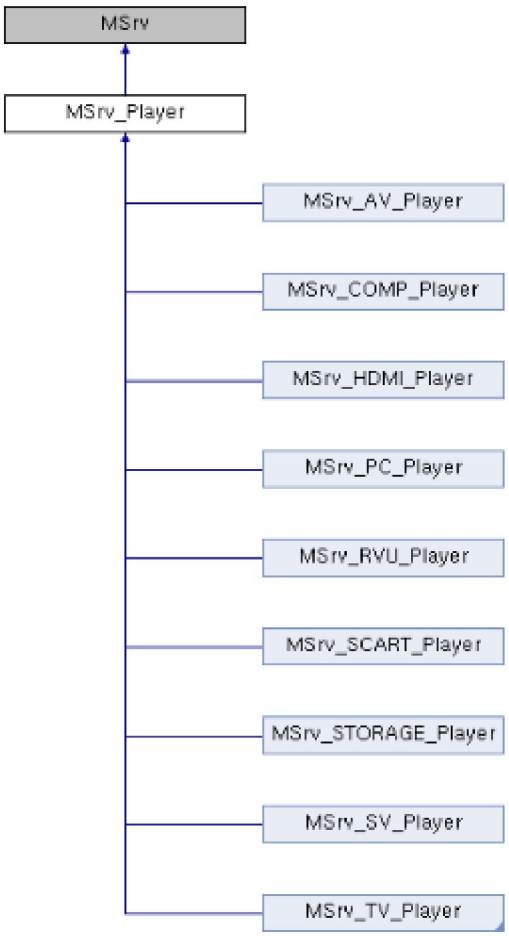
MSDK 调用 SystemInfo => 不支持
SystemInfo 调用 MSDK => 支持
Msrv 调用 SystemInfo => 支持
TVOS 调用 Msrv => 支持

2)、各模块的功能介绍

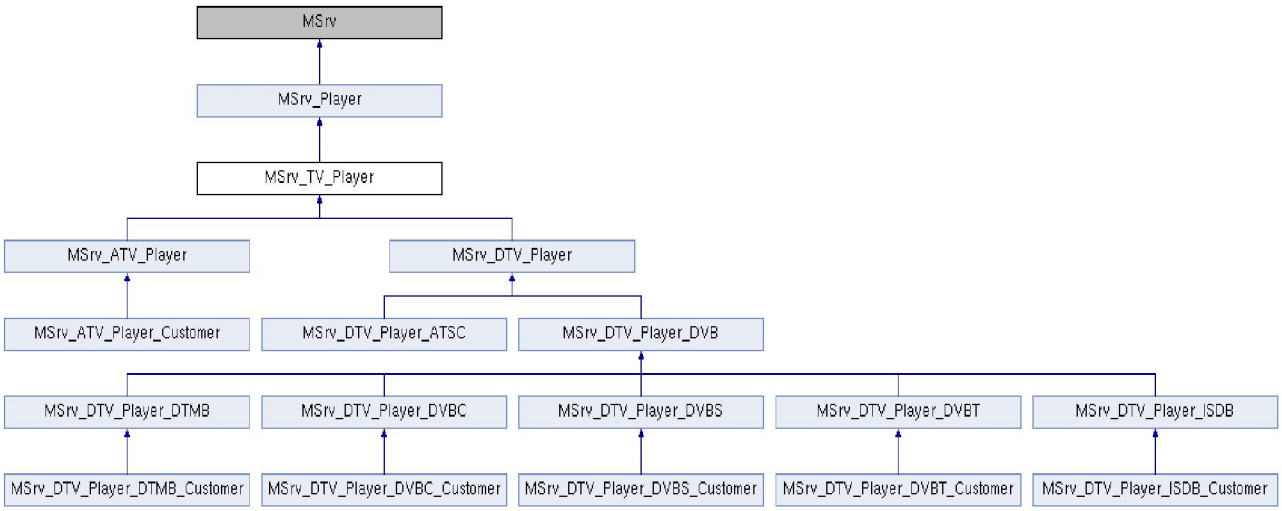
A：Supernova/ MStarSDK：这部分就是 mstar TV API，也就是以 mapi_开头的文件都是在这里面，不过已经打包了。

B：Supernova/ projects/tvos：这个是与 android 通讯的 binder 机制中服务端代码。
C：Supernova/ projects/systeminfo：这个解析所有的 ini 文件及 board 文件，把解析出来的数据放到 mapi 层方便调用，或存在变量里方便 msrv 层函数调用。
D：Supernova/ projects/devices：一些芯片的驱动，比如 tuner、demodulator 等
E：Supernova/ projects/msrv：一些功能的处理处理逻辑及调用 mapi 层功能等。
F：Supernova/ projects/board：板型定义、存放所有的 ini 文件，数据库，开机音乐，开机 logo 等。

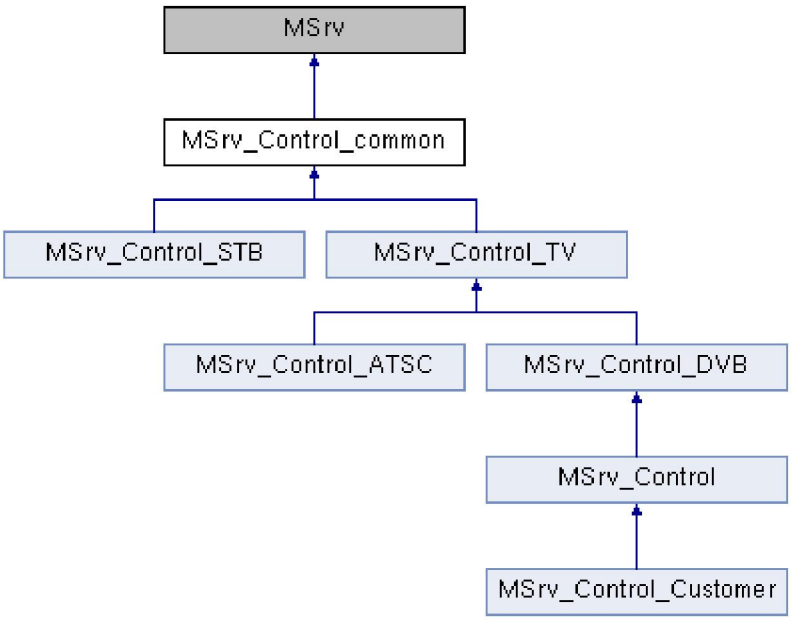
3)、重要类的继承关系图
A：MSrv_Player 的子类图



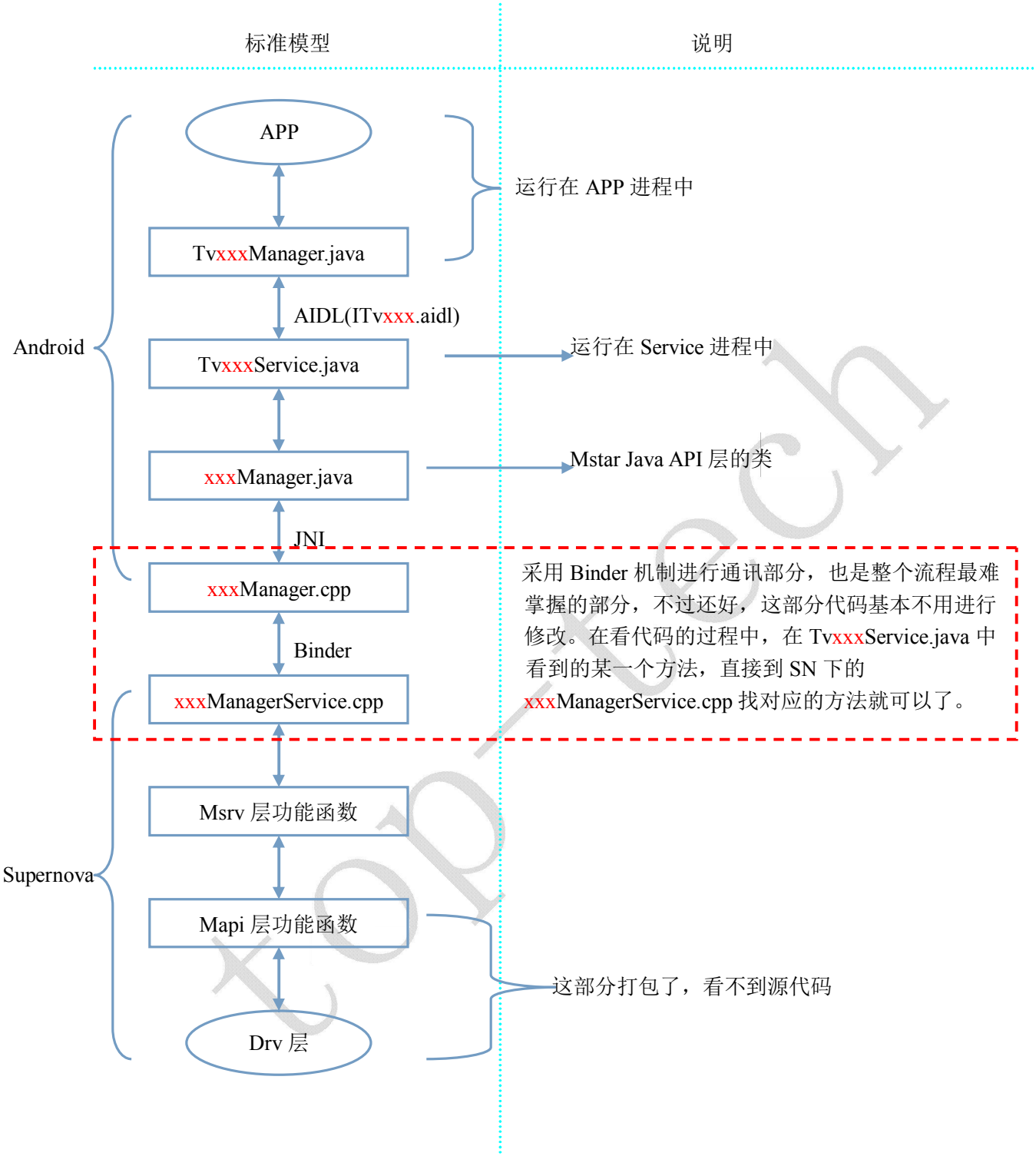
B：MSrv_TV_Player 子类图



C：MSrv_Control_common 子类图



三、APP 与 SN 通讯流程



四、举例

以 MTvPlayer 应用中的 Picture Mode 为例，简要阐述一下 APP 是怎么调用到 SN 层的。从 android 启动的流程可以知道，androd 启动的第一个应用就是 Launcher，而从 Activity 的生命周期可以知道，Activity 在跑完 OnResume 方法时，就表明这个 Activity 已经启动完成了。通过查看我们目前的 Launcher 入口 LauncherActivity 的 OnResume 的代码可知，它直接启动了 MTvPlayer 应用的 RootActivity。所以在系统启动完成后，不进行任何操作时，我们

的系统目前最顶层 Activity 是 RootActivity，此时当我们去操作按键时，它就会去执行 RootActivity 中的 OnKeyDown 方法，在 OnKeyDown 方法中，会根据不同的按键值去执行不同的功能。当我们按 Menu 键时，它就会启动 MainMenuActivity，也就是我们看到 TV OSD 了。根据 MainMenuActivity 的启动流程，我们可以找到在 PictureViewHolder 中与 Picture Mode 相关的部分如下：

```
public void findViews() {
……
comboBtnPictureMode = new ComboButton(activity, activity.getResources().getStringArray(
                    R.array.str_arr_pic_picturemode_vals), R.id.linearlayout_pic_picturemode, 1, 2,
                    true) {
                @Override
                public void doUpdate() {
                    int specifyPicMode = TvPictureManager.PICTURE_MODE_NORMAL;
                    SetFocusableForUserMode();
                    if (TvPictureManager.getInstance() != null) {
                        specifyPicMode = comboBtnPictureMode.getIdx();
                        TvPictureManager.getInstance().setPictureMode(comboBtnPictureMode.getIdx());
                        …….
                    }
                    freshDataToUIWhenPicModChange();
                }
            };
}
```

下面就以 ComboButton()方法中的回调方法 doUpdate()作为我们的分析入口，在 doUpdate()中调用了 TvPictureManager.getInstance().setPictureMode()方法，这个方法是怎么将数据传递到 SN 的？这就是我们下面要分析的重点。

下面从四个部分阐述整个流程，分别是：

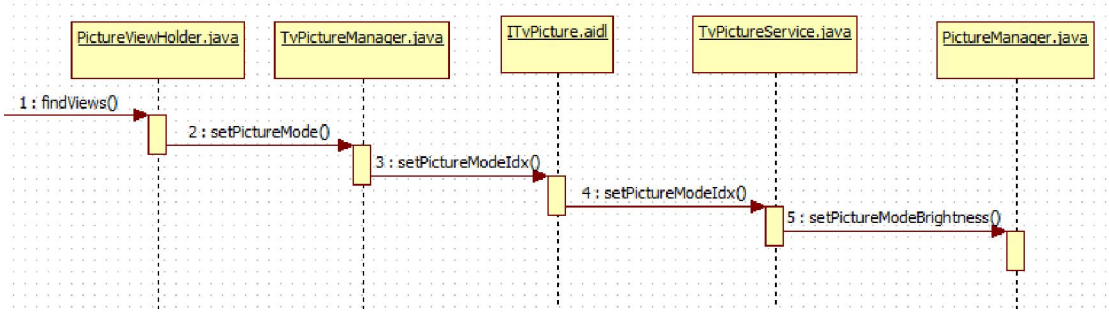
一、利用 AIDL 技术，如何从当前的应用进程的数据传递到 Service 进程？

二、利用 JNI 技术，如何将数据从 Java 服务框架传递到 CPP 本地服务框架？

三、利用 Binder 机制，如何将数据从客户端（Android）传递到服务端（Supernova）？

四、Supernova 层是如何将用户数据写到芯片存寄器的？

1、如何从当前的应用进程的数据传递到 Service 进程



在系统启动的时候，会启动一个 Service 进程，把所有的 Service 都启动起来，并且一直在后台运行，而在应用启动的时候，也会在虚拟机中开一个独立的进程，但是这两个进程之间不能直接进行数据交换，所以这里就采用一些跨进程通讯机制，比如广播、Intent、AIDL、

Binder 等。

Step1:

\kitkat-mstar-master\device\mstar\common\apps\MTvPlayer\src\com\mstar\tv\tvplayer\ui\holder\PictureViewHolder.java

```
public void findViews() {
……
comboBtnPictureMode = new ComboButton(activity, activity.getResources().getStringArray(
                    R.array.str_arr_pic_picturemode_vals), R.id.linearlayout_pic_picturemode, 1, 2,
                    true) {
                @Override
                public void doUpdate() {
                    int specifyPicMode = TvPictureManager.PICTURE_MODE_NORMAL;
                    SetFocusableForUserMode();
                    if (TvPictureManager.getInstance() != null) {
                        specifyPicMode = comboBtnPictureMode.getIdx();
                        TvPictureManager.getInstance().setPictureMode(comboBtnPictureMode.getIdx());
                        …….
                    }
                    freshDataToUIWhenPicModChange();
                }
            };
 ……
}
```

在初始化 picture mode 选项时，是采用 ComboButton 这个自定义方法，当用户在操作 Picture Mode 项，且发生数据变化时，就会回调 ComboButton 这个自定义方法里的 doUpdate()方法以便更新数据。这里的 TvPictureManager 是采用单例设计模式，通过 TvPictureManager.getInstance()就获得了 TvPictureManager 的对象。

Step2:

\kitkat-mstar-master\device\mstar\common\libraries\tv2\java\com\mstar\android\tv\TvPictureManager.java

```
    public boolean setPictureMode(int pictureMode) {
        ITvPicture service = getService();
        try {
            return service.setPictureModeIdx(pictureMode);
        } catch (RemoteException e) {
            e.printStackTrace();
        }
        return false;
    }
```

这里通过 getService()获取到了 ITvPicture 对象，getService()的代码如下：

```
private static ITvPicture getService() {
        if (mService != null) {
            return mService;
```

```
        }
        mService = TvManager.getInstance().getTvPicture();
        return mService;
    }
```

getTvPicture()的代码如下：

```
public ITvPicture getTvPicture() {
        try {
            return mService.getTvPicture();
        } catch (RemoteException e) {
            e.printStackTrace();
            return null;
        }
    }
```

这里的 mService 在调用 TvManager.getInstance()时初始化的，是 TvService 的一个实例。所以在调用 mService.getTvPicture()时，实际获取到的是 TvPictureService 对象。

Step3:
\kitkat-mstar-master\device\mstar\common\libraries\tv2\java\com\mstar\android\tv\ITvPicture.aidl

```
interface ITvPicture {
    boolean setPictureModeIdx(in int ePicMode);
    ……
    }
```

AIDL 文件的定义是采用接口形式的，在编译的时候会生成相应的 Java 文件，比如 ITvPicture.aidl 文件在编译的时候就生成了对应的 ITvPicture.java 文件，在 ITvPicture.java 里的文件内容不能进行修改，在 ITvPicture.java 里会生一个名为 Stub 的抽象类，这个类被对应的 Service 继承。

Step4:
\kitkat-mstar-master\device\mstar\common\libraries\tv2\MTvService\src\com\mstar\tv\service\TvPictureService.java

```
@Override
    public boolean setPictureModeIdx(int ePicMode) throws RemoteException {

        PictureModeSetting picturMode = DatabaseDesk.getInstance(mContext)
                .queryPictureModeSettings(ePicMode, getCurrentInputSource());
        DatabaseDesk.getInstance(mContext).updatePictureMode(ePicMode,
getCurrentInputSource());
        try {
            TvManager.getInstance().getPictureManager()
                    .setPictureModeBrightness(picturMode.brightness);
            ……
        } catch (TvCommonException e) {
            e.printStackTrace();
```

```
        }
        return true;
    }
```

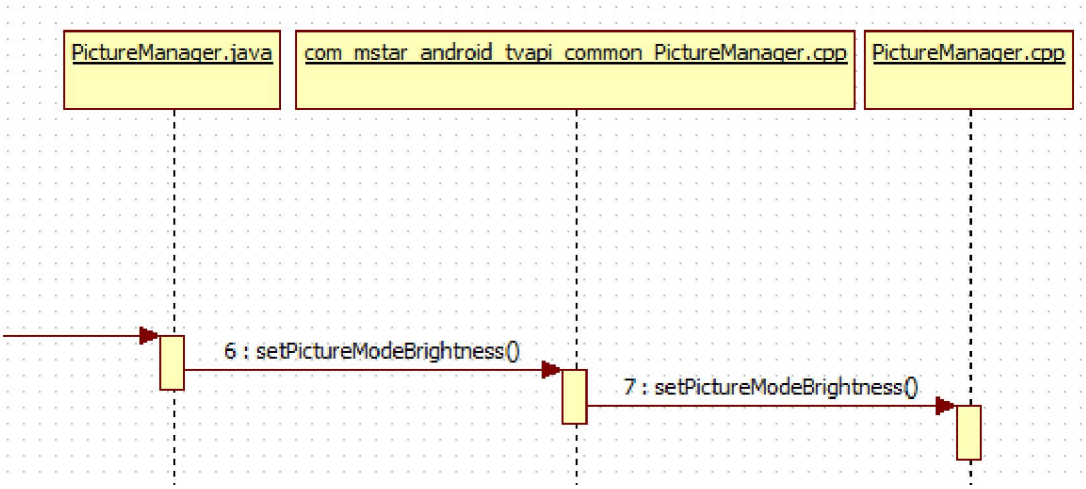
TvPictureService 类是 ITvPicture.Stub 的实现类。在 setPictureModeIdx()方法中主要做了三件事情：一、读出当前图像模式的值；二、更新数据到数据库；三、传递值到 SN 层。Picture Mode 分为很多种，比如标准、生动、用户等，而每一种模式又包括了亮度、对比度、色调、饱和度和清晰度五种。所以在调节图像模式时调节了一系列的值。这里就单独以调节亮度为例。

Step5:
\kitkat-mstar-master\device\mstar\common\libraries\tvapi\java\com\mstar\android\tvapi\common\PictureManager.java

```
    static {
        try {
            System.loadLibrary("picturemanager_jni");
            native_init();
        } catch (UnsatisfiedLinkError e) {
            System.err.println("Cannot load picturemanager_jni library:\n" + e.toString());
        }
    }
    ……
public native final void setPictureModeBrightness(short value) throws TvCommonException;
```

这里的 setPictureModeBrightness()方法并没有看到任何实现，像是只在这里进行了声明，但是在这个方法声明时多加了一个关键字 native，有这个关键字的方法声明表示声明了一个本地方法，也就是说这个将会去调用对应的 JNI 里面的方法。但是要使用 JNI 时，就要在类的前面以静态形式把对应的 JNI 库加载进来，即 System.loadLibrary("picturemanager_jni")。

2、如何将数据从 Java 服务框架传递到 CPP 本地服务框架



Step6:
\kitkat-mstar-master\device\mstar\common\libraries\tvapi\jni\com_mstar_android_tvapi_common_PictureManager.cpp

```
    static JNINativeMethod methods[] = {
```

```
……
{"setPictureModeBrightness",                                    "(S)V",          (void
*)com_mstar_android_tvapi_common_PictureManager_setPictureModeBrightness},
……
}
```

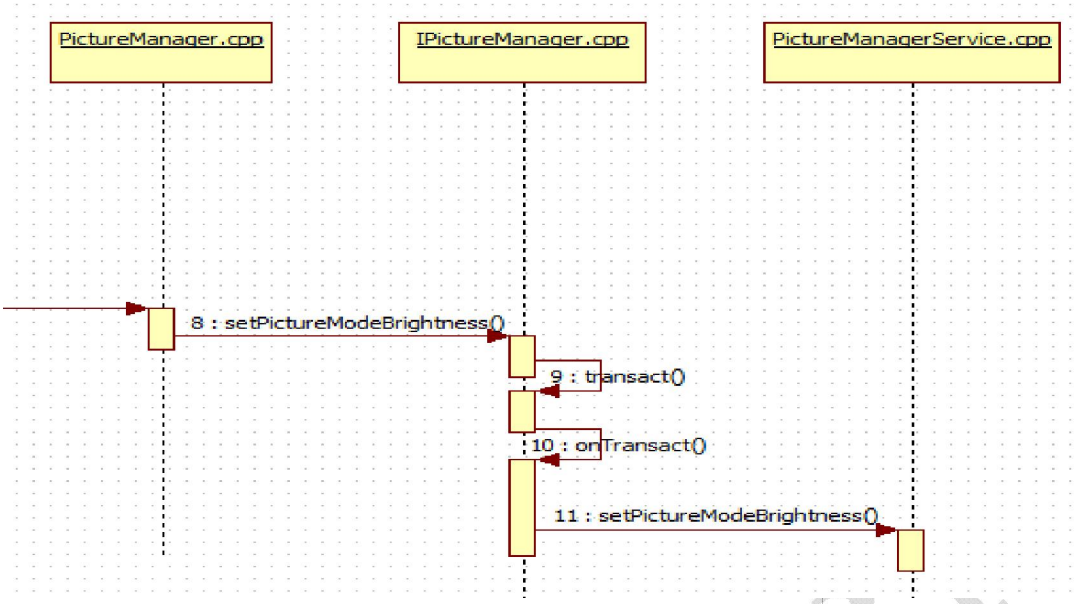
在系统启动的时候，会根据 JNI 的入口 JNI_OnLoad()方法，把 methods[]注册到系统中。这里说明一下，怎么样以最快的速度通过 java 文件去找到对应的 jni 文件？有个规则就是 jni 文件的命名都是以 java 文件的包名和类名组成的，把包名中的“.”替换成“_”即可。比如说 PictureManager 类的包名是 com.mstar.android.tvapi.common，类名是 PictureManager，又 jni 文件都是用 C++语言来写的，所以 PictureManager 类对应用的 JNI 文件名就是 com_mstar_android_tvapi_common_PictureManager.cpp 。在 PictureManager 类中调用 setPictureModeBrightness()方法时，就会去查找 methods[]这个表，查找到对应的字符串，然后会去执行相应的方法 com_mstar_android_tvapi_common_PictureManager_setPictureModeBrightness()。

Step7:
\kitkat-mstar-master\device\mstar\common\libraries\tvapi\jni\com_mstar_android_tvapi_common_PictureManager.cpp

```
void com_mstar_android_tvapi_common_PictureManager_setPictureModeBrightness
(JNIEnv *env, jobject thiz, jshort brighntum) {
    ALOGI("setPictureModeBrightness");
    sp<PictureManager> ms = getPictureManager(env, thiz);
    …….
    ms->setPictureModeBrightness(brighntum);
    return;
}
```

sp<xxx>表示定义一个强指针，是 Android 中智能指针的一种。智能指针是 C++中的一个概念，通过基于引用计数的方法，解决对象的自动释放的问题。在 C++编程中，有两个很让人头痛的问题：一是忘记释放动态申请的对象从而造成内存泄露；二是对象在一个地方释放后，又在别的地方被使用，从而引起内存访问错误。程序员往往需要花费很大精力进行精心设计，以避免这些问题的出现。在使用智能指针后，动态申请的内存将会被自动释放（有点类似 Java 的垃圾回收），不需要再使用 delete 来释放对象，也不需要考虑一个对象是否已经在其它地方被释放了，从而使程序编写工作减轻不少，而程序的稳定性大大提高。所以可以把 sp<xxx>当成指针来看。这里通过 getPictureManager()去新建了一个 PictureManager 的对象，并且 ms 指向了这个对象。

3、如何将数据从客户端（Android）传递到服务端（Supernova）



这部分的内容主要是采用 Binder 机制进行通讯的。需要一定的 binder 机制基础知识，才能更好地去阅读代码。

Step8:

\kitkat-mstar-master\vendor\mstar\supernova\projects\tvos\picturemanager\libpicturemanager\PictureManager.cpp

```
void PictureManager::setPictureModeBrightness(int16_t value)
{
    if(mPictureManager == NULL)
        return ;
    return mPictureManager->setPictureModeBrightness(value);
}
```

在 PictureManager.java 的构造方法中，调用一个本地方法 native_setup()，而这个方法最终会通过 JNI 调到 PictureManager.cpp 中的 connect()方法。并且把 PictureManager 通过 connect()方法传递给 PictureManagerService 类的 connect()方法中，返回的是一个 Client 对象，但类型被转为 sp<IPictureManager>。而 Client 类是 BnPictureManager 类的子类，BnPictureManager 类是继承 BnInterface<IPictureManager> 类，BnPictureManager 类是 binder 机制中客户端的本地代理类，BpPictureManager 类是 binder 机制中客户端的远程代理类，它继承了 BpInterface<IPictureManager> 类，而接口类 IPictureManager 中的所有方法都是在类 BpPictureManager 中实现，所以 mPictureManager 指向的是 IPictureManager 类。所以在通过 mPictureManager 去调用 IPictureManager 类的方法时，会去调用 BpPictureManager 类中的方法。

Step9:

\kitkat-mstar-master\vendor\mstar\supernova\projects\tvos\picturemanager\libpicturemanager\IPictureManager.cpp

```
void BpPictureManager::setPictureModeBrightness(int16_t value)
{
    ALOGV("Send PICTURE_SetPictureModeBrightness1\n");
```

```
    Parcel data, reply;
    data.writeInterfaceToken(IPictureManager::getInterfaceDescriptor());
    data.writeInt32(value);
    remote()->transact(PICTURE_SetPictureModeBrightness1, data, &reply);
    return ;
}
```

remote()是客户端的本地代理的对象，所以通过 remote()->transact()方法可以调用到 BnPictureManager 类中的 onTransact()方法。

Step10:
\kitkat-mstar-master\vendor\mstar\supernova\projects\tvos\picturemanager\libpicturemanager\IPictureManager.cpp

```
status_t BnPictureManager::onTransact(uint32_t code,
                                      const Parcel& data,
                                      Parcel* reply,
                                      uint32_t flags)
{
    switch(code)
    {
    ……
    case PICTURE_SetPictureModeBrightness1:
                {
                  ALOGV("Receive PICTURE_SetPictureModeBrightness1\n");
                CHECK_INTERFACE(IPictureManager, data, reply);

                int32_t value =    data.readInt32();
                setPictureModeBrightness(value);
                return NO_ERROR;
                }break;
    ……
    }
}
```

PictureManagerService 类里的一个内部类 Client 是继承了 BnPictureManager 类的，所在这里的 setPictureModeBrightness()会调用 PictureManagerService 类的内部类 Client 类里的方法。

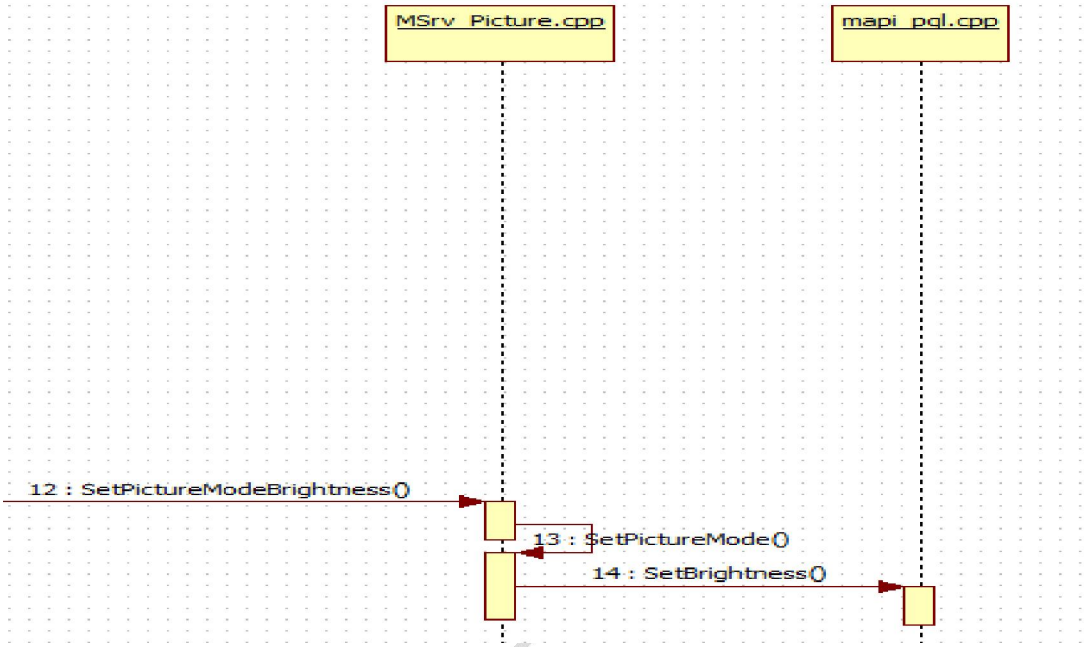
Step11:
\Supernova\projects\tvos\picturemanager\libpicturemanagerservice\ PictureManagerService.cpp

```
void PictureManagerService::Client::setPictureModeBrightness(int16_t value)
{
    Mutex::Autolock lock(m_FuncLock);
    #if(1==RELEASE_BINDER_TEST)
    TEST_SETPICTUREMODEBRIGHTNESS()
```

```
    #endif
    ALOGV("PictureManagerService::Client::setPictureModeBrightness\n");
    MSrv_Control::GetMSrvPicture()->SetPictureModeBrightness(value);
}
```

通过 MSrv_Control::GetMSrvPicture()获取到了 MSrv_Picture 一个对象。

4、Supernova 层是如何将用户数据写到芯片存寄器的



Step12:

\Supernova\projects\msrv\common\src\ MSrv_Picture.cpp

```
void MSrv_Picture::SetPictureModeBrightness(U8 u8Value)
{
    MAPI_INPUT_SOURCE_TYPE CurrentMapiInputType;
#if(PIP_ENABLE == 1)
……
#else
    CurrentMapiInputType = MSrv_Control::GetInstance()->GetCurrentInputSource();
#endif
    SetPictureMode(CurrentMapiInputType, PICTURE_ITEM_BRIGHTNESS, (U8)u8Value);

}
```

Step13:

\Supernova\projects\msrv\common\src\ MSrv_Picture.cpp

```
void MSrv_Picture::SetPictureMode(MAPI_INPUT_SOURCE_TYPE enInputSrc,
PICTURE_ITEM enItem, U8 u8Value)
{
    U8 u8RelValue;
    mapi_pql *pPql;
```

```
    pPql = mapi_interface::Get_mapi_pql(mapi_pql::GetWinType());

    switch(enItem)
    {
        case PICTURE_ITEM_BRIGHTNESS:
        {
            U8 u8Brightness, u8SubBrightness;
            T_MS_VIDEO stVideoTemp;

            u8Brightness = u8Value;
            MSrv_Control::GetMSrvSystemDatabase()->GetVideoSetting(&stVideoTemp,
&enInputSrc);
            u8SubBrightness = stVideoTemp.g_astSubColor.u8SubBrightness;

            pPql->SetBrightness(

pPql->Transfer_Bri(FactoryAdjBrightness(BrightnessN100toReallyValue(enInputSrc,
u8Brightness), u8SubBrightness),    mapi_pql_datatype::E_MAPI_PQL_BRI_TRANS_RED),

pPql->Transfer_Bri(FactoryAdjBrightness(BrightnessN100toReallyValue(enInputSrc,
u8Brightness), u8SubBrightness),    mapi_pql_datatype::E_MAPI_PQL_BRI_TRANS_GREEN),

pPql->Transfer_Bri(FactoryAdjBrightness(BrightnessN100toReallyValue(enInputSrc,
u8Brightness), u8SubBrightness),    mapi_pql_datatype::E_MAPI_PQL_BRI_TRANS_BLUE));
        }
        break;
……
    }
}
```

总结：整个流程涉及的语言有 Java 和 C++，涉及的跨进行通讯机制有 AIDL、JNI、Binder，也运用到了面向对象的多态、继承、单例设计模式等，所以要能很好地理解并熟练地运用整个流程，还得很好地把基础知识学牢固。