Android 解决方案

开发指南

文档版本　10

发布日期　2015-08-20

商标声明

 、HISILICON、海思和其他海思商标均为深圳市海思半导体有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受海思公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

深圳市海思半导体有限公司

地址：　深圳市龙岗区坂田华为基地华为总部　邮编：518129

网址：　http://www.hisilicon.com

客户服务邮箱：　support@hisilicon.com

前 言

概述

本文档主要介绍 Android 解决方案的功能、常用的接口和内部工作原理，通过实例介绍各模块的开发过程以及注意事项。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
Hi3716C	V2XX
Hi3716M	V4XX
Hi3719C	V1XX
Hi3719M	V1XX
Hi3718C	V1XX
Hi3718M	V1XX
Hi3798C	V1XX
Hi3796C	V1XX
Hi3798M	V1XX
Hi3796M	V1XX
HiSTBAndroid	V500R001
HiSTBAndroid	V600R001

读者对象

本文档（本指南）主要适用于以下工程师：

- 技术支持工程师
- 软件开发工程师

作者信息

章节号	章节名称	作者信息
1	概述	C00210106
2	环境配置	G00180855
3	内存配置	G00180855
4	HiMediaPlayer	T00194509/Y00303705
5	HiDLNA	D47906/Z00222835/Y00227502
6	HiMultiScreen	D47906/Z00222835
7	HiTranscoder	D47906/Z00222835
8	HiMiracast	W00269687
9	HiKaraoke	Y00304459

修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

修订日期	版本	修订说明
2015-08-20	10	新增 5.4.1.10 章节。
2015-06-25	09	删除 Hi3798CV200 和 Android5.X 信息；更新 2.3 章节；新增 5.4.1.2~5.4.1.9 章节。
2015-04-01	08	修改 2.2 开发环境搭建 和 2.3 开发编译，增加 Android5.0 环境搭建和编译。新增 8.4 章节和对 Hi3798CV200 的支持。
2015-03-24	07	新增第 9 章。
2014-10-31	06	新增 4.4.2、4.4.3 章节；修改 7.4.2.2 的步骤 8；修改 2.3 章节的各个目录；修改 3.2 章节
2014-09-18	05	修改 Multiscreen；新增 HiMiracast。
2014-07-15	04	新增 Hi379X 系列芯片内容。
2014-04-04	03	新增 6.4.4.3 章节。

修订日期	版本	修订说明
2014-03-13	02	修改第 5 章 HiDLNA 和第 7 章 HiTranscode。
2013-12-13	01	新增 5.4.1.1 和 7.4.2 章节；修改文档兼容 Android4.4 和 Hi3716MV400 芯片。
2013-07-31	00B01	第 1 次临时发布。

目 录

插图目录

表格目录

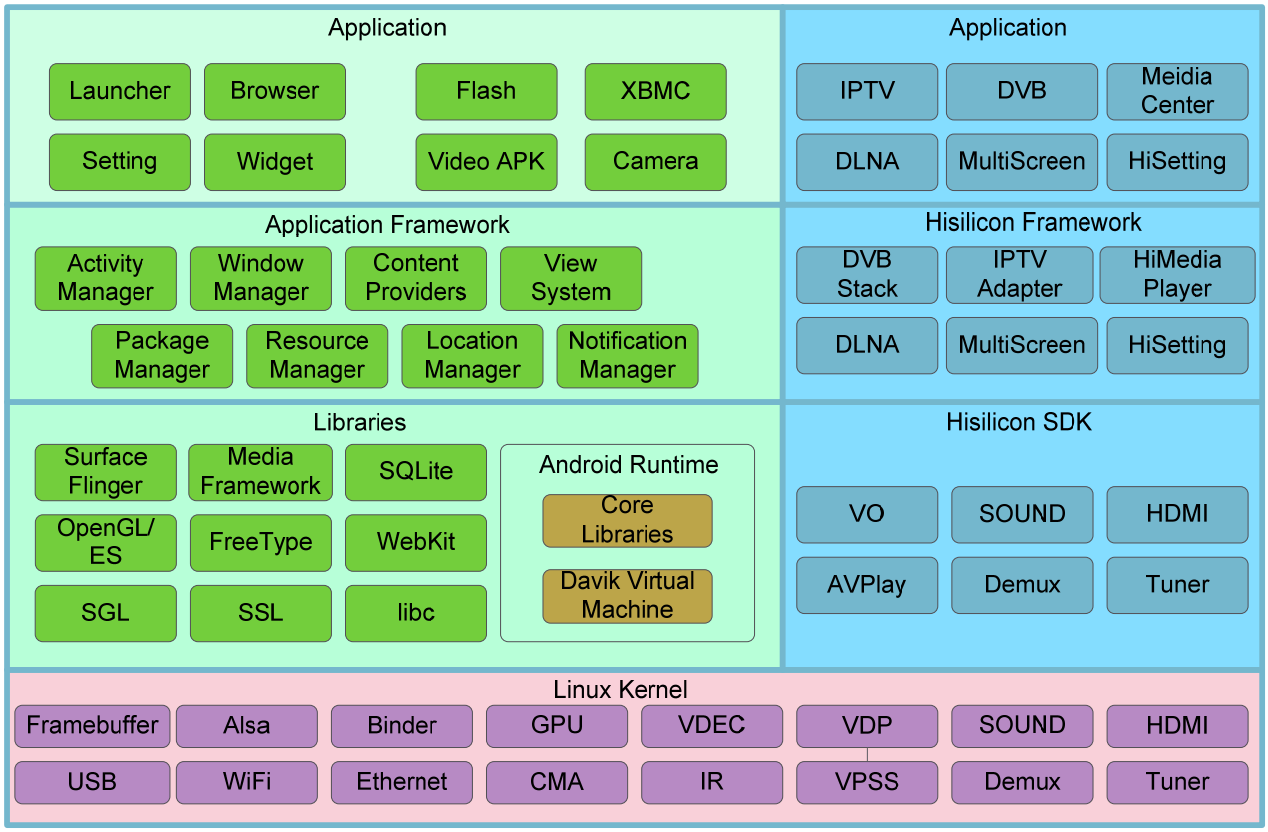
1 概 述

1.1 Android 解决方案整体架构

海思 Android 解决方案整体架构上，可以分为两部分：Android 基础架构、海思扩展架构。

Android 基础架构对 Android 原生设计和接口尽量保持不变，海思只进行少量的功能增加和修改。海思扩展架构提供了海思特有应用及接口，基于海思 SDK，一部分功能以 UNF 接口提供，一部分功能提供 JNI 层和 Java 层的接口封装。架构如图 1-1 所示。

图1-1 Android 解决方案整体架构



海思解决方案，尽量少的改动 Android 原生架构，并把扩展的功能和接口与 Android 原生架构尽量解耦，主要考虑到 Android 版本的快速升级和用户对扩展接口的稳定的需求，同时提高 Android 的应用兼容性。用户定制开发的功能如果基于海思扩展架构，那么 Android 版本升级时，由于 Android 基础架构改动较少，海思可以支持用户快速的升级版本，并且扩展架构中海思提供的接口变化会很小，用户的定制开发功能就能最大程度继承下来。

Android 基础架构主要包含以下几层：

- Application 层

  Android 原生应用层，支持 Flash，浏览器（HTML5 视频），视频客户端，XBMC 等开放市场应用。

- Framework 层

  Android 原生框架层，海思会进行少量的修改，以支持一些扩展功能，比如有线网络，PPPOE 等。

- Libraries 层

  Android 原生 Lib 层。除了基本的原生库支持，在 Kernel 之上，海思还会进行硬件抽象层（HAL）的适配实现，包括 OpenMAX，Alsa，GPU，Camera 等。

海思扩展架构主要分为以下几层：

- Application 层

  海思开发的文件浏览器，媒体播放器，HiSetting 等参考应用实现。用户开发的 IPTV，DVB，多屏等应用可以在此层实现。

- Framework 层

  海思开发的 HiMediaPlayer，DLNA，多屏，DisplaySetting，AudioSetting 等扩展框架。用户可以基于 Hisilicon SDK 在此层实现自己的扩展框架。

- SDK

  海思 UNF 接口实现层，封装驱动及硬件实现。

1.2 海思修改和新增功能

1.2.1 Android 基础架构

海思修改和新增功能：

- 修改适配：
  - 媒体框架扩展：涉及模块 Media Framework，View System
  - 设备挂载卸载：涉及模块 VOLD，MountService
  - 音量调节：涉及模块 Audio Framework
  - 支持 U 盘升级：涉及模块 Recovery
  - HAL 适配：涉及模块 OpenMAX，GPU，Alsa，Camera，WiFi Driver，PowerManager，etc
- 新增：

- 有线网络：涉及模块 Ethernet，Setting
- PPPOE：涉及模块 PPPOE，Setting
- 文件系统扩展(UBI,NTFS)：涉及模块 VOLD，Recovery

1.2.2 海思扩展架构

- 海思提供的 Java 层接口：
  - HiMediaPlayer
  - HiDLNA
  - HiMultiScreen
- 海思提供的 C/C++层接口：
  - HiTranscoder
  - Hisilicon SDK UNF 接口（参考 HMS 开发指南）
- 新增功能
  - 显示设置：涉及模块 显示分辨率，显示区域调整，图像设置
  - 音频设置：涉及模块 HDMI 输出，SPIDF 输出，蓝光降级输出
  - 视频设置：涉及模块 视频宽高比
  - 多屏互动：涉及模块 HiDLNA，HiMultiScreen，HiTranscoder，Miracast
  - 视频播放：涉及模块 HiMediaPlayer
  - 应用：涉及模块 媒体中心，媒体（蓝光 3D）播放器，NFS，Samba，HiAnimation 等

2 开发环境配置

本章主要介绍 Linux 服务器端 Android 开发环境搭建、编译和镜像烧写方法，在 PC 端搭建 Android 应用开发环境，请参考 Google 开发者网站的说明，网址如下：

http://developer.android.com/sdk/index.html

说明

本章节以 Hi3798MV100 为例，其他芯片类似。

2.1 Android 开发环境要求

- 操作系统：64 位 Ubuntu 10.04
- 硬盘空间：最小 100GB
- Python 版本：2.4～2.7
- JDK 版本 1.6
- 交叉编译工具链：arm-hisiv200-linux

注意

使用其他 Ubuntu 版本，可能会有兼容性问题，Ubuntu10.04 版本为经过验证最稳定开发 Android 的操作系统版本。

从 Android (2.3.x) Gingerbread 版本开始，编译环境都要求为 64 位操作系统。

2.2 开发环境搭建

2.2.1 自动化配置

在“发布包/Software/ServerInstall”目录中，我们提供了安装系统环境和交叉编译工具的自动化脚本：ServerInstall.sh，需要用 root 用户，在脚本所在目录执行此脚本来进行自动化配置：

```
Software/ServerInstall$sudo ./ServerInstall.sh
```

注意

- **执行此脚本之前请将服务器接入互联网。**
- **请在脚本所在目录执行。**
- **脚本需要 root 权限才可执行。**
- **脚本执行过程中需要与用户交互。**
- **如果脚本执行失败，可以根据提示解决错误后再重新执行脚本，或者按照本章余下内容手动配置开发环境。**

2.2.2 手动安装

步骤 1 添加更新源。

确保能连接上互联网，添加安装源。

```
$ sudo add-apt-repository "deb http://archive.ubuntu.com/ubuntu hardy main multiverse"
$ sudo add-apt-repository "deb http://archive.ubuntu.com/ubuntu hardy-updates main multiverse"
```

或者编辑 sources.list：

```
$ vim /etc/apt/sources.list
```

在 sources.list 文件尾部添加如下内容并保存：

```
deb http://archive.ubuntu.com/ubuntu hardy-updates main multiverse
deb http://archive.ubuntu.com/ubuntu hardy main multiverse
```

说明

上列网址仅供参考，请确认网站 http://archive.ubuntu.com/ubuntu 是否存在 hardy 和 hardy-updates 目录，此网站会实时更新，假如不存在换个存在的链接即可。

步骤 2 更新安装包。

```
$ sudo apt-get update
```

步骤 3 安装工具包。

```
$ sudo apt-get install git-core gnupg flex bison gperf \
build-essential zip curl zlib1g-dev libc6-dev lib32ncurses5-dev \
ia32-libs x11proto-core-dev libx11-dev lib32readline5-dev \
lib32z-dev libgl1-mesa-dev g++-multilib mingw32 tofrodos \
python-markdown libxml2-utils xsltproc python uboot-mkimage lzma
```

步骤 4 安装 JDK1.6。

```
$ sudo apt-get install sun-java6-jdk
```

用户按提示操作即可。

如果有好几个 jdk 版本，则需要切换到指定版本，方法如下：

```
$ sudo update-alternatives --config java
```

有 2 个手动模式选项可用于替换 java 版本（提供/usr/bin/java）。

表2-1 手动模式选项用于替换 java 版本

选择	路径	优先级	状态
0	/usr/lib/jvm/java-6-openjdk/jre/bin/java	1061	自动模式
1	/usr/lib/jvm/java-1.5.0-sun/jre/bin/java	53	手动模式
2	/usr/lib/jvm/java-6-openjdk/jre/bin/java	1061	手动模式

要维持当前值[*]请按回车键，或者键入选择的编号，如：1

或者通过手动编辑/etc/profile：

```
$ sudo vim /etc/profile
```

在尾部添加如下语句并保存：

```
export JAVA_HOME="/usr/lib/jvm/java-1.6.0-sun"
export PATH="/usr/lib/jvm/java-1.6.0-sun/bin":$PATH
```

这里的具体的 JDK 路径由所安装的 jdk 版本实际情况确定。

步骤 5 确认 JDK 版本号正确。

```
$ java –version
```

JDK1.6 版本号如下：

```
java version "1.6.0_26"
Java(TM) SE Runtime Environment (build 1.6.0_26-b03)
Java HotSpot(TM) 64-Bit Server VM (build 20.1-b02, mixed mode)
```

经验证，JDK1.6 版本中，最低小版本为：java version "1.6.0_24"，如果低于此版本，如 java version "1.6.0_19"等，请升级至 java version "1.6.0_24"或更高版本，否则会编译出错。

步骤 6 安装交叉编译工具链。

第一次拿到发布包，或遇到发布包中交叉编译工具链的版本更新时才需要安装交叉编译工具链。

安装交叉编译工具链必须有 root 权限。另外，在执行安装命令前，请确保服务器上的 shell 是 bash，目前发布包只支持在 bash 下运行。如果服务器上的 shell 不是 bash，推荐解决办法如下：

卸载 dash 或者把默认的 sh 改成 bash。一般删除原来的 sh 软链接，重新建立一个指向 bash 的软链接即可：

```
$ cd /bin
$ rm –f sh
$ ln –s /bin/bash /bin/sh
```

另外，安装交叉编译工具链的时候会覆盖之前已经安装过同名编译器，如果以前使用过海思的相关编译器，并且做了修改，那么修改将会丢失。因此如果做了修改，请备份后再进行安装。

交叉编译工具链路径：

```
Software/ServerInstall/arm-hisiv200-linux
```

切换至 root 用户后，在交叉工具链安装包目录下执行命令：

```
$ ./cross.install
```

退出服务器登录，重新登录服务器。

输入“arm-hisiv200-linux-”后按 tab 键，如果能自动将命令补齐，则说明安装成功。

如果无法自动补齐，通过下面命令确认环境变量是否正确：

```
$ echo $PATH
```

如果不正确请用 root 用户编辑/etc/profile 文件：

```
$ sudo vi /etc/profile
```

添加如下语句：

```
export PATH="/home/root/bin/x86-arm/arm-hisiv200-linux/target/bin:$PATH"
```

步骤 7 设置最大打开文件数。

当编译过程中遇到如下错误时：

```
Exception in thread "main" java.lang.RuntimeException: error while
reading to file: java.io.FileNotFoundException, msg:cts/tools/vm-
tests/src/dot/junit/opcodes/aget_wide/Test_aget_wide.java (Too many open
files)
      at
util.build.BuildDalvikSuite.parseTestMethod(BuildDalvikSuite.java:743)
      at
util.build.BuildDalvikSuite.handleTests(BuildDalvikSuite.java:359)
      at util.build.BuildDalvikSuite.compose(BuildDalvikSuite.java:170)
      At util.build.BuildDalvikSuite.main(BuildDalvikSuite.java:136)
```

需要设置打开最大文件数限制，方法如下：

```
$ sudo vi /etc/security/limits.conf
```

$ 在最后一行加入如下内容：

```
*               -      nofile          8192
```

添加完成后，limits.conf文件最后几行应该类似如下内容：

```
#<domain>      <type>  <item>         <value>
#


#*               soft    core            0
#root            hard    core            100000
#*               hard    rss             10000
#@student        hard    nproc           20
#@faculty        soft    nproc           20
#@faculty        hard    nproc           50
#ftp             hard    nproc           0
#ftp             -       chroot          /ftp
#@student        -       maxlogins       4


# End of file
*               -      nofile          8192
```

重新登录后，执行 ulimit –n：检查是否设置正确，如果配置正确，命令执行结果应为：8192

步骤 8 确认服务器 umask 的值

在编译服务器 shell 中输入 umask 命令，查看返回值是否为 0022

```
$ umask
```

如果不是，则需要编辑/etc/profile 文件

```
$ sudo vi /etc/profile
```

添加如下语句：

```
umask 022
```

重新登录后，确认 umask 值是否正确即可。

步骤 9 建立快速编译 cache 目录

在服务器根目录建立/run/shm 目录

```
$ sudo mkdir -p /run/shm/
```

给所有用户开放权限

```
$ sudo chmod 777 -R /run/shm/
```

注意

快速编译功能为 Android 官方提供，详情请参考 Android 官网相关说明：

http://source.android.com/source/initializing.html#setting-up-ccache

Ubuntu 12.04 及以上版本不需要执行步骤 9。

----结束

2.3 开发编译

2.3.1 源码获取

源码在发布包目录下的 Software 目录下，名为 HiSTBAndroidVXXXR001CXXSPCXXX.tar.gz

将代码包拷贝至服务器上，执行命令解压源码包：

tar xzf HiSTBAndroidVXXXR001CXXSPCXXX.tar.gz

2.3.2 环境配置

进入代码根目录下，执行：

```
source build/envsetup.sh
lunch Hi3798MV100-eng
```

注意

- 每次重新登录或者切换编译目录后，都需要执行上述命令。
- 本节中所有的命令，除 2.3.12 以外，都需要在代码根目录下执行。
- 2.3.3 、2.3.4 、2.3.8 、2.3.13 小节中描述命令都会同时编译出烧写在 Nand 器件上的镜像和烧写在 Emmc 器件上的镜像，其中，烧写在 Nand 器件上镜像的输出目录为 out\target\product\Hi3798MV100\Nand，烧写在 Emmc 器件上镜像的输出目录为：out\target\product\Hi3798MV100\Emmc。
- 本节中所有的命令不相互依赖，均可以独立执行。
- lunch 命令后的参数 Hi3798MV100-eng 含义如下：Hi3798MV100：产品名称，编译结果目录 out 目录的目录结构根据此名称生成，out/target/product/"产品名称"/，如 out/target/product/Hi3798MV100/对应的产品代码目录在 device/hisilicon/"产品名称"/，如：device/hisilicon/Hi3798MV100：编译默认为 eng 模式。
- 获取当前版本支持的所有产品名称，可以在执行完 source buid/envsetup.sh 命令后，直接执行 lunch 命令即可。
- 不同的参数对应不同的产品，本文中以 Hi3798MV100 产品为例，说明版本使用方式，其他产品以此类推。
- 代码编译文件 Android.mk 中，LOCAL_MODULE_TAGS 的默认值都为 optional，此种配置在编译完成后不会将编译生成文件自动拷贝至编译结果目录目录。如果需要编译时默认拷贝至编译结果目录，编译生成文件为动态库、jar 包、可执行文件和预编译文件时，需要在 Android.mk 中增加：ALL_DEFAULT_INSTALLED_MODULES += $(LOCAL_MODULE)编译生成文件为 apk 时，则需要在 Android.mk 中增加：ALL_DEFAULT_INSTALLED_MODULES += $(LOCAL_PACKAGE_NAME)

2.3.3 完整编译

进入代码根目录下，执行：

```
make bigfish -j32 2>&1 | tee bigfish.log
```

此命令会编译出下面的镜像：

out\target\product\Hi3798MV100\Nand 目录下：

```
Android系统内核镜像:                kernel.img
Android system分区镜像:             system_[x]_[y].ubi
Android userdata分区镜像:           userdata_[x]_[y].ubi
Android cache分区镜像:              cache_[x]_[y].ubi
```

Android sdcard分区镜像：	sdcard_[x]_[y].ubi
bootargs分区镜像：	bootargs.bin
分区表：	Hi3798MV100.xml
recovery小系统内核镜像：	recovery.img
recovery用update包：	update.zip
fastboot镜像：	fastboot.bin
默认预编译的baseparam分区镜像：	baseparam.img
默认预编译的logo分区镜像：	logo.img

其中 ubi 镜像中[x]为页大小，[y]为块大小，如，system_4K_1M.ubi，请根据实际情况选择烧写。

out\target\product\Hi3798MV100\Emmc 目录下：

Android系统内核镜像：	kernel.img
Android system分区镜像：	system.ext4
Android userdata分区镜像：	userdata.ext4
Android cache分区镜像：	cache.ext4
Android sdcard分区镜像：	sdcard.ext4
bootargs分区镜像：	bootargs.bin
分区表：	Hi3798MV100-emmc.xml
recovery小系统内核镜像：	recovery.img
recovery用update包：	update.zip
fastboot镜像：	fastboot.bin
默认预编译的baseparam分区镜像：	baseparam.img
默认预编译的logo分区镜像：	logo.img

out\target\product\Hi3798MV100\目录下：

烧写工具：	HiTool-[version](STB).zip

其中，[version]为 HiTool 工具的某一个发布版本号，具体版本请以发布包中的为准。

注意

- -j32 参数为执行命令的线程数，请根据实际情况调整-j 参数后面数字的大小。
- bigfish.log 在代码根目录下，记录了编译过程的打印信息。
- 其他编译命令也有类似规则，不再赘述。

2.3.4 编译 Android 内核分区镜像

进入代码根目录下，执行：

```
make kernel –j32 2>&1 | tee kernel.log
```

此命令会编译出下面的镜像：

```
kernel.img
```

此镜像会分别拷贝至 out\target\product\Hi3798MV100\目录的 Nand 目录和 Emmc 目录下。即，Nand 器件上使用的 Android 内核分区镜像与 Emmc 器件上使用的 Android 内核分区镜像相同。

编译过程文件存放在：

```
out/target/product/Hi3798MV100/obj/KERNEL_OBJ/
```

2.3.5 修改 Android 内核配置

进入代码根目录下，执行：

```
make kernel_menuconfig
```

- 默认 Android 内核配置文件记录在 device/hisilicon/Hi3798MV100/BoardConfig.mk 文件的 ANDROID_KERNEL_CONFIG 变量中。
- 此命令修改的内核配置文件存放在临时目录：out/target/product/Hi3798MV100/obj/KERNEL_OBJ/.config，请注意将修改后的文件保存至：device/hisilicon/bigfish/sdk/source/kernel/linux-3.10.y/arch/arm/configs/目录下对应文件。
- 修改完成后执行 2.3.4 节中的命令，会编译出新的内核镜像。
- 如需要修改 Android 系统编译内核时所使用的内核配置文件名称，请修改 device/hisilicon/Hi3798MV100/BoardConfig.mk 中 ANDROID_KERNEL_CONFIG 变量的值为新的文件名，并同步修改 device/hisilicon/bigfish/sdk/configs/目录下的对应的 sdk 配置文件中 CFG_HI_KERNEL_CFG 变量的值。对应的 sdk 的配置文件的名称由 device/hisilicon/Hi3798MV100/BoardConfig.mk 中的 HISI_SDK_ANDROID_CFG 变量定义。新的内核配置文件请保存到 device/hisilicon/bigfish/sdk/source/kernel/linux-3.10.y/arch/arm/configs/目录下。
- 内核配置文件要求均以_defconfig 结尾。
- 建议执行此命令前，删除内核临时文件存放目录：out/target/product/Hi3798MV100/obj/KERNEL_OBJ/

2.3.6 编译 NAND Flash 器件上的 Android 系统分区镜像

进入代码根目录下，执行：

```
make ubifs -j32 2>&1 | tee ubifs.log
```

此命令会编译出下面的镜像：

out\target\product\Hi3798MV100\Nand 目录下：

```
system_[x]_[y].ubi
userdata_[x]_[y].ubi
cache_[x]_[y].ubi
sdcard_[x]_[y].ubi
kernel.img
```

其中，.ubi 扩展名的镜像，如：

```
system_4K_1M.ubi
data_4K_1M.ubi
cache_4K_1M.ubi
```

```
sdcard_4K_1M.ubi
```

4K: PageSize：页大小

1M: BlockSize：块大小

2.3.7 编译 Emmc 器件上的 Android 系统分区镜像

进入代码根目录下，执行：

```
make ext4fs -j32 2>&1 | tee ext4fs.log
```

此命令会编译出下面的镜像：

out\target\product\Hi3798MV100\Emmc 目录下：

```
system.ext4
userdata.ext4
cache.ext4
sdcard.ext4
kernel.img
```

2.3.8 编译 Android recovery 小系统内核分区镜像

进入代码根目录下，执行：

```
make recoveryimg -j32 2>&1 | tee recovery.log
```

此命令会编译出下面的镜像：

```
recovery.img
```

此镜像会分别拷贝至 out\target\product\Hi3798MV100\目录的 Nand 目录和 Emmc 目录下。即，Nand 器件上使用的 Android recovery 小系统内核分区镜像与 Emmc 器件上使用的 Android recovery 小系统内核分区镜像相同。

编译过程文件存放在：

```
out/target/product/Hi3798MV100/obj/RECOVERY_OBJ/
```

2.3.9 修改 Android recovery 小系统内核配置

进入代码根目录下，执行：

```
make recovery_menuconfig
```

注意

- 默认 Android recovery 小系统内核配置文件记录在 device/hisilicon/Hi3798MV100/BoardConfig.mk 文件的 RECOVERY_KERNEL_CONFIG 变量中。
- 此命令修改的内核配置文件存放在临时目录：out/target/product/Hi3798MV100/obj/RECOVERY_OBJ/.config，请注意将修改后的文件保存至：device/hisilicon/bigfish/sdk/source/kernel/linux-3.10.y/arch/arm/configs/目录下对应文件。
- 修改完成后执行 2.3.8 节中的命令，会编译出新的 Android recovery 小系统内核镜像。
- 如需要修改默认 Android recovery 小系统内核配置文件名称，请修改 device/hisilicon/Hi3798MV100/BoardConfig.mk 中 RECOVERY_KERNEL_CONFIG 变量的值为新的文件名。新的内核配置文件请保存到 device/hisilicon/bigfish/sdk/source/kernel/linux-3.10.y/arch/arm/configs/目录下。
- 内核配置文件要求均以_defconfig 结尾。
- 建议执行此命令前，删除内核临时文件存放目录：out/target/product/Hi3798MV100/obj/RECOVERY_OBJ/

2.3.10 编译 Android recovery 升级包 update.zip

进入代码根目录下，执行：

```
make updatezip -j32 2>&1 | tee updatezip.log
```

此命令会编译出下面的镜像：

out\target\product\Hi3798MV100\Nand 目录下：

```
update.zip
```

out\target\product\Hi3798MV100\Emmc 目录下：

```
update.zip
```

如不需要 recovery update 包，此命令可以不执行。

2.3.11 清除编译结果

进入代码根目录下，执行：

```
make clean
```

注意

- 所有的编译过程文件均在 out 目录下，执行此命令将清除所有编译结果。
- 如需单独清除 Android 系统内核临时文件、Android recovery 小系统内核临时文件和 fastboot 临时文件，直接删除对应的临时文件目录即可。

2.3.12 修改海思 SDK 配置文件

以 Hi3798MV100 为例，SDK 配置文件存放目录：

```
device/hisilicon/bigfish/sdk/configs/hi3798mv100/
```

此命令需要到 SDK 目录下执行：

```
cd device/hisilicon/bigfish/sdk
```

执行命令：

```
make menuconfig SDK_CFGFILE=configs/hi3798mv100/需要修改的配置文件名
```

可以修改对应的配置文件，如：

```
make menuconfig
SDK_CFGFILE=configs/hi3798mv100/hi3798mdmo_hi3798mv100_android_cfg.mak
```

保存退出后，修改将保存在 device/hisilicon/bigfish/sdk/configs/hi3798mv100/目录下对应的配置文件。

注意

- 产品使用的 SDK 配置文件的名称在各产品 BoardConfig.mk 的 HISI_SDK_ANDROID_CFG 变量中定义，如：
  device/hisilicon/Hi3798MV100/BoardConfig.mk
  HISI_SDK_ANDROID_CFG :=hi3798mdmo_hi3798mv100_android_cfg.mak

- 执行此命令会通过弹出的图形界面修改 SDK 配置文件，修改后保存退出，修改后的配置文件保存在 SDK 配置文件存放目录，原配置文件保存为："配置文件名.old"，存放在 SDK 配置文件存放目录。
- 用户也可以进入 SDK 配置文件存放目录，直接手动修改对应的配置文件。
- 修改前，请确定所使用的 SDK 配置文件的文件名。

2.3.13 编译 fastboot 分区镜像

进入代码根目录下，执行：

```
make hiboot -j32 2>&1 | tee hiboot.log
```

此命令会编译出下面的镜像：

out\target\product\Hi3798MV100\Nand 目录下：

```
fastboot-burn-nand.bin
```

编译过程文件存放在：

```
out/target/product/Hi3798MV100/obj/NAND_HIBOOT_OBJ/
```

out\target\product\Hi3798MV100\Emmc 目录下：fastboot.bin

编译过程文件存放在：

```
out/target/product/Hi3798MV100/obj/EMMC_HIBOOT_OBJ/
```


注意

- 完整编译时，会调用此命令编译 fastboot
- 编译 fastboot 依赖于 device/hisilicon/Hi3798MV100/BoardConfig.mk 中定义的 sdk 配置文件中对 fastboot 的配置：

```
# emmc fastboot configure
EMMC_BOOT_CFG_NAME :=
hi3798mdmo1a_hi3798mv100_ddr3_1gbyte_16bitx2_4layers_emmc.cfg
EMMC_BOOT_REG_NAME :=
hi3798mdmo1a_hi3798mv100_ddr3_1gbyte_16bitx2_4layers_emmc.reg
EMMC_BOOT_ENV_STARTADDR :=0x100000
EMMC_BOOT_ENV_SIZE=0x10000
EMMC_BOOT_ENV_RANGE=0x10000

# nand fastboot configure
NAND_BOOT_CFG_NAME :=
```

```
hi3798mdmo1a_hi3798mv100_ddr3_1gbyte_16bitx2_4layers_nand.cfg
NAND_BOOT_REG_NAME :=
hi3798mdmo1a_hi3798mv100_ddr3_1gbyte_16bitx2_4layers_nand.reg
NAND_BOOT_ENV_STARTADDR :=0x800000
NAND_BOOT_ENV_SIZE=0x10000
NAND_BOOT_ENV_RANGE=0x10000
```

如需修改 fastboot 中配置，如 bootargs 的存放位置、reg 文件等，请在 BoardConfig.mk 中修改好对应的配置文件后，再执行此命令。

- **此命令可以直接执行，不需要完整编译以后再执行。**
- **fastboot 镜像除 fastboot 本身功能外，还与分区表，bootargs 分区镜像相关，如其中一个更新，则需要对应更新其他的镜像。**
- **如不需要更新 fastboot 镜像，此命令可以不执行。**

2.4 镜像烧写

2.4.1 Flash 分区表

以 NAND Flash 器件为例，用户可以根据单板 Flash 配置分配 Flash 分区表，但是要遵循如下原则，Emmc 器件请参考默认的 Emmc 器件的分区表。

- SPI Flash 上每个分区大小都要是 page size 的整数倍
- NAND Flash 上每个分区大小都要是 block 大小的整数倍
- NAND Flash 上每个分区要有一定剩余空间，并保证至少 2 个 block 的坏块冗余

以下面示例 Flash 分区表为例说明，如表 2-2 所示。

表2-2 示例 Flash 分区表

序号	分区名称	功能	Flash 空间的实际需求 单位：Byte	分配的 Flash 空间 单位：Byte	备注	是否可裁剪
1	fastboot	Bootloader，启动系统。 根据按键或 misc 分区的内容进入 Recovery 模式或系统启动模式	512K	512K	SPI Flash	否
2	bootargs	存放的分区信息、内存信息、内核启动信息等。	64K	64K	SPI Flash	否

序号	分区名称	功能	Flash 空间的实际需求 单位：Byte	分配的 Flash 空间 单位：Byte	备注	是否可裁剪
3	recovery	Recovery 模式。 内容包括 Kernel、recovery rootfs 及 recovery app。	3456K	3456K	SPI Flash	否
4	deviceinfo	存放 MAC Addr、Device ID、产品 ID 等设备信息。	64K	64K	SPI Flash	否
5	baseparam	开机画面显示、瞬播的参数区，由 HiTool 工具来生成；具体参数包括：开机画面显示宽高、DAC 系数、制式。	1K	8M	NAND Flash	否
6	pqparam	存放图像质量参数	200K	8M	NAND Flash	否
7	logo	开机画面，由 HiTool 工具生成。	1M 根据实际情况大小可能变化	20M	NAND Flash	否
8	logobak	开机画面备份分区，由 HiTool 工具生成，当 logo 分区失效时，此分区起作用	1M 根据实际情况大小可能变化	20M	NAND Flash	否
9	fastplay	开机动画，由 HiTool 工具来生成	10M 根据实际情况大小可能变化	40M	NAND Flash	否
10	fastplaybak	开机动画备份分区，由 HiTool 工具生成，当 fastplay 分区失效时，此分区起作用	10M 根据实际情况大小可能变化	40M	NAND Flash	否
11	kernel	Linux kernel + rootfs	10M	40M	NAND Flash	否
12	misc	Bootloader Control Block，存放 recovery bootloader message	512K	20M	NAND Flash	否
13	system	Android 系统分区	200M 根据实际情况大小可能变化	500M	NAND Flash	否
14	userdata	用户数据	100M 根据实际情况大小可能变化	1024M	NAND Flash	否

序号	分区名称	功能	Flash 空间的实际需求 单位：Byte	分配的 Flash 空间 单位：Byte	备注	是否可裁剪
15	cache	用于 recovery 过程的 cmd、log 及 intent 信息交互 用于存放升级包（fastboot、kernel）、补丁包（system）	50M 如果 system 用升级包，则此分区需要配大	100M	NAND Flash	否 大小可根据升级的需求配置
16	sdcard	内置 sd 卡分区	2350M	2284M	NAND Flash	否

注意

对于分区表，本文档给出的是一个 demo，可能和发布代码不一致。请以实际的分区表为准。

2.4.2 Flash 烧写

2.4.2.1 烧写准备

以烧写 Nand 器件为例：

准备好编译生成的镜像，编译结果目录位置：

`out/target/product/Hi3798MV100/Nand`

- fastboot 镜像名称：fastboot.bin
- bootargs 镜像名称：bootargs.bin
- baseparam 镜像名称：baseparam.img
- logo 镜像名称：logo.img
- kernel 镜像名称：kernel.img
- recovery 镜像名称：recovery.img
- recovery 用 update 包名称：update.zip
- system 镜像名称：system_[x]_[x].ubi
- userdata 镜像名称：userdata_[x]_[x].ubi
- cache 镜像名称：cache_[x]_[x].ubi
- sdcard 镜像名称：sdcard_[x]_[x].ubi
- 烧写工具：out/target/product/Hi3798MV100/HiTool-[version](STB).zip

2.4.2.2 烧写方法

步骤 1 连接好单板的网口线、串口线及电源线。

步骤 2 打开烧写工具，输入正确的 PC 配置和单板配置，传输方式选择“网口烧写”。

步骤 3 选择按分区烧写，选择分区表文件，如 demo 分区表：Hi3798MV100-nand.xml。

步骤 4 选择需要烧写的镜像。

步骤 5 点击“烧写”按钮并重新给单板上电。

步骤 6 烧写成功后，会提示“烧写成功”。

----结束

3 内存配置

说明

本章节以 Hi3798MV100 为例，其他芯片类似。

3.1 内存分配方式概述

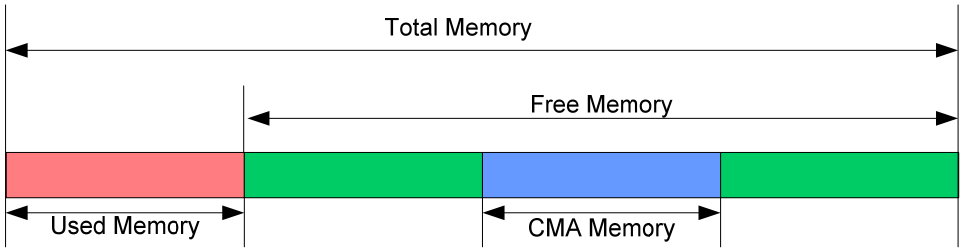
Android 解决方案采用 CMA 的方式分配大块连续物理内存供驱动使用。系统启动时，会根据配置的 CMA 内存大小，从系统空闲内存中，分配连续的物理内存。

配置 CMA 内存大小的方法有两种，bootargs 配置优先级比内核配置高：

- 在 bootargs 环境变量中配置。
- 在内核配置文件中配置。

CMA 内存在整个内存中的分布如图 3-1 所示。

图3-1 CMA 内存分布示意图



CMA 的内存分配优势是可以与 Kernel 共享，当 CMA 内存有空闲时，Kernel 可以使用 CMA 的内存，比预留一块 Kernel 不管理的内存分配方法，能更有效的利用内存。
CMA 分配的内存是物理连续的，主要提供给编解码，转码，图形显示等模块使用。

3.2 修改 CMA 内存配置

如果除 DDR 大小外，其他硬件无差异，则 512MB、1GB、2GB DDR 大小的三种硬件单板可以使用同一种镜像烧写。不同 DDR 大小的 bootargs 参数，在 bootargs 分区中，用不同的环境变量配置，如：

```
512MB DDR: bootargs_512M=mem=512M mmz=ddr,0,0,235M
1GB DDR: bootargs_1G=mem=1G mmz=ddr,0,0,400M
2GB DDR: bootargs_2G=mem=2G mmz=ddr,0,0,600M
```

CMA 内存大小可以通过配置对应 DDR 大小的环境变量修改，也可以通过修改 bootargs 环境变量修改。但是，如果 bootargs 环境变量中配置了 CMA 内存大小后，对应 DDR 大小的环境变量的配置将失效。

说明

由于 bootargs 分区不建议用户经常升级，因此该方法建议仅用于调试时使用。

通过修改 bootargs 环境变量，修改 CMA 内存分配大小的方法步骤如下：

步骤 1 按住“ctrl+C”键重启单板进入 fastboot 命令行。

步骤 2 输入命令 printenv 读取 bootargs 环境变量的配置。

```
printenv
```

步骤 3 修改环境变量中 DDR 和 CMA 的配置：

- 方法一：在对应 DDR 大小的环境变量中修改默认 CMA 配置。修改 mmz=ddr,0,0,XXXM 中 XXX 的值为实际需要的值即可。
- 方法二：在 bootargs 环境变量的最后加上 mem=YYY mmz=ddr,0,0,XXXM，YYY 为总 DDR 大小，XXX 为 CMA 大小。如需要将 DDR 配置为 512MB，CMA 配置为 220MB，则需要在 bootargs 最后加上 mem=512M mmz=ddr,0,0,220M，即：

修改前 bootargs 环境变量：

```
console=ttyAMA0,115200
blkdevparts=mmcblk0:1M(fastboot),1M(bootargs),10M(recovery),2M(deviceinfo),8M(baseparam),8M(pqparam),20M(logo),20M(logobak),40M(fastplay),40M(fastplaybak),40M(kernel),20M(misc),8M(userapi),8M(hibdrv),8M(qbflag),300M(qbdata),800M(system),1024M(userdata),100M(cache),-(sdcard)
```

修改后 bootargs 环境变量：

```
console=ttyAMA0,115200
blkdevparts=mmcblk0:1M(fastboot),1M(bootargs),10M(recovery),2M(deviceinfo),8M(baseparam),8M(pqparam),20M(logo),20M(logobak),40M(fastplay),40M(fastplaybak),40M(kernel),20M(misc),8M(userapi),8M(hibdrv),8M(qbflag),300M(qbdata),800M(system),1024M(userdata),100M(cache),-(sdcard) mem=512M
mmz=ddr,0,0,220M
```

注意

新增加的红色字符串和前面的字符串之间要有空格隔开。另外，方法二修改完成后，默认三种 DDR 大小对应的环境变量配置将失效，DDR 大小和 CMA 大小将以 bootargs 环境变量配置为准。如需使用默认三种 DDR 大小对应的环境变量配置，只需将 bootargs 环境变量恢复原值即可。

步骤 4 保存环境变量，重启。在串口输入：

```
saveenv
reset
```

----结束

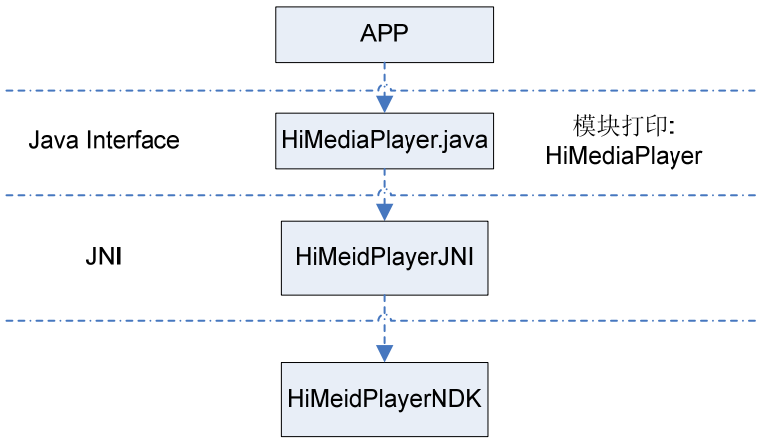
4 HiMediaPlayer

4.1 概述

HiMediaPlayer 提供了一套多媒体播放的整个框架。通过该框架提供的相关接口，上层应用程序可以实现在 Android 环境下多种媒体类型的播放，并且具有海思媒体播放的扩展接口，使用底层海思媒体播放模块，使播放性能更佳，并且可以在字幕，音轨方面实现更多功能。

HiMediaPlayer 媒体播放架构如图 4-1 所示。

图4-1 HiMediaPlayer 媒体播放架构图



模块说明：

- Java Interface

  JAVA 对外接口层主要分为两部分：

  - HiMediaPlayer.java

    主要向应用层提供媒体播放接口。

  - HiBDInfo.java

    用以实现蓝光原盘文件夹或 ISO 等蓝光光盘的播放导航。获取蓝光相关媒体信息，如 Title 等信息。

- JNI：

  对应上层应用提供媒体播放接口和蓝光信息查询，对下层调用 NDK 和 libbluray 实现媒体播放。

4.2 重要概念

【HiPlayer】

海思媒体播放模块。

【Surface】

视频播放的显示区域。

【ISO】

一种镜像文件，蓝光码流常以这种文件方式存在。

4.3 功能描述

4.3.1 功能特点

HiMediaPlayer相对Android原生架构具有如下特点：

- 支持的音视频格式更多。

  利用海思媒体播放模块 HiPlayer，可以支持比 Android 原生更多的音视频格式。

  Android 原生支持格式：MP4，M4A，MKV，MP3，WAV，FLAC，OGG，AAC，MPEG-TS，3GP，MIDI-MID，MIDI-XMF，MIDI-MXMF，MIDI-RTTTL，MIDI-RTX，MIDI-OTA，MIDI-IMY，WEBM，M3U8

- 扩展格式：ASF，M1V，M2V，MPG/DATA，VOB，TS，MTS/M2TS，AVI，MKV，RM，RMVB，WMV，MOV，M4V，F4V，3GP，3G2，tp，trp，m2p，cue，ape，m3u8，m3u9，pls，iso，BDMV dir
- 接口功能更多。

  可以支持快进、快退，切字幕，切音轨等功能，更加符合电视、机顶盒产品播放器的形态。

- 性能更高。

  利用海思媒体播放模块 HiPlayer 使高清码流、蓝光码流播放更流畅。

4.3.2 功能接口

基本播放监听功能。该功能模块提供以下 API：

- HiMediaPlayer：初始化 HiMediaPlayer
- setOnPreparedListener：设置 prepare 函数的回调接口
- setOnCompletionListener：设置播放完成的回调接口

- setOnBufferingUpdateListener：设置 Buffer 状态监听回调接口
- setOnSeekCompleteListener：设置 Seek 完成的回调接口
- setOnVideoSizeChangedListener：设置窗口状态变化的回调接口
- setDataSource：设置播放数据源
- setDisplay：设置播放的应用层 Surface
- setVideoRange：设置播放的视频显示区域
- prepare：准备播放（同步）
- prepareAsync：准备播放（异步）
- start：开始播放
- pause：暂停播放
- stop：停止播放
- reset：重置
- release：释放 HiMediaPlayer

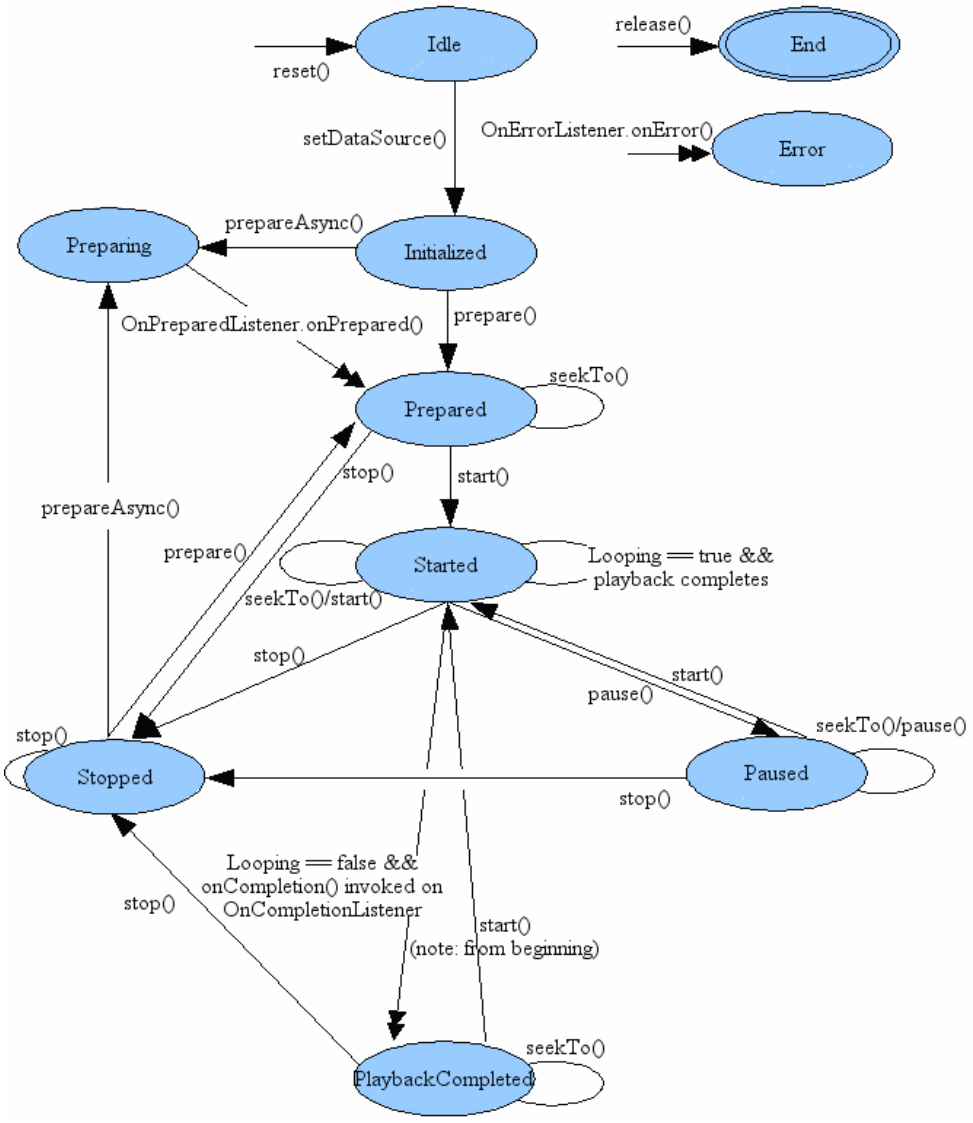
还支持海思扩展功能：

- setAudioTrack：设置播放的音频流 ID
- setAudioChannel：设置音频播放声道模式
- setSubFontColor：设置字幕显示时颜色
- setSubFontStyle：设置字幕显示时字体类型
- setSubFontSize：设置字幕显示字体大小
- setSubFontSpace：设置字幕显示时字符间距
- setSubFontLineSpace：设置字幕显示时行间距大小
- setSubEncode：设置字幕编码格式
- setSubTrack：设置字幕显示流 ID
- setSubTimeOffset：设置字幕正负同步时间
- setSubVertical：设置字幕显示垂直位置
- setSubPath：导入外挂字幕
- setBufferSizeConfig：设置缓冲 buffer 中数据大小门限值
- getBufferSizeConfig：获取当前播放缓冲 buffer 配置 size 数据信息
- setBufferTimeConfig：设置缓冲 buffer 中时间大小门限值
- getBufferTimeConfig：获取当前播放缓冲 buffer 时间配置信息
- setFreezeMode：设置当前播放器冻屏模式
- getFreezeMode：获取当前播放器冻屏模式
- setStereoVideoFmt：指定视频输入格式
- setStereoStrategy：设置视频输出格式

4.3.3 模块原理

HiMediaPlayer 运行流程如图 4-2 所示。

图4-2 HiMediaPlayer 运行流程



HiMediaPlayer 的调用流程如下：

步骤 1 准备阶段：

1. 创建一个 HiMediaPlayer 对象；
2. 设置各种回调函数；
3. 设置播放文件的路径；
4. 设置 Surface，用于字幕，视频显示；
5. 设置视频显示区域左上角坐标：X,Y;视频宽高：W,H
6. 使用 preapare 或者 parepareAsync 函数进行文件识别和解码器等初始化；
7. 监听 OnPraprae 回调，完成播放器的准备工作，播放器状态从 Idle 到 Prepared 状态。

步骤 2 播放控制。

在 prepare 之后，调用各种同步操作控制命令，如 Start/Stop/Pause/setSpeed/getXXXInfo 等。

步骤 3 Seek 操作：

- 由于 seek 操作在底层需要进行文件位置重定位，数据缓冲区调整以及解码器清空等诸多操作，因此 seek 操作是异步操作；
- 调用 seek 后，需要监听 OnSeekComplete()回调函数来完成整个 seek 操作；
- 整个 seek 过程完成前，系统处于 Started 状态，reset,release 操作以及再次的 seek 操作是不允许的。

步骤 4 事件处理。

播放器在运行过程中，会产生各种信息事件（如 buffer 状态变更等）以及可能产生多种错误事件（如文件不支持）等，播放器应用需要必须监听错误事件，以及根据需要监听各种信息事件，以及监听文件播放结束的事件。

步骤 5 播放结束。

- 根据状态图，应用程序可以随时调用 reset 函数来关闭 HiPlayer 打开的文件，释放当前 HiPlayer 占用的各种资源。或者调用 release 函数（内部自动调用 reset 函数）来结束整个播放过程。
- 播放器 reset 之后会清空所有播放信息，如果再播放，需要从头开始整个播放流程。

----结束

4.4 开发指引

HiMediaPlayer 开发有如下几个典型场景：

- HiMediaPlayer JNI 播放
- 开源定制化指引
- SmoothStraming

4.4.1 HiMediaPlayer JNI 播放

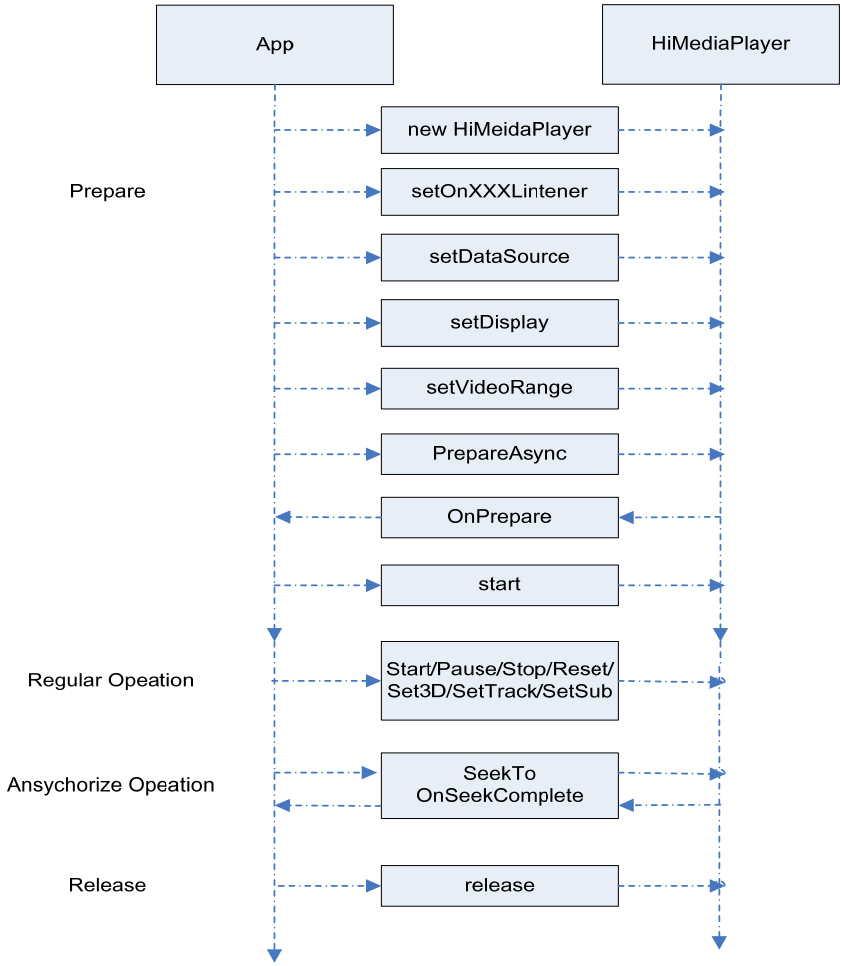
场景说明

在海思 HiSTBAndroidVX00R001 项目 Android 解决方案的源码中，利用 HiMediaPlayer 接口开发视频播放等 Android 应用。

工作流程

HiMediaPlayer 播放流程如图 4-3 所示。

图4-3 HiMediaPlayer 播放流程



相应步骤：

步骤 1 HiMediaPlayer 静态包引入

在 Android.mk 文件中静态链接 jar 包

LOCAL_STATIC_JAVA_LIBRARIES += HiMediaPlayer

步骤 2 在代码中引用 HiMediaPlayer 包

引用 HiMediaPlayer 包。

```
import com.hisilicon.android.mediaplayer.HiMediaPlayer;
import
com.hisilicon.android.mediaplayer.HiMediaPlayer.OnCompletionListener;
import com.hisilicon.android.mediaplayer.HiMediaPlayer.OnErrorListener;
import
com.hisilicon.android.mediaplayer.HiMediaPlayer.OnFastBackwordCompleteLis
tener;
import com.hisilicon.android.mediaplayer.HiMediaPlayer.OnInfoListener;
```

```
import  com.hisilicon.android.mediaplayer.HiMediaPlayer.OnPreparedListener;
```

步骤 3　初始化对象。

```
HiMediaPlayer  mMediaPlayer = new HiMediaPlayer();
```

步骤 4　设定侦听（回调）

- 设置 prepare 或者 prepareAsync 的回调：
  `mMediaPlayer.setOnPreparedListener(mPreparedListener);`
- 设置视频大小变化的回调：
  `mMediaPlayer.setOnVideoSizeChangedListener(mSizeChangedListener);`
- 设置播放完成的回调：
  `mMediaPlayer.setOnCompletionListener(mCompletionListener);`
- 设置快进快退完成的回调：
  `mMediaPlayer.setOnFastBackwordCompleteListener(mFastBackwordCompleteListener);`
- 设置出错的回调：
  `mMediaPlayer.setOnErrorListener(mErrorListener);`
- 设置缓存大小变化的回调：
  `mMediaPlayer.setOnBufferingUpdateListener(mBufferingUpdateListener);`

步骤 5　设置播放文件路径。

```
mMediaPlayer.setDataSource(mContext, mUri);
```

步骤 6　设置播放画面。

```
mMediaPlayer.setDisplay(mSurfaceHolder);
```

步骤 7　设置播放视频区域。

```
mMediaPlayer.setVideoRange(mX, mY, mW, mH);
```

步骤 8　准备播放。

```
mMediaPlayer.prepareAsync();
```

步骤 9　调用 onPrepared 之后

```
mMediaPlayer.start();
```

----结束

注意事项

- HiMediaPlayer 具体 API 接口请参见《HiMediaPlayer API 开发参考》
- 海思扩展接口，除了 setStereoVideoFmt，setStereoStrategy 外，其余接口都是在 Prepared 状态下调用才会设置成功。

示例

HiMediaPlayer 播放示例：

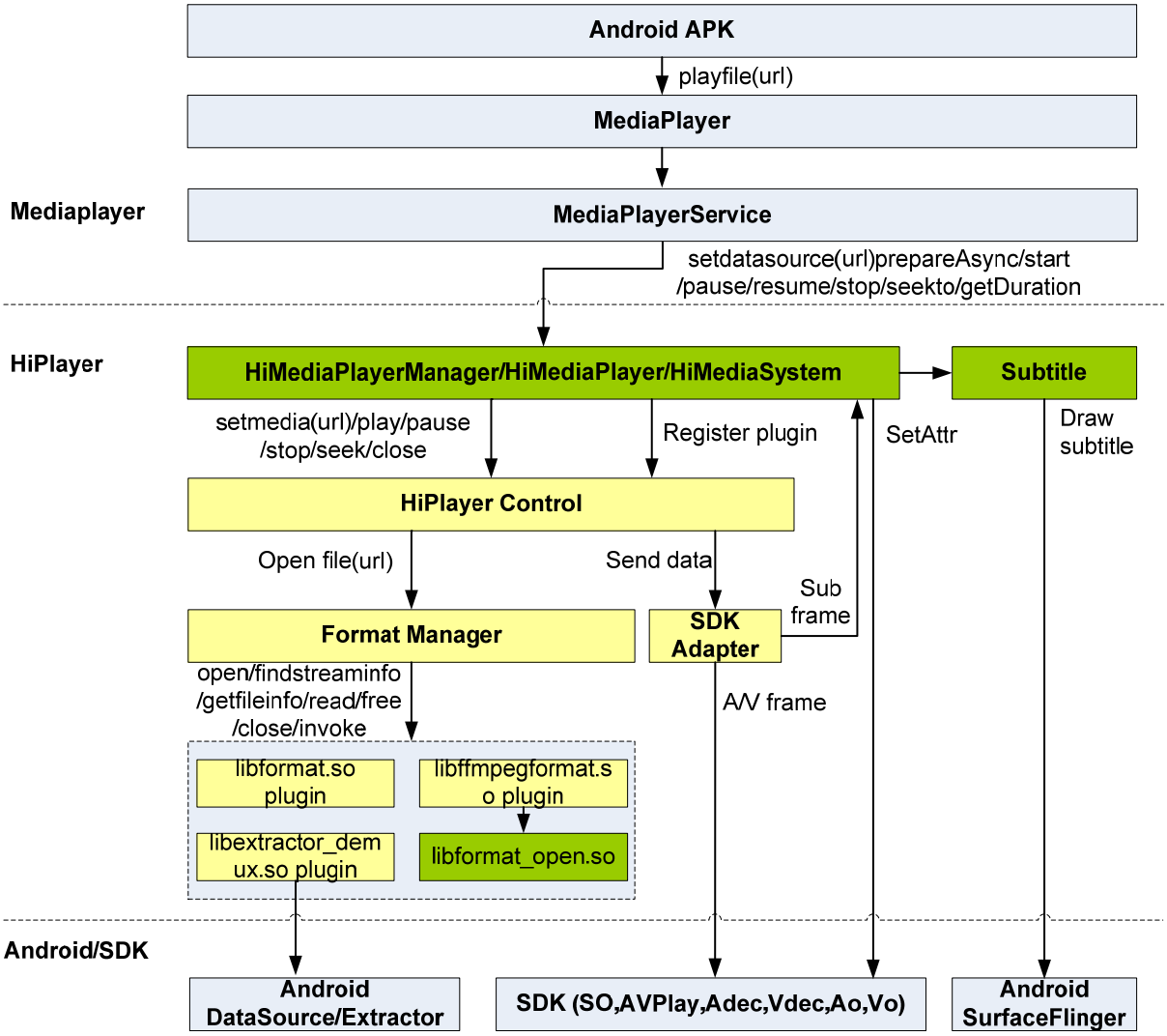
```
mMediaPlayer = new HiMediaPlayer();
mMediaPlayer.setOnPreparedListener(mPreparedListener);
mMediaPlayer.setOnVideoSizeChangedListener(mSizeChangedListener);
mMediaPlayer.setOnCompletionListener(mCompletionListener);
mMediaPlayer.setOnFastBackwordCompleteListener(mFastBackwordCompleteListe
ner);
mMediaPlayer.setOnInfoListener(m3DModeReceivedListener);
mMediaPlayer.setOnErrorListener(mErrorListener);
mMediaPlayer.setOnBufferingUpdateListener(mBufferingUpdateListener);
mCurrentBufferPercentage = 0;
mMediaPlayer.setDataSource(mContext, mUri);
mSurfaceHolder.setFixedSize(getVideoWidth(), getVideoHeight());
mMediaPlayer.setDisplay(mSurfaceHolder);
Surface mSurface;
mSurface = mSurfaceHolder.getSurface();

int[] location = new int[2];
getLocationOnScreen(location);

int mX, mY, mW, mH;
if (null != mSurface)
{
mX = location[0];
mY = location[1];
mW = mVideoWidth;
mH = mVideoHeight;
mMediaPlayer.setVideoRange(mX, mY, mW, mH);
}

mMediaPlayer.prepareAsync();
```

4.4.2 开源定制化指引

4.4.2.1 原理框架图

海思媒体播放器除了提供一套 NDK JNI 接口外，还对接到了 Android 标准的 MediaPlayer 上，与 StageFright、NuPlayer 层次平级，MediaPlayer 通过路由方式根据输入的文件选择不同的播放器，海思媒体播放器对接到 MediaPlayer 框架图如图 4-4 所示。

图4-4 HiMediaPlayer 播放流程



说明

图 4-4HiPlayer 层次，绿色（ ）部分为源码提供，浅黄色（ ）部分为二进制库提供。

框架图分为三层：

- MediaPlayer 层
- HiPlayer 层
- Android/SDK 层

HiPlayer 层结构及提供的功能如下：

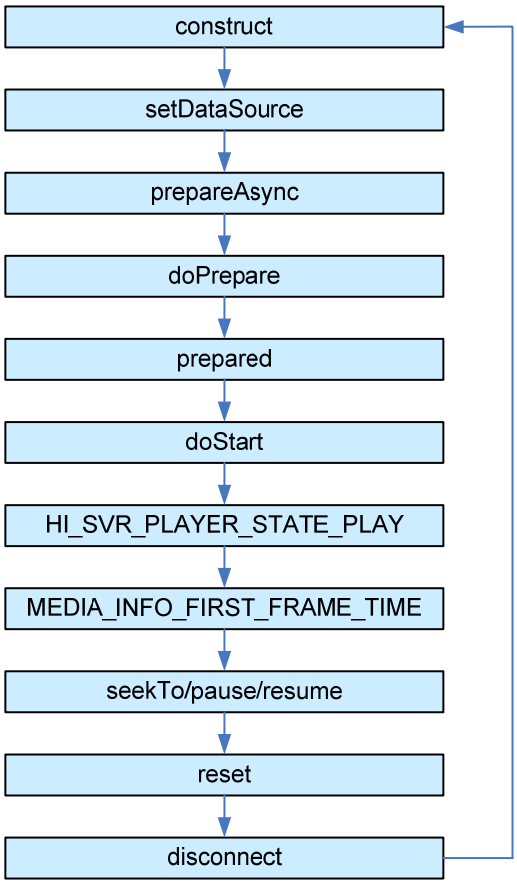
- HiMediaPlayer：HiPlayer API 到 MediaPlayer 接口适配层；
- HiMediaSystem：设备初始化模块，比如 AVPlay、HiPlayer 等；

- Subtitle：字幕输出模块，将 HiPlayer 解析出来的图像字幕或 UTF-8 编码的文本字幕绘制到 Android Surface；
- HiPlayer Control：播放器主控模块，负责接收 HiMediaPlayer 下发的播放控制命令，并执行相关的播放控制操作，同时通过事件方式上报播放状态信息给 HiMediaPlayer；
- Format Manager：播放器解析器插件管理模块，负责管理 HiMediaSystem 里面注册给 HiPlayer 的所有解析器插件，当 APK 指定播放一个媒体文件时，Format Manager 会逐个遍历解析器，直到找到一个合适的解析器，后续播放过程中使用该解析器解析媒体文件；
- SDK Adapter：SDK 适配层，负责与 SDK AVPlay、SO、AudioTrack、Window 的对接；
- libffmpegformat.so plugin：媒体文件解析器，负责 MP4、MKV、FLV、TS、AVI 等媒体文件格式的解析；
- libformat.so plugin：字幕、协议组件，包含 SRT、SMI、SSA/ASS、IDX+SUB 等格式字幕的解析器及 FILE 协议；
- libextractor_demux.so plugin：SmoothStreaming Demux 解析器，该 plugin 会调用到 Android Extractor（Android 媒体文件解析器）及 Datasource，实现 SmoothStreaming+Playready 播放功能；
- libformat_open.so：HLS、HTTP、TCP 协议，负责提供 HLS、HTTP、TCP 流媒体协议功能，是目前海思播放器流媒体协议开源部分；

4.4.2.2 播放流程

Android APK 调用 MediaPlayer 播放媒体文件流程如图 4-5 所示。（图中为播放关键步骤）。

图4-5 媒体播放关键步骤



关键步骤中 HiPlayer 对应的操作说明如下：

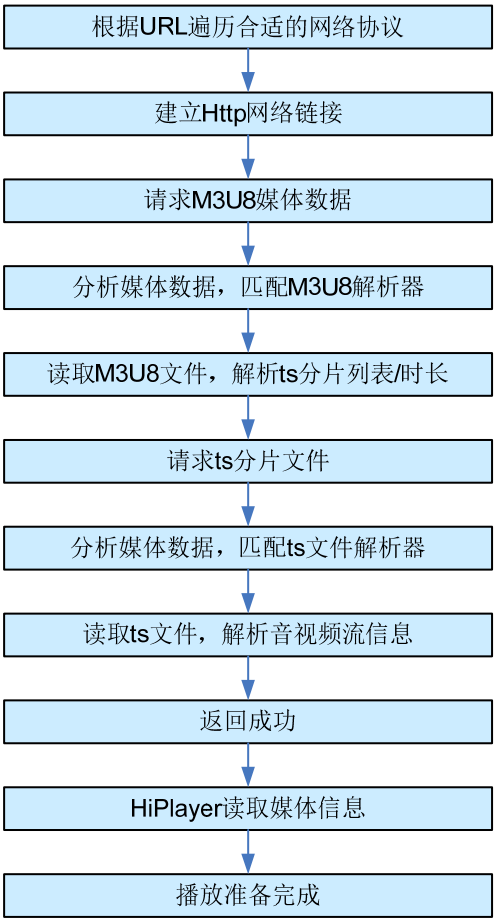
- prepareAsync

  创建 HiPlayer 对象，如果创建失败，则播放失败，创建成功，则将用户设置的一些网络播放参数传递给 HiPlayer，同时发送消息触发 doPrepare 操作。
  prepareAsync 为异步操作，调用到的 HiPlayer 的接口为 HI_SVR_PLAYER_Create。

- doPrepare

  播放准备，该操作主要动作是打开媒体文件，分析出媒体文件中流信息，如音视频流个数、音视频 Codec 类型等，也是最耗时的一个操作，调用的 HiPlayer 接口为 HI_SVR_PLAYER_SetMedia，HiPlayer 调用的解析器接口为 fmt_find、fmt_open、fmt_find_stream、fmt_getinfo（接口定义参考 hi_svr_format.h）。针对网络流媒体，如 HLS，SetMedia 详细步骤如图 4-6 所示。

图4-6 SetMedia 详细步骤



说明

- 针对 HLS 直播流，无时长，在播放过程中会定时刷新 M3U8 文件，不支持 seek 操作，对于点播流，媒体文件播放时长等于分片时长总和，播放过程中支持 seek 操作，从离 seek 时间点最近的分片开始播放。
- 针对 TS 直播流，无时长，不支持 seek 操作，对于点播流，通过获取文件头、文件尾时间戳，两者相减得到文件时长，seek 操作通过 seek 时间点、文件时长及文件大小计算出偏移位置，并从偏移位置开始播放。

- doStart

  启动播放，步骤如下：

2. HiPlayer 内部会根据从解析器中获取的媒体信息，初始化音视频解码器。

   如果出现错误，播放器会根据结果上报 HI_SVR_PLAYER_ERROR_PLAY_FAIL、HI_SVR_PLAYER_ERROR_AUD_PLAY_FAIL、HI_SVR_PLAYER_ERROR_VID_PLAY_FAIL 事件，音视频通道启动成功（只要音视频有一个通道启动成功就认为可以开始播放），HiPlayer 状态从 STOP 转换到 PLAY 状态。

3. HiPlayer 开始调用解析器 fmt_read、fmt_free（接口定义参考 hi_svr_format.h 头文件）接口读取音视频帧（frame）数据，并将帧数据送给 AVPlay 解码、播放。
4. 第一帧画面输出，HiPlayer 会上报 HI_SVR_PLAYER_EVENT_FIRST_FRAME_TIME 事件，通知 HiMediaPlayer 第一帧画面已经输出了。

   所以统计起播耗时，直接统计从调用 setDataSource 到上报 HI_SVR_PLAYER_EVENT_FIRST_FRAME_TIME 事件的时间就可以准确得到起播耗时。

   针对网络播放，存在网络状态不足以流畅播放媒体文件的情况，为了防止还未开始播放就进入网络缓冲，HiPlayer 会确保先播放一小段画面，然后再判断当前的网络状态。

   针对 HLS 直播场景，如果出现网络状态异常（如网络断开、请求数据超时等），并且在异常的情况下，客户端未及时取到分片文件，则在网络恢复后，播放器会取当前服务器最后保留的分片，可能会出现画面跳跃的情况，即网络异常，分片信息丢失。点播场景下不会有此现象。

   如图 4-7 直播场景下的列表，左边为时间点 1 分片信息，右边为时间点 2 分片信息，时间点 2-时间点 1≈100s，在这 100s 时间内网络断开一直未恢复，则在到达时间点 2 时，播放器直接从“20121026T025514-174321.ts”开始请求播放。

图4-7 HLS 直播 M3U8 列表 sample

#EXTM3U #EXT-X-VERSION:3 #EXT-X-TARGETDURATION:11 #EXT-X-MEDIA-SEQUENCE:174310 #EXTINF:10, 20121026T025514-174311.ts #EXTINF:10, 20121026T025514-174312.ts #EXTINF:10, 20121026T025514-174313.ts #EXTINF:10, 20121026T025514-174314.ts #EXTINF:10, 20121026T025514-174315.ts	#EXTM3U #EXT-X-VERSION:3 #EXT-X-TARGETDURATION:11 #EXT-X-MEDIA-SEQUENCE:174320 #EXTINF:10, 20121026T025514-174321.ts #EXTINF:10, 20121026T025514-174322.ts #EXTINF:10, 20121026T025514-174323.ts #EXTINF:10, 20121026T025514-174324.ts #EXTINF:10, 20121026T025514-174325.ts

说明

播放过程中，HiPlayer 从解析器中读取的数据单位为帧（frame），必须包含帧长度(length)、时间戳（pts）、流索引(streamidx)、是否为关键帧(keyflag)）信息，解析器在解析原始码流时，需要进行组帧操作，比如 MP4 封装 H264 视频，文件容器封装的单位可能是 NAL，那解析器在解析视频流时，必须将 NAL 组成 frame。

- seekTo

  seek 到指定时间点播放，HiPlayer 调用解析器 fmt_seek_pts 接口，传入流索引、seek 时间点、seek 方向信息，告知解析器跳到时间点对应的位置，并从该位置开始解析帧数据。解析器 fmt_seek_pts 函数返回成功后，HiPlayer 会将 AVPlay 音视频缓冲清除掉，确保播放的是 seek 之后读取的码流数据。

针对 HLS 网络点播，seek 操作会先调用到 HLS 解析器，HLS 解析器根据 seek 时间点、分片信息找到相近的分片，关闭掉老的网络链接，建立新的网络链接，请求新的分片数据，并从该分片开始解析媒体帧数据，如果 seek 时间点大于 HLS 时长，则直接退出播放。

seek 操作为异步操作，应用下发 seek 命令后，MediaPlayer 通过 MEDIA_SEEK_COMPLETE 事件通知应用 seek 是否完成。

- reset

  复位播放器，该操作会销毁 MediaPlayer 实例，对应的 HiMediaPlayer 操作是销毁 HiPlayer，HiPlayer 会通知解析器终止当前操作，并快速退出。

4.4.2.3 开源代码使用说明

开源代码目前包含 HLS、HTTP、TCP 三个协议，全部基于 ffmpeg 框架开发，提供 HLS 播放功能。在播放启动时，通过 ffmpeg register 接口将这三个协议注册到 ffmpeg demux/protocol 列表中，当播放 HLS、HTTP 网络流媒体文件时，会遍历到该协议并使用该协议建立网络链接和 HLS 播放。

目录结构

开源代码目录结构如下：

```
device/hisilicon/bigfish/external/libformat_open/
├── Android.mk
├── hls.c
├── hls_key_decrypt.c
├── hls_read.c
├── http.c
├── svr_allformat.c
├── svr_iso_dec.c
├── svr_udf_protocol.c
└── tcp.c
```

开源代码存放路径为：device/hisilicon/bigfish/external/libformat_open，hls.c、hls_key_decrypt.c、hls_read.c 实现 HLS 码流播放功能，http.c、tcp.c 实现 HTTP 协议客户端网传功能，svr_allformat.c 实现将 HLS、HTTP、TCP 注册到 ffmpeg demux/protocol 列表中的功能，svr_iso_dec.c、svr_udf_protocol.c 不需关注，不需做任何修改。

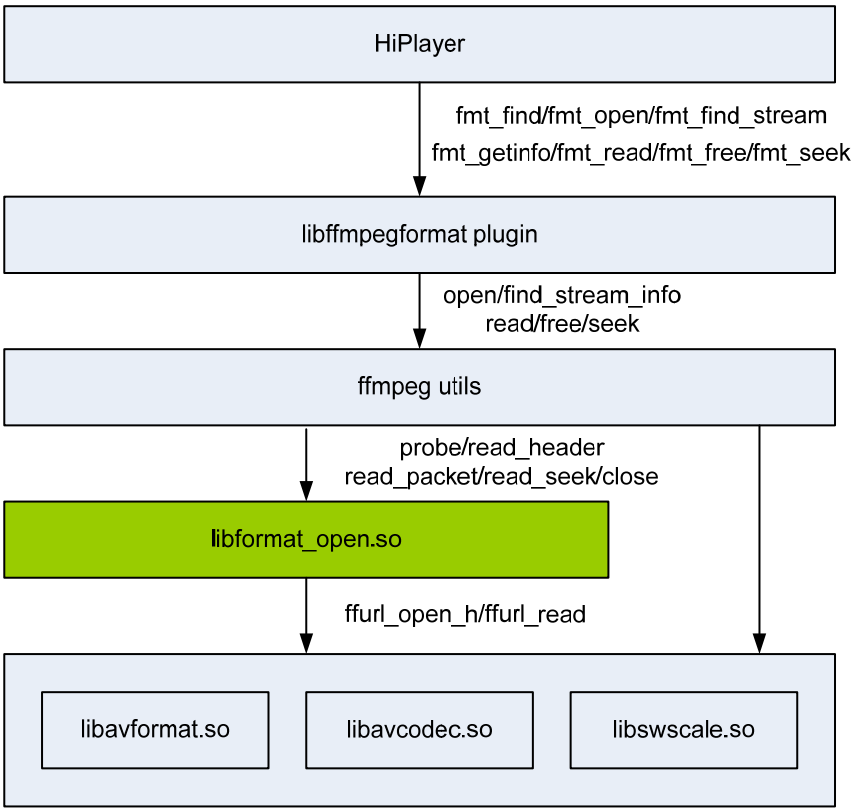
编译方式

libformat_open 为独立库，修改后可以在 libformat_open 目录下直接 mm 编译生成 libformat_open.so，通过 adb 方式将库 push 到单板/system/lib 目录，重启 media 服务，就可以测试修改。

HLS 框架

HLS 框架中 libformat_open plugin 与 ffmpeg 及 libffmpegformat plugin 关系如图 4-8 所示。

图4-8 libformat_open plugin 与 ffmpeg 及 libffmpegformat plugin 关系



libformat_open 实现依赖 libavformat、libavcodec、libswscale 库，ffmpeg utils 为 ffmpeg api 接口，提供 seek、packet 组帧功能。

如图 4-8，当播放一个 HLS 网络流时，步骤如下：

步骤 1 HiPlayer 会调用 libffmpegformat plugin 打开媒体文件；

步骤 2 libffmpegformat plugin 调用 ffmpeg utils 查找合适的协议、解析器，如 libformat_open.so 中的 HLS、HTTP，然后建立网络链接，请求媒体文件；

步骤 3 ffmpeg utils 调用 libformat_open HLS 协议解析 M3U8 文件，获取 TS 分片信息；

步骤 4 libformat_open 建立新的链接，请求 TS 分片，并调用 libavformat 解析 TS 数据；

步骤 5 播放过程中，libffmpegformat plugin 调用 ffmpeg utils 读取音视频帧数据，ffmpeg utils 调用 libformat_open 读取 packet 数据并进行组帧。

----结束

HLS 内部主要函数

HLS 内部主要函数说明如下：

- HLS_probe

M3U8 格式识别函数，判断输入的数据内容是否符合 M3U8 文件格式，并返回识别打分，此函数返回的打分决定后续播放是否使用 HLS 协议。

- HLS_read_header

  调用 M3U8 文件完成 HLS 流分析，得到 HLS 码率信息及每种码率对应的 TS 分片信息，并识别出直播、点播流，得到所有分片信息后，该函数默认打开码率最高的分片，并调用 libavformat 解析出媒体信息。

- HLS_read_packet

  帧数据读取函数，ffmpeg utils 调用该函数读取 packet 数据并进行组帧，该函数调用 libavformat 读取 packet 数据，在读数据前会先判断用户是否指定码率播放，如果是，则关闭当前码率文件，重新打开新的码率文件，并读取新的码率文件数据。

- HLS_read_seek

  跳到指定时间点分片，该函数根据用户传入的时间点，遍历 HLS 列表，找到离时间点最近的分片，并从该分片开始播放。直播流不支持 seek，直接返回失败，对于点播流，如果 seek 时间点超过 HLS 码流总时长，则直接结束播放。

- HLS_close

  退出播放，关闭文件请求，关闭网络链接，释放分配的资源。

- hlsParseM3U8

  按行读取 M3U8 文件，并按 HLS 规范解析 M3U8 文件，如分片 URL、TargetDuration、分片时长、是否加密码流、ENDLIST 等。

- hlsReadDataCb

  数据回调函数，当调用 HLS_read_packet 函数时会触发该函数调用，该函数逻辑比较复杂，涉及到码率切换，新的分片请求及直播场景下 M3U8 文件刷新，HLS 功能主要逻辑在此函数中实现。正常情况下该函数返回从 HTTP 协议中读取到的原始媒体数据，供 libavformat 解析器解析。

- hlsOpenSegment

  请求新的分片数据，针对 HLS AES-128 加密流，该函数考虑到客户扩展 AES-128 秘钥获取途径，首先会调用秘钥获取适配层 CLIENT_Init/ CLIENT_GetKey 函数获取秘钥，如果获取失败，再走通用路径获取。

- hlsReadLine

  针对 M3U8 文本文件，返回文本文件中的一行，方便 M3U8 文件解析。

- hlsNewHlsStream

  创建一个新的 HLS 流，针对 HLS 多码率的场景下。

HLS 内部数据结构

HLS 内部数据结构属性如表 4-1 所示。

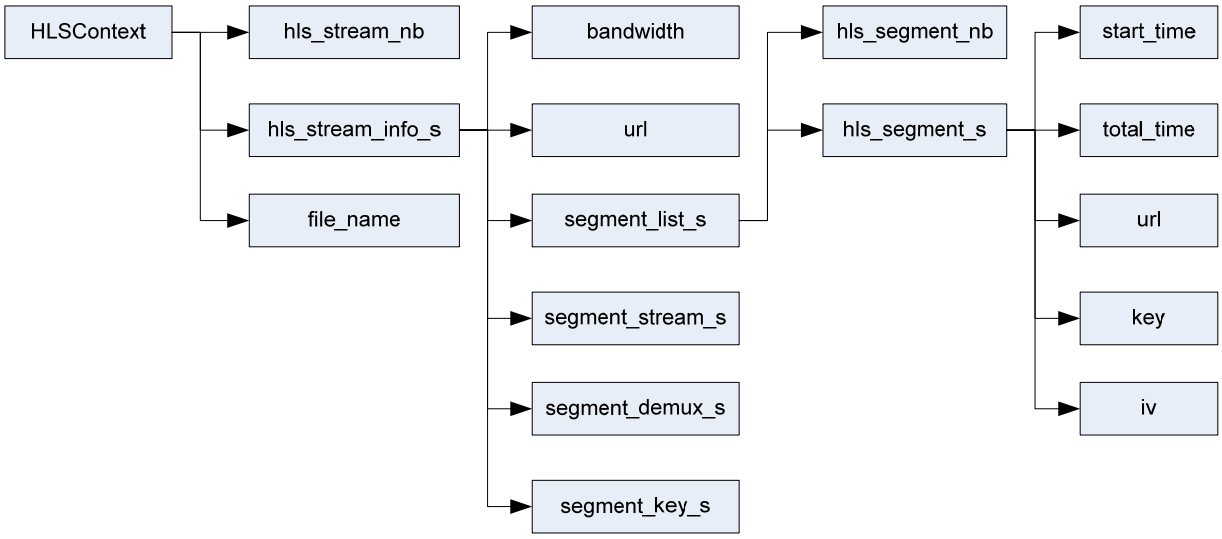
表4-1 HLS 内部数据结构说明

属性名称	含义
HLSContext	HLS 协议内部数据结构，其他解析器、协议不可见
hls_stream_nb	HLS 流中包含的码率，比如 3 种码率，则该属性值为 3

属性名称	含义
hls_stream_info_s	HLS 流中每种码率对应的分片、带宽等信息，即二级 M3U8 个数
file_name	一级 M3U8 文件名称+路径，即 URL
bandwidth	码率信息
url	二级 M3U8 文件 URL
segment_list_s	二级 M3U8 文件中包含的分片信息
segment_stream_s	当前打开的分片 IO
segment_demux_s	当前打开的分片解析器信息
segment_key_s	当前打开的分片对应的秘钥 IO，仅 AES-128 加密流有效
hls_segment_nb	某种码率中对应的分片个数
hls_segment_s	某种码率中对应的分片信息
start_time	分片起始时间，等于前面分片时长总和
total_time	分片总时长，从 M3U8 文件中解析得到
url	分片 URL
key	该分片解密秘钥 URL
iv	该分片解密秘钥 IV 向量

HLS 内部数据结构及包含关系如图 4-9 所示，从左往右依次为包含关系。

图4-9 HLS 数据结构



HLS sample

sample 目录结构如下：

```
hls_sample/
├── hisilicon_key.txt
└── hls_multi_rate
    ├── main_playlist.m3u8
    ├── stream-128000
    │   ├── stream-128000-1.ts
    │   ├── stream-128000-2.ts
    │   ├── stream-128000-3.ts
    │   └── stream-128000.m3u8
    ├── stream-1708000
    │   ├── stream-1708000-1.ts
    │   ├── stream-1708000-2.ts
    │   ├── stream-1708000-3.ts
    │   └── stream-1708000.m3u8
    └── stream-764000
        ├── stream-764000-1.ts
        ├── stream-764000-2.ts
        ├── stream-764000-3.ts
        └── stream-764000.m3u8
```

1 级 M3U8 main_playlist.m3u8 文件内容如下：

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=128000
stream-128000/stream-128000.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=764000
stream-764000/stream-764000.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=1708000
stream-1708000/stream-1708000.m3u8
```

可以看出：

- 该 HLS 流有 3 种码率，即有 3 个二级 M3U8，码率分别为：
  - 128000
  - 764000
  - 1708000bps
- 对应的 hls_stream_nb 值为 3，有三组 hls_stream_info_s 信息
- file_name 等于码流根目录+hls_sample/hls_multi_rate/main_playlist.m3u8，

如：在某个机器上装一个 apache 服务器，将 hls_sample 放在 apache 服务器根目录，则 file_name 应该等于：

http://10.67.225.7/htdocs/hls_sample/hls_multi_rate/main_playlist.m3u8

二级 M3U8 文件内容如下，根据二级 M3U8 内容依次可以解析出 HLS 后续几个数据结构中的属性值。

```
#EXTM3U
#EXT-X-KEY:METHOD=AES-128,
URI="https://10.67.225.27/htdocs/hls_sample/hisilicon_key.txt",IV=0x00000
00000000000000000000000000
#EXT-X-TARGETDURATION:10
#EXT-X-VERSION:3
#EXTINF:10,
stream-128000-1.ts
#EXTINF:10,
stream-128000-2.ts
#EXTINF:10,
stream-128000-3.ts
#EXT-X-ENDLIST

#EXTM3U
#EXT-X-KEY:METHOD=AES-128,
URI="https://10.67.225.27/htdocs/hls_sample/hisilicon_key.txt",IV=0x00000
00000000000000000000000000
#EXT-X-TARGETDURATION:10
#EXT-X-VERSION:3
#EXTINF:10,
stream-764000-1.ts
#EXTINF:10,
stream-764000-2.ts
#EXTINF:10,
stream-764000-3.ts
#EXT-X-ENDLIST

#EXTM3U
#EXT-X-KEY:METHOD=AES-128,
URI="https://10.67.225.27/htdocs/hls_sample/hisilicon_key.txt",IV=0x00000
00000000000000000000000000
#EXT-X-TARGETDURATION:10
#EXT-X-VERSION:3
#EXTINF:10,
stream-1708000-1.ts
#EXTINF:10,
stream-1708000-2.ts
```

```
#EXTINF:10,
stream-1708000-3.ts
#EXT-X-ENDLIST
```

4.4.2.4 HLS 定制化说明

以中国移动视频基地视频直播为例，除了直播、点播功能外，还定义了时移的功能（比如当前时间为 12:00，如果用户想看早 08:00 点新闻，则可以通过时移的方式观看）。由于 HLS 并没有定义时移规范，视频基地通过在 URL 中增加关键字段并定义规范来实现时移业务。

时移功能定制

以下 URL 为视频基地时移 URL：

```
“http://ips.itv.cmvideo.cn/140_0_3736680_0.m3u8?id=3736680&scId=&isChannel=true&time=300&serviceRegionIds=140&codec=ALL&quality=100&vSiteCode=&offset=0&livemode=2&starttime=20141024T095821.00Z”
```

其中：

- “livemode”等于 2 为时移，等于 1 为直播；
- “starttime”为时移开始时间，“20141024T095821”为 2014 年 10 月 24 日 9 点 58 分 21 秒。

由于时移节目是有时长的，比如两小时，视频基地规范要求，如果时移播放完了，需要接着播放下一段时移码流，而播放下一段时移码流，需要重新组织新的 URL，需要更新第一次传入的 URL 中的“starttime”字段，更新的方法为：

新的“starttime”=第一次传入的 URL 中的“starttime”+已经播放的分片总和

还是以上述 URL 为例，假设按此 URL 获取到的分片时长总和为 1 小时 50 分，当下载完这 1 小时 50 分的分片后，URL 中的“starttime”应该刷新为：

20141024T095821.00Z + 110 分 = 20141024T114821.00Z

并使用新的 URL 刷新 M3U8 列表。

针对视频基地时移功能，需要在第一次获取的分片下载完后，计算新的 URL，并使用新的 URL 请求分片。

在开放的 hls.c 文件中：

- 直播场景下，M3U8 列表刷新在 hlsReadDataCb 函数中执行，可以看到函数有“if (v->seg_list.hls_seg_cur >= v->seg_list.hls_seg_start + v->seg_list.hls_segment_nb)”判断，即判断是否所有分片都下载完了。
- 时移场景下，第一次获取的分片下载完会执行到该分支，添加 URL 刷新可以在此分支中完成。标准 HLS 直播流，是在已有的分片下载完或者达到一定时间间隔后，使用初始的播放 URL 刷新 M3U8 列表。

视频基地时移场景下是在分片下载完后先要更新 URL，再使用新的 URL 请求新的时移分片，请参考 hls.c 中 hlsReadDataCb 函数的实现。

HLS 二次开发

HLS 规范也在不断演进，加上 HLS 应用广泛，比较易于通过增加私有定义来扩展新的功能，hls.c 实现独立，也比较容易修改来适配私有定义的功能。HLS 内部定义了一个私有数据结构 HLSContext（详见 avformat.h 头文件，目录：device/hisilicon/bigfish/external/ffmpeg/libavformat），如果修改需要增加新的属性，可以在 HLSContext 末尾定义，不允许修改其他数据结构或在已定义的数据结构中间增加。

4.4.3 SmoothStreaming

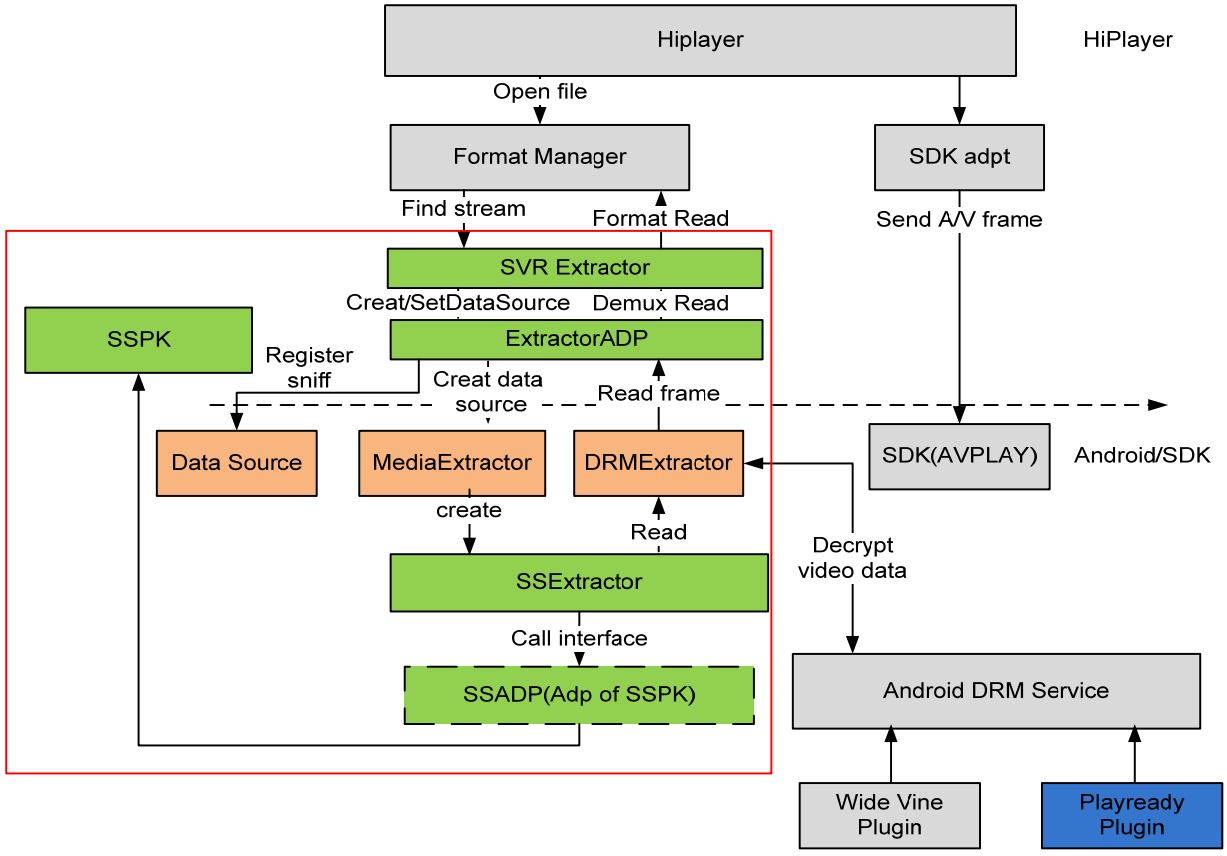
场景说明

SSPK(SmoothStreaming Porting kit)是微软提供的一种 Adaptive Streaming 方案，目的是使码流质量能够动态的适应用户的网络状况，使用户达到最佳的观看体验。

工作流程

海思目前方案的软件整体架构如图 4-4 所示。其中 libextractor_demux.so 为 SmoothStreaming 解析器，其内部工作流程如图 4-10 所示。

图4-10 SmoothStreaming 内部模块工作流程图



其中，红框内的绿色部分即为 libextractor_demux.so 解析器。

- SVR Extractor 和 ExtractorADP

  此部分为开源代码，具体路径为：
  device\hisilicon\bigfish\hidolphin\component\plugin_extractor，其目录结构为：

```
├── Android.mk
├── demux
│   └── svr_extractor.c
├── extrator_adp
│   ├── svr_extrator_adp.cpp
│   ├── svr_extrator_adp.h
│   └── svr_extrator_intf.h
├── hi_type.h
├── project.config
└── sample_test.c
```

  其中 SVR Extractor 对应的源文件为 svr_extractor.c，其作用是对解析器接口进行定义，对接 Hiplayer Format Manager.

  ExtractorADP 模块对应的源文件是 svr_extractor_adp.cpp，其功能如下：

  - 给上层模块即 SVR Extractor 提供 C 接口；
  - 创建数据源、读数据等操作。（此部分为开源代码，不作具体描述，请参考 svr_extrator_adp.cpp）

- SSExtractor：

  此模块为 Android 原生对 SSPK 的适配，主要流程如下：

  - 创建 SSPK 解析器

```
s32Ret = HI_SVR_SMOOTHSTREAMING_Create(&struInitParam,
&mHiSSPlayer);
```

  - 打开文件源

```
s32Ret = HI_SVR_SMOOTHSTREAMING_Open(mHiSSPlayer, &mMediaParam);
```

  - 获取流的加密属性

```
s32Ret = HI_SVR_SMOOTHSTREAMING_GetParam(mHiSSPlayer,
HI_SVR_SMOOTHSTREAMING_ATTR_DRM_DOMAIN,&stDrmHeaderInfo);
```

  - 启动 SmoothStreaming 码流解析

```
s32Ret = HI_SVR_SMOOTHSTREAMING_Play(mHiSSPlayer);
```

  - 创建数据读取线程

```
createThreadEtc(SSExtractor::sched_thread, this,
"ssextrator_thread", ANDROID_PRIORITY_DEFAULT, 0, NULL);
```

  - 读取 SSPK 解析出来的数据

```
status = HI_SVR_SMOOTHSTREAMING_ReadFrame(mHiSSPlayer,
&pstFmtFrame);
```

- SSADP

是 SSPK 的适配层，对外提供接口。具体接口描述请参考 hi_svr_smoothstreaming.h 和 hi_svr_smoothstreaming_adp.h。

- SSPK

  SSPK(SmoothStreaming Porting kit)由微软提供。

- DRMExtractor

  Android 原生，其功能是进行数据传输、调用解密接口对加密流进行解密。其具体操作如下：

  - 获取原始数据

    ```
    originalMediaSource = mOriginalExtractor->getTrack(index);
    ```

  - 判断是否加密流

    否：`return originalMediaSource;`

    是：`MediaSource = new DRMSource(originalMediaSource,…);`

    ```
    return MediaSource;
    ```

  - 读数据

    对于清流，将直接读取数据。

    加密流数据的读取：

    读取原始数据：mOriginalMediaSource->read(buffer, options))

    调用解密接口对数据进行解密 mDrmManagerClient->decrypt(...)

- Playready

  解密库，相关内容请参考《Android PlayReady 开发指南》。

注意事项

- 资费说明

  最终用户产品供应商必须向 Microsoft 购买 SSPK 产品授权，其具体资费请参考：http://www.microsoft.com/en-us/mediaplatform/sspk.aspx.

- 编译说明

  编译之前需确保 device/hisilicon/Hi3798MV100/device.mk 中：

  ```
  PRODUCT_PROPERTY_FOR_DRM_CLIENT=true
  PRODUCT_PROPERTY_FOR_DRM_SERVICE=true。
  ```

- 功能支持

  - 支持多码率自适应切换

    支持多码率自适应切换，可以根据当前的网络带宽，选择播放不同码率的视频。最小切换时间，一个分片时间（一般 2s 左右，具体以服务器提供码流分片时间为准）。切换视频质量的等级由服务器提供，例如微软官方 SmoothStreaming 码流服务器(http://playready.directtaps.net/smoothstreaming/)提供如下。

    ```
    http://playready.directtaps.net/smoothstreaming/SSWSS720H264/SuperSpeedway_720_230.ismv
    http://playready.directtaps.net/smoothstreaming/SSWSS720H264/SuperSpeedway_720_331.ismv
    http://playready.directtaps.net/smoothstreaming/SSWSS720H264/SuperSpeedway_720_477.ismv
    http://playready.directtaps.net/smoothstreaming/SSWSS720H264/SuperSpeedway_720_688.ismv
    http://playready.directtaps.net/smoothstreaming/SSWSS720H264/SuperSpeedway_720_991.ismv
    http://playready.directtaps.net/smoothstreaming/SSWSS720H264/SuperSpeedway_720_1427.ismv
    ```

http://playready.directtaps.net/smoothstreaming/SSWSS720H264/SuperSpeedway_720_2056.ismv
http://playready.directtaps.net/smoothstreaming/SSWSS720H264/SuperSpeedway_720_2962.ismv

根据当前网络情况，从上面列出（服务器端提供的内容相同，质量（码率）不同的码流）的码流中匹配符合当前网络的最佳码流。

- 支持格式

  目前音视频编码格式只支持 VC1/WMA、H264/AAC

- 播放控制如表 4-2 所示。

表4-2 播放控制

功能	播放	暂停	快进	快退	Seek	停止
网络直播	√	√	×	×	×	√
网络点播	√	√	√	√	√	√

说明：√表示支持，×表示不支持

- 支持软 PlayReady 加密流播放（playready 相关的内容请参考《playready 开发指南》）

- 功能接口

  其功能接口请参考 Android 原生 mediaplayer 功能接口。

- 测试说明

  测试准备：

  - 确保 getprop drm.service.enabled 和 getprop drm.client.enabled 都为 true
  - 确认 data/playready/prpd 目录下存在如下证书：
    - bgroupcert.dat
    - zgpriv.dat
    - devcerttemplate.dat
    - priv.dat
  - 确认 system/app 中有 DrmAssist-Recommended.apk
  - 确认 playready 库存在（参考 Android PlayReady 开发指南 3.4.2）
  - 打开应用 DRM（桌面图标和名字均为 DRM），选择 UrlPlay，默认会打开：

    http://playready.directtaps.net/smoothstreaming/
  - 选择需要播放的码流，点击对应的 manifest 文件即可播放

    此网页(http://playready.directtaps.net/smoothstreaming/)上没有的码流，可以使用命令播放（由于 DrmAssist-Recommended.apk 只是一个 demo，有些功能 apk 可能尚未实现，如 seek 和倍速播放，建议非加密流直接使用命令播放，加密流先使用 apk 获取 license，再使用命令播放），如：

    ```
    am start -n
    com.hisilicon.android.videoplayer/.activity.VideoActivity -d
    http://116.199.4.50/CDN/101_GZTV.isml/manifest
    ```

示例

步骤 1　创建播放器

```
mVideoView = (VideoView) findViewById(R.id.videoView1);
```

步骤 2　设置错误监听器

```
mVideoView.setOnErrorListener(this);
```

步骤 3　设置媒体控制器

```
mVideoView.setMediaController(ctlr);
```

步骤 4　设置播放路径

```
mVideoView.setVideoPath(path);
```

步骤 5　获取权限（具体流程参考《Android PlayReady 开发指南》第 4 章 开发应用）

```
response = drmClient.acquireDrmInfo(request);
```

步骤 6　启动播放

```
mVideoView.start();
```

----结束

5 HiDLNA

5.1 概述

DLNA（数字生活网络联盟 Digital Living Network Alliance）旨在实现家庭内部的设备互通和媒体交互，HiDLNA 是 DLNA 协议的标准实现。

HiDLNA 组件包括以下三项独立的应用：

- DMS(Digital Media Share)：数字媒体服务，用于机顶盒上媒体文件的共享，使得 DMP 和 DMC(数字媒体控制点 Digital Media Controller)设备可以访问，并可以选择其分享的文件进行播放；
- DMR(Digital Media Render)：数字媒体呈现，用于提供媒体控制和连接管理服务，以便接收 DMC 的推送媒体和播放控制；
- DMP（Digital Media Player）：数字媒体播放器，用于搜索发现媒体共享服务器，浏览媒体资源文件并可选择文件进行播放。

DMC 是 DLNA 控制点，也是 DLNA 的一个标准实现，主要用于搜索发现 DMS 和 DMR，并可以访问 DMS 上的资源以及选择控制 DMR 进行播放，DMC 不在本文档的描述范围。本章描述了 HiDLNA 组件在 Android 和 Linux 环境下的接口使用方法，用于指导在海思芯片平台上进行基于 HiDLNA 组件功能的开发，其中第 5.2 章分析了 Android 的接口调用，第 5.3 章分析了 Linux 环境下的接口调用。

5.2 Android AIDL 接口的使用与定义

AIDL（Android Interface definition language）Android 接口描述语言，是一种 android 内部进程间通信 IPC（Interprocess Communication）接口的描述语言。

在 Android 平台上，每个应用程序都运行在自己的进程空间，一个进程通常不能访问另一个进程的内存空间，而在开发 Android 应用中，经常会在不同的进程间传递对象，因此为了实现 android 内部进程间通信，Android 提供了 AIDL 接口机制，用于约束两个进程间的通信规则，实现 Android 设备上两个进程间通信。

使用 AIDL 接口，进程之间的通信信息，首先会被转换成 AIDL 协议消息，然后发送给对方，对方收到 AIDL 协议消息后再转换成相应的对象，由于进程之间的通信信息需要双方转换，所以 Android 采用代理类在背后实现了信息的双向转换，代理类由 Android 编译器生成，对开发人员来说是透明的。

使用的 AIDL 步骤大致分如下四步：

步骤 1 创建自己的 AIDL 文件，该文件定义的方法和域可用于客户端，其可以引用其他 AIDL 文件中定义的接口；

步骤 2 将创建的 AIDL 文件添加到编译器中；

步骤 3 实现所定义 AIDL 接口中的内部抽象类；

步骤 4 将接口提供给客户端使用，告诉在客户端如何调用服务端的 AIDL 描述的接口对象。

Android 官方对 AIDL 的使用步骤作了详细描述，详细可参阅 `http://developer.android.com/guide/components/aidl.html`

在使用 HiDLNA AIDL 接口前，我们先介绍一下 Android 的 Service（服务）概念：

Service：是一个没有界面且能长时间运行于后台的应用组件，其它应用的组件可以启动一个服务运行于后台，即使用户切换到另一个应用也会继续运行．另外，一个组件可以绑定到一个 service 来进行交互，即使这个交互是进程间通讯也没问题。

HiDLNA 中的 DMS 和 DMR 在 Android 中是以 Service 的形式存在的，并提供了 AIDL 接口，本文接下来将描述 HiDLNA 中的 AIDL 使用方法，使用前请先了解 HiDLNA 提供的 AIDL 接口定义。

----结束

5.2.2 HiDLNA Android AIDL 接口的使用方法

5.2.2.1 概述

下面以 HiDLNA 中 DMR 提供的 aidl 文件 IDMRServer.aidl 为例说明 HiDLNA 中 AIDL 接口的使用方法，主要步骤如下：

步骤 1 创建一个 Service，在 Service 里将 DMR 注册到系统服务中，调用如下方法：

```
IDMRServer.Stub dmrBinder = new DMRBinder(Context);
ServiceManager.addService("dmr", dmrBinder);
```

步骤 2 使用第一步注册的服务，调用如下方法：

```
IDMRServer dmrServer =
IDMRServer.Stub.asInterface(ServiceManager.getService("dmr"));
```

步骤 3 启动 DMR 服务，调用如下方法：

```
dmrServer.create()
```

----结束

5.2.2.2 DMP AIDL 接口使用方法

DMP 提供的类存放在包 com.hisilicon.dlna.file 下，主要使用步骤如下：

步骤 1 获取设备列表。调用如下方法：

```
List<DMSDevice> searchedDeviceList = DMSDevice.searchDMSDevices()
```

该方法返回值 List<DMSDevice>，即为获取到的 DMS 设备列表。

步骤 2 获取 DMS 共享的一级目录列表，调用如下方法：

```
DMSFile[] files = DMSFile.getTopFiles((DMSDevice)device)
```

该方法返回值 DMSFile[]，即为获取到 DMS 一级目录文件列表。

步骤 3 获取子目录，调用如下方法：

```
DMSFile[] files = ((DMSFile)focusFile).listFiles()
```

该方法返回值 DMSFile[]，即为获取到 DMS 本级子目录文件列表。

----结束

说明

获取目录列表是分批获取的，即如果一个目录下有 2000 个子目录，则 listFiles 第一次返回一批（200 个左右），第二次获取下一批（201~400），如果此时想从第一个获取，则调用如下方法：DMSFile.reset()。

5.2.2.3 DMR AIDL 接口使用方法

DMR 提供的类存放在包 com.hisilicon.dlna.file 下，主要使用步骤如下：

步骤 1 创建一个 Service，在 Service 里将 DMR 注册到系统服务中，调用如下方法：

```
IDMRServer.Stub dmrBinder = new DMRBinder(Context);
ServiceManager.addService("dmr", dmrBinder);
```

该方法主要启动 DMR 服务，并将 DMR 的服务注册到系统服务中，以方便在客户端进程中能使用该服务，其中参数“dmr”指定要启动一个 DMR。

步骤 2 使用第一步注册的服务，调用如下方法：

```
IDMRServer dmrServer =
IDMRServer.Stub.asInterface(ServiceManager.getService("dmr"));
```

通过获取 DMR 服务，进一步来使用 DMR 提供的操作方法，其中参数“dmr”指定要获取 DMR 的服务。

步骤 3 控制 DMR 操作，调用如下方法：

1. 启动 DMR 服务，调用如下方法：
   dmrServer.create(“dmrName”)
   其中“dmrName”为 DMR 的名称
2. 关闭 DMR 服务，调用如下方法：
   dmrServer.destroy();
3. 播放器暂停状态（pause）通知，通过如下回调实现：
   dmrServer.getDMRCallback().pauseNotify()

当播放器由播放转为暂停时，该方法会将暂停状态通知给客户端。

----结束

5.2.2.4 DMS AIDL 接口使用方法

DMS 提供的类存放在包 com.hisilicon.dlna.file 下，主要使用步骤如下：

步骤 1 创建一个 Service，在 Service 里将 DMS 注册到系统服务中，调用如下方法：

```
IDMSServer.Stub dmsBinder = new DMSBinder(Context);
ServiceManager.addService("dms", dmsBinder);
```

该方法主要启动 DMS 服务，并将 DMS 的服务注册到系统服务中，以方便在客户端进程中能使用该服务，其中参数“dms”指定要启动一个 DMS。

步骤 2 使用第一步注册的服务，调用如下方法：

```
IDMSServer dmsServer =
IDMSServer.Stub.asInterface(ServiceManager.getService("dms"));
```

通过获取 DMS 服务，进一步来使用 DMS 提供的操作方法，其中参数“dmr”指定要获取 DMS 的服务。

步骤 3 控制 DMS 操作，调用如下方法：

1. 启动 DMS,调用方法如下:

   dmsBinder.create(“dmsName”, “/data”);

   其中第一个参数“dmsName”为设置的 DMS 名称，第二个参数 “/data”为设置的 DMS 共享的目录。

2. 添加共享目录，调用如下方法：

   dmsBinder.addShareDir(“dmsName”, “/sdcard/”);

   其中第一个参数“dmsName”为设置的 DMS 名称，第二个参数 ““/sdcard/”为添加的 DMS 共享的目录。

----结束

5.2.3 HiDLNA Android AIDL 接口的定义

5.2.3.1 HiDLNA AIDL API 功能简介

HiDLNA 提供的 AIDL API，主要用于注册和启动 DMS、DMR 服务，开启 DMP 搜索设备和浏览文件功能，实现在海思芯片平台上 DLNA 功能。

5.2.3.2 HiDLNA AIDL API 函数参考

HiDLNA 提供如下 AIDL API：

- IDMRPlayerController：构造函数，获取 DMR 播放器的信息。
- getDuration：获取 DMR 播放器的总时间。
- getCurrentPosition：获取 DMR 播放器的当前时间。

- IDMRCallback：构造函数，DMR 播放器状态回调。
- playNotify：播放器播放状态回调。
- pauseNotify：播放器暂停状态回调。
- seekNotify：接口保留。
- stopNotify：播放器停止状态回调。
- setCurTime：接口保留。
- getCurTime：接口保留。
- setAllTime：接口保留。
- getAllTime：接口保留。
- IDMRServer：构造函数，DMR 服务控制接口。
- create：打开 DMR、DMS 设备接口。
- destroy：销毁 DMR、DMS 接口。
- setName：第一次启动时设置 DMR 名称接口。
- setDeviceName：启动后设置 DMR、DMS 名称。
- startActivity：启动启动一个应用（Activity）接口。
- sendBroadcast：发送一个广播。
- IDMRCallback：获取 DMR 回调接口。
- registerPlayerController：注册播放器控制接口。
- unregisterPlayerController：注销播放器控制接口。
- IDMRPlayerController：获取播放器控制接口。
- IDMSServer：构造函数，DMS 服务控制接口。
- addShareDir：DMS 添加共享目录。
- delShareDir：DMS 删除共享目录。

interface IDMRPlayerController

【目的】

构造函数，用于获取 DMR 播放器的信息，包括播放的总时间和当前时间。

【语法】

```
interface IDMRPlayerController
{
    int getDuration();
    int getCurrentPosition();
}
```

【描述】

播放器实现此播放接口，并通过 registerPlayerController 将此接口注册到 DMRServer 里，以便客户端获取播放器的一些信息时调用。

【参数】

无

【返回值】

无

【错误码】

无

【需求】

包 com.hisilicon.dlna.file

【注意】

无

【举例】

无

int getDuration()

【目的】

构造函数 IDMRPlayerController 的一个实现，客户端通过此接口获取播放总时长。

【语法】

int getDuration();

【描述】

客户端通过此接口获取播放总时长。

【参数】

无

【返回值】

返回值	描述
大于等于 0	播放总时长，单位 ms
其他	错误

【错误码】

无

【需求】

包 com.hisilicon.dlna.file 和构造函数 IDMRPlayerController

【注意】

无

【举例】

无

int getCurrentPosition()

【目的】

构造函数 IDMRPlayerController 的一个实现，客户端通过此接口获取播放当前时长。

【语法】

```
int getCurrentPosition ();
```

【描述】

客户端通过此接口获取播放当前时长。

【参数】

无

【返回值】

返回值	描述
大于等于 0	播放当前时长，单位 ms
其他	错误

【错误码】

无

【需求】

包 com.hisilicon.dlna.file 和构造函数 IDMRPlayerController

【注意】

无

【举例】

无

interface IDMRCallback

【目的】

构造函数，播放器状态回调接口。

【语法】

```
interface IDMRCallback
```

```
{
    void playNotify();
    void pauseNotify();
    void seekNotify(int pos);
    void stopNotify();
    void setCurTime(int time);
    int getCurTime();
    void setAllTime(int time);
    int getAllTime();
}
```

【描述】

构造函数，播放器状态回调接口，播放器通过此接口在状态改变时通知客户端。

【参数】

无

【返回值】

无

【错误码】

无

【需求】

包 com.hisilicon.dlna.file

【注意】

无

【举例】

无

【举例】

无

void playNotify()

【目的】

构造函数 IDMRCallback 的一个实现，当播放器开始播放时用此接口将播放状态回报给 DMC。

【语法】

void playNotify();

【描述】

当播放器开始播放时用此接口将播放状态回报给客户端。

【参数】

无

【返回值】

无

【错误码】

无

【需求】

包 com.hisilicon.dlna.file 和构造函数 IDMRCallback

【注意】

无

【举例】

无

void pauseNotify ()

【目的】

构造函数 IDMRCallback 的一个实现，当播放器暂停时用此接口将播放器状态回报给客户端。

【语法】

void pauseNotify ();

【描述】

当播放器暂停时用此接口将播放器状态回报给客户端。

【参数】

无

【返回值】

无

【错误码】

无

【需求】

包 com.hisilicon.dlna.file 和构造函数 IDMRCallback

【注意】

无

【举例】

无

void seekNotify (int pos)

【目的】

构造函数 IDMRCallback 的一个实现，当播放器 seek 时用此接口将播放器状态回报给客户端。

【语法】

void seekNotify (int pos);

【描述】

当播放器 seek 时用此接口将播放器状态回报给客户端。

【参数】

参数名称	描述	输入/输出
pos	Seek 到的时间点	输入

【返回值】

无

【错误码】

无

【需求】

包 com.hisilicon.dlna.file 和构造函数 IDMRCallback

【注意】

本接口为预留接口，调用此接口无效。

【举例】

无

void stopNotify ()

【目的】

构造函数 IDMRCallback 的一个实现，当播放器停止时用此接口将播放器状态回报给客户端。

【语法】

void stopNotify ();

【描述】

当播放器停止时用此接口将播放器状态回报给客户端。

【参数】

无

【返回值】

无

【错误码】

无

【需求】

包 com.hisilicon.dlna.file 和构造函数 IDMRCallback

【注意】

无

【举例】

无

void setCurTime(int time)

【目的】

构造函数 IDMRCallback 的一个实现，通过此接口设置播放器的当前时间。

【语法】

void setCurTime(int time);

【描述】

通过此接口设置播放器的当前时间。

【参数】

参数名称	描述	输入/输出
time	设置播放器的当前时间	输入

【返回值】

无

【错误码】

无

【需求】

包 com.hisilicon.dlna.file 和构造函数 IDMRCallback

【注意】

本接口为预留接口，调用此接口无效。

【举例】

无

int getCurTime ()

【目的】

构造函数 IDMRCallback 的一个实现，客户端通过此接口获取播放当前时长。

【语法】

int getCurTime ();

【描述】

客户端通过此接口获取播放当前时长。

【参数】

无

【返回值】

返回值	描述
大于等于 0	播放当前时长
其他	错误

【错误码】

无

【需求】

包 com.hisilicon.dlna.file 和构造函数 IDMRCallback

【注意】

本接口为预留接口，调用此接口无效。

【举例】

无

void setAllTime (int time)

【目的】

构造函数 IDMRCallback 的一个实现，通过此接口设置播放器的总时长。

【语法】

void setAllTime (int time);

【描述】

通过此接口设置播放器的总时长。

【参数】

参数名称	描述	输入/输出
time	设置播放器的总时间	输入

【返回值】

无

【错误码】

无

【需求】

包 com.hisilicon.dlna.file 和构造函数 IDMRCallback

【注意】

本接口为预留接口，调用此接口无效。

【举例】

无

int getAllTime ()

【目的】

构造函数 IDMRCallback 的一个实现，客户端通过此接口获取播放总时长。

【语法】

int getAllTime ();

【描述】

客户端通过此接口获取播放总时长。

【参数】

无

【返回值】

返回值	描述
大于等于 0	播放总时长
其他	错误

【错误码】

无

【需求】

包 com.hisilicon.dlna.file 和构造函数 IDMRCallback

【注意】

本接口为预留接口，调用此接口无效。

【举例】

无

interface IDMRServer

【目的】

构造函数，用来控制 DMR 启动和销毁等功能的接口。

【语法】

```
interface IDMRServer
{
    boolean create(String name);
    boolean destroy();
    void setName(String name);
    boolean setDeviceName(String dmrName, String name);
    void startActivity(in Intent intent);
    void sendBroadcast(in Intent intent);
    IDMRCallback getDMRCallback();
    void registerPlayerController(IDMRPlayerController playerController);
    void unregisterPlayerController();
    IDMRPlayerController getDMRPlayerController();
}
```

【描述】

构造函数，用来控制 DMR 启动和销毁等功能。

【参数】

无

【返回值】

无

【错误码】

无

【需求】

包 com.hisilicon.dlna.file

【注意】

无

【举例】

参考 5.2.2.3 节 DMRAIDL 接口使用方法

boolean create(String name)

【目的】

构造函数 IDMRServer 的一个实现，用来开启 DMR 服务。

【语法】

boolean create(String name);

【描述】

用来开启 DMR 服务。

【参数】

参数名称	描述	输入/输出
time	设置播放器的总时间	输入

【返回值】

返回值	描述
true	创建成功
false	创建失败

【错误码】

无

【需求】

包 com.hisilicon.dlna.file

【注意】

无

【举例】

参考 5.2.2.3 节 DMR AIDL 接口使用方法

boolean destroy ()

【目的】

构造函数 IDMRServer 的一个实现，用来关闭 DMR 服务。

【语法】

boolean destroy();

【描述】

用来关闭 DMR 服务。

【参数】

无

【返回值】

返回值	描述
true	关闭成功
false	关闭失败

【错误码】

无

【需求】

包 com.hisilicon.dlna.file

【注意】

无

【举例】

无

void setName (String name)

【目的】

构造函数 IDMRServer 的一个实现，用来设置 DMR 名称。

【语法】

void setName (String name);

【描述】

用来设置 DMR 名称。

【参数】

参数名称	描述	输入/输出
name	设置 DMR 的名称	输入

【返回值】

无

【错误码】

无

【需求】

包 com.hisilicon.dlna.file

【注意】

无

【举例】

无

void setDeviceName (String name)

【目的】

构造函数 IDMRServer 的一个实现，用来启动后更换设置 DMR 名称。

【语法】

boolean setDeviceName(String dmrName, String name);

【描述】

用来启动后更换设置 DMR 名称。

【参数】

参数名称	描述	输入/输出
dmrName	新的 DMR 的名称	输入
name	原来的 DMR 名称	输入

【返回值】

返回值	描述
true	设置成功
False	设置失败

【错误码】

无

【需求】

包 com.hisilicon.dlna.file

【注意】

无

【举例】

无

void startActivity(in Intent intent)

【目的】

构造函数 IDMRServer 的一个实现，用来启动一个 Activity。

【语法】

void startActivity(in Intent intent);

【描述】

用来设置 DMR 名称。

【参数】

参数名称	描述	输入/输出
Intent intent	Activity 的名称	输入

【返回值】

无

【错误码】

无

【需求】

包 com.hisilicon.dlna.file

【注意】

无

【举例】

无

void sendBroadcast(in Intent intent)

【目的】

构造函数 IDMRServer 的一个实现，用来发送一个广播。

【语法】

void sendBroadcast(in Intent intent);

【描述】

用来发送一个广播。

【参数】

参数名称	描述	输入/输出
Intent intent	Activity 的名称	输入

【返回值】

无

【错误码】

无

【需求】

包 com.hisilicon.dlna.file

【注意】

无

【举例】

无

IDMRCallback getDMRCallback()

【目的】

构造函数 IDMRServer 的一个实现，用来获取 DMR 回调接口。

【语法】

IDMRCallback getDMRCallback();

【描述】

用来获取 DMR 回调接口。

【参数】

无

【返回值】

返回值	描述
IDMRCallback	IDMRCallback 构造函数

【错误码】

无

【需求】

包 com.hisilicon.dlna.file

【注意】

无

【举例】

参见 IDMRCallback 接口

void registerPlayerController(IDMRPlayerController playerController)

【目的】

构造函数 IDMRServer 的一个实现，用来注册播放器控制接口。

【语法】

void registerPlayerController(IDMRPlayerController playerController)

【描述】

用来注册播放器控制接口。

【参数】

参数名称	描述	输入/输出
playerController	IDMRPlayerController 构造函数	输入

【返回值】

无

【错误码】

无

【需求】

包 com.hisilicon.dlna.file

【注意】

与函数 unregisterPlayerController 成对使用

【举例】

参考 IDMRPlayerController 构造函数

void unregisterPlayerController ()

【目的】

构造函数 IDMRServer 的一个实现，注销播放器控制接口。

【语法】

void unregisterPlayerController ()

【描述】

注销播放器控制接口，在播放器关闭后必须调用此接口。

【参数】

无

【返回值】

无

【错误码】

无

【需求】

包 com.hisilicon.dlna.file

【注意】

- 在播放器关闭后必须调用此接口。
- 与函数 registerPlayerController 成对使用

【举例】

无

IDMRPlayerController getDMRPlayerController()

【目的】

构造函数 IDMRServer 的一个实现，获取播放器控制接口。

【语法】

IDMRPlayerController getDMRPlayerController()

【描述】

获取播放器控制接口。

【参数】

无

【返回值】

返回值	描述
IDMRPlayerController	IDMRPlayerController 构造函数

【错误码】

无

【需求】

包 com.hisilicon.dlna.file

【注意】

无

【举例】

无

interface IDMSServer

【目的】

构造函数，DMS 服务控制接口。

【语法】

```
interface IDMSServer
{
    boolean create(String dmsName, String dir);
    boolean destroy();
    boolean setDeviceName(String dmsName, String name);
    boolean addShareDir(String dmsName, String dir);
boolean delShareDir(String dmsName, String dir);
}
```

【描述】

构造函数，DMS 服务控制接口。

【参数】

无

【返回值】

无

【错误码】

无

【需求】

包 com.hisilicon.dlna.file

【注意】

无

【举例】

参考“5.2.2.4 DMS AIDL 接口使用方法”。

boolean create(StringdmsName, Stringdir)

【目的】

构造函数 IDMSServer 的一个实现，用来开启 DMS 服务。

【语法】

boolean create(String dmsName, String dir);

【描述】

用来开启 DMS 服务。

【参数】

参数名称	描述	输入/输出
dmsName	设置 DMS 的名称	输入
dir	设置 DMS 共享目录	输入

【返回值】

返回值	描述
true	创建成功
False	创建失败

【错误码】

无

【需求】

包 com.hisilicon.dlna.file

【注意】

无

【举例】

参考“5.2.2.3 DMR AIDL 接口使用方法”。

boolean destroy ()

【目的】

构造函数 IDM S Server 的一个实现，用来关闭 DM S 服务。

【语法】

boolean destroy();

【描述】

用来关闭 DM S 服务。

【参数】

无

【返回值】

返回值	描述
true	关闭成功
False	关闭失败

【错误码】

无

【需求】

包 com.hisilicon.dlna.file

【注意】

无

【举例】

无

boolean setDeviceName(String dmsName, String name)

【目的】

构造函数 IDM S Server 的一个实现，用来启动后更换 DM S 名称。

【语法】

boolean setDeviceName(String dmsName, String name);

【描述】

用来设置 DMR 名称。

【参数】

参数名称	描述	输入/输出
dmsName	新的 DMS 的名称	输入
name	原来的 DMS 名称	输入

【返回值】

返回值	描述
true	设置成功
False	设置失败

【错误码】

无

【需求】

包 com.hisilicon.dlna.file

【注意】

无

【举例】

无

boolean addShareDir(String dmsName, String dir)

【目的】

构造函数 IDMS Server 的一个实现，用于添加 DMS 的共享目录。

【语法】

boolean addShareDir(String dmsName, String dir)

【描述】

用于添加 DMS 的共享目录。

【参数】

参数名称	描述	输入/输出
dmsName	DMS 的名称	输入
dir	DMS 共享的目录	输入

【返回值】

返回值	描述
true	添加成功
False	添加失败

【错误码】

无

【需求】

包 com.hisilicon.dlna.file

【注意】

无

【举例】

无

boolean delShareDir(String dmsName, String dir)

【目的】

构造函数 IDMS Server 的一个实现，用来删除 DMS 的共享目录。

【语法】

boolean delShareDir(String dmsName, String dir)

【描述】

用来删除 DMS 的共享目录。

【参数】

参数名称	描述	输入/输出
dmsName	DMS 的名称	输入
name	需要删除的 DMS 共享目录	输入

【返回值】

返回值	描述
true	删除成功
False	删除失败

【错误码】

无

【需求】

包 com.hisilicon.dlna.file

【注意】

无

【举例】

无

5.3 Linux 接口 API 调用场景与示例分析

5.3.1 概述

本章节是对《HiDLNA API Development Reference.chm》的补充，演示了在具体场景下都会用到哪些函数，以及这些函数的调用顺序。本文档描述了如何通过一系列的 API 调用实现 DMR、DMP 或 DMS 的功能。

5.3.1.1 DMS 调用场景

场景一

用 DMS 共享一个或几个目录，主要骤如下：

步骤 1 启动并初始化 DMS 协议栈：

```
HI_DLNA_InitStack(HI_DLNA_INIT_MODE_DMS);
```

该方法启动并初始化了 DMS 协议栈。

步骤 2 添加共享的目录路径：

```
HI_DLNA_AddDmsSharedDir (path1,strlen(path1),HI_NULL_PTR);
HI_DLNA_AddDmsSharedDir (path2,strlen(path2), HI_NULL_PTR);
… … …
HI_DLNA_AddDmsSharedDir (pathx,strlen(pathx), HI_NULL_PTR);
```

----结束

场景二

删除由 DMS 共享出的一个或多个目录。要在场景一的基础上才能删除目录，否则没有任何意义。主要骤如下：

步骤 1 启动并初始化 DMS 协议栈：

```
HI_DLNA_InitStack(HI_DLNA_INIT_MODE_DMS);
```

步骤 2 添加共享的目录路径：

```
HI_DLNA_AddDmsSharedDir (path1,strlen(path1), HI _NULL_PTR);
HI_DLNA_AddDmsSharedDir (path2,strlen(path2), HI _NULL_PTR);
… … …
HI_DLNA_AddDmsSharedDir (pathx,strlen(pathx), HI _NULL_PTR);
```

步骤 3 删除共享目录：

```
HI_DLNA_DelDmsSharedDir (path1,strlen(path1), HI _NULL_PTR);
HI_DLNA_DelDmsSharedDir (path2,strlen(path2), HI _NULL_PTR);
… … …
HI_DLNA_DelDmsSharedDir (pathx,strlen(pathx), HI _NULL_PTR);
```

----结束

场景三

退出 DMS。同样必须在场景一的基础上才能执行退出，没有必要在删除目录后再执行退出 DMS 命令。

注意

HI_DLNA_DestoryStack **前可以有未被** delete **的共享目录，共享目录会被自动** delete。

主要使用步骤如下：

步骤 1 启动并初始化 DMS 协议栈：

```
HI_DLNA_InitStack(HI_DLNA_INIT_MODE_DMS);
```

步骤 2 可以根据需要添加要共享的目录，也可以什么共享目录都不添加。不添加共享目录不会影响 DMS 协议栈的启动或者关闭；

步骤 3 退出关闭协议栈：

```
HI_DLNA_DestoryStack();
```

----结束

场景四

更改上传的目标路径。更改上传目录需要启动 DMS，并通过下面的 API 修改。也可以直接编辑 dlna.ini 文件。两种修改方法都需要在 DMS 重启后能生效。主要步骤如下：

步骤 1 启动并初始化 DMS 协议栈：

```
HI_DLNA_InitStack(HI_DLNA_INIT_MODE_DMS);
```

步骤 2 修改上传目录：

```
HI_DLNA_SetDefaultUploadPath(Path_For_Upload);
```

此时已经修改成功，但是对于当前正在运行的 DMS 并不生效，只有当下次启动 DMS 时才会生效。

----结束

场景五

修改 DMS 的名称并立即生效。DMS 的名称在每次启动时都需要修改，否则会显示一个默认的名称。主要使用步骤如下：

步骤 1 启动并初始化 DMS 协议栈：

```
HI_DLNA_InitStack(HI_DLNA_INIT_MODE_DMS);
```

步骤 2 设置 DMS 名称：

```
HI_DLNA_SetDeviceName(Name_For_DMS);
```

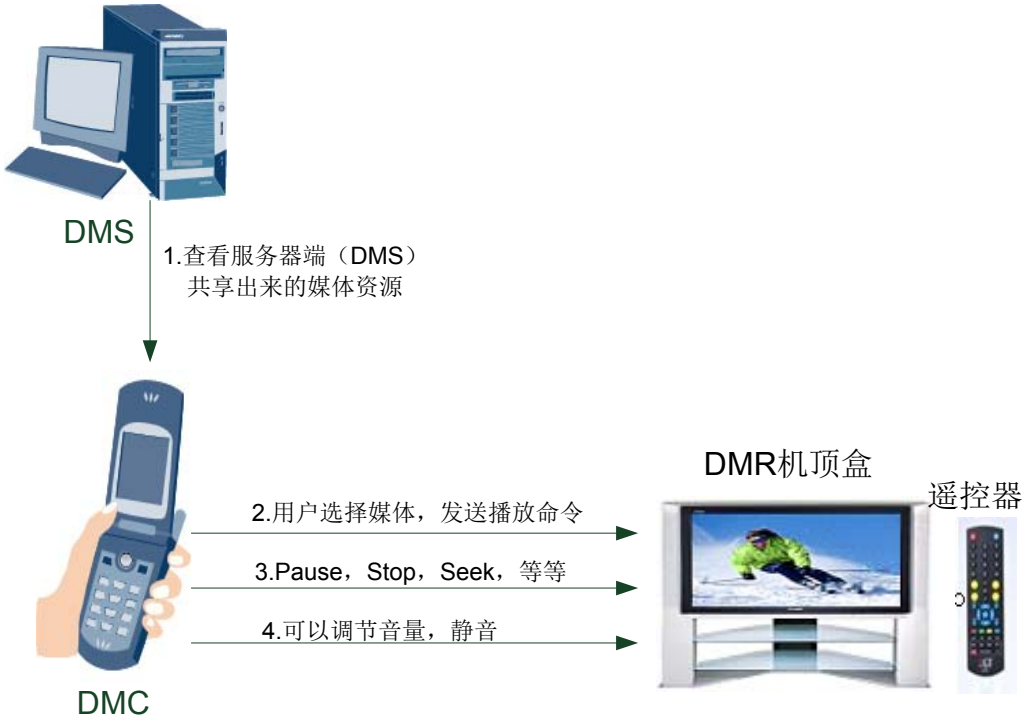
----结束

5.3.1.2 DMR 播放控制场景

DMR 常见的场景流程

DMR 常见的场景如图 5-1 所示。

图5-1 DMR 场景示例图



机顶盒为 DMR，主要用于接受 DMC 端控制，响应 DMC 操作，以图 5-1 为例，PC 作为 DMS 端，共享内容，手机作为 DMC 端，可以浏览 DMS 端共享的内容，查看共享出的文件信息，并可以选择某个文件推送到机顶盒（DMR）上播放。

另外，在机顶盒播放过程中，也可以用遥控器执行相应的操作，例如使用遥控器退出，暂停等。

注意

机顶盒端播放状态改变时，除了要控制机顶盒端播放器状态外，DMR 还需要调用接口通知 DMC 和 DMS 设备，而 DMP 不需要通知 DMC 和 DMS 设备。如暂停播放时 DMR 需要通知 DMC 和 DMS，DMP 不需要。

流程分析（以 sample_dmr.c 为例）

场景一启动 DMR

该接口必须首先调用，调用成功后，DMR 启动，此时可以使用 PC 或者手机上客户端发现该 DMR（默认名称为"Hi MediaRender"）。DMR 设备名称名称在 Init 时不可以通过参数输入，但可以使用 HI_DLNA_SetDeviceName 修改 DMR 的设备名称，HI_DLNA_SetDeviceName 会再次重启协议栈：

```
SampleCode_DlnaApiStackInit();
```

场景二 关闭 DMR

调用该接口后，DMR 将关闭，不再可用：

```
SampleCode_DlnaApiStackDeInit();
```

场景三 修改 DMR 名称

该函数输入参数即为新设置的 DMR 名称，DMR 默认名称为 Hi MediaRender。DMR 启动后可以通过调用该接口，修改 DMR 显示名称，修改后的名称，可以通过 PC 或者手机上的 DMC 客户端查看：

```
HI_DLNA_SetDeviceName(HI_CONST HI_CHAR *pcNewName);
```

场景四 IP 地址变化

如果机顶盒 IP 地址发生变化，可以通过调用此接口实现协议栈重启：

```
HI_DLNA_IpChange();
```

场景五 接收播放控制

DMR 主要用于接收 DMC 控制，响应 DMC 的操作行为。播放控制必须首先启动 DMR。参考示例函数向协议栈注册需要响应的事件：

```
SampleCode_DlnaApiRegisterDmrCallback();
```

调用该接口向协议栈注册响应的事件后，DMR 就能在函数中接收到 DMC 端的控制事件，例如设置 URL，播放，暂停等等，然后根据这些事件，做相应的操作。下面分别介绍对 DMC 端控制事件进行响应的接口函数：

- 接口 Callback_DlnaDmrSetMediaUri

```
HI_U32 Callback_DlnaDmrSetMediaUri
(
    HI_U32    ulInstanceID,
    HI_UCHAR  *pucUri,
    HI_U32    ulUrlLen,
    HI_U32    ulMediaId,
    HI_UCHAR  *pucMetaData,
    HI_U32    ulMetaDataLen,
    HI_VOID   *pvAuxData
)
```

  说明：

  – 收到该事件后，用于设置媒体的 URL 以及显示媒体元信息 MetaData（这些元信息可能包括文件名、文件大小、文件类型等等）。

- 当机顶盒收到该事件时，并不需要启动播放器，仅仅是设置 URL，当收到播放命令 play 时才开始启动播放器。
- 参数说明：

  pucUri：媒体的 URL，当开始播放时，将该 url 传给播放器；

  ulUrlLen：URL 的长度；

  pucMetaData：媒体元信息，该参数提供播放媒体的信息，例如媒体的名称，支持的协议栈等等，该参数也可能什么都不提供，取决于 DMS 端行为。pucMetaData 这个数据没有有标准的结构定义，解析名称等媒体信息请参考协议定义；

  ulMetaDataLen：媒体元信息的长度。

- 接口 Callback_DlnaDmrPlayInd

```
Callback_DlnaDmrPlayInd
(
    HI_U32    ulInstanceID,
    HI_UCHAR  *pucPlaySpeed,  /* string */
    HI_U32    ulStrLen,
    HI_VOID   *pvAuxData
)
```

  说明：

  - 当机顶盒收到该事件时，开始启动播放器，要播放的媒体 URL 是接口 Callback_DlnaDmrSetMediaUri 设置的 URL。
  - 如果从没有收到过 Callback_DlnaDmrSetMediaUri 事件，仅仅收到该事件是没有意义的，允许播放前一次的 URL。
  - 参数说明：

    pucPlaySpeed 指明播放速度，默认为 1，其余参数可暂不考虑。

- 接口 Callback_DlnaDmrPauseInd

  当收到该命令时，机顶盒暂停正在播放的媒体，如果当前不处于播放状态，该事件是没有意义的。由 DMC 发起的暂停处理，只要调用播放器的暂停处理接口就可以了，不涉及 DLNA 的任何处理。

- 接口 Callback_DlnaDmrStopInd

  当收到该命令时，机顶盒停止正在播放的媒体，停止后，如果一直存在 URL，再次播放允许继续播放上次的 URL。

- 接口 Callback_DlnaDmrSeekInd

```
HI_U32 Callback_DlnaDmrSeekInd
(
    HI_U32     ulInstanceID,
    HI_DLNA_SEEKMODE_E enSeekMode,
    HI_DLNA_PARATYPE_E enSeekDataType,
    HI_VOID      *pvSeekTarget,
    HI_VOID      *pvAuxData
)
```

说明：

- 当收到该接口事件时，需要根据 seek 类型，做相应的 seek 操作。

- 接口 Callback_DlnaDmrGetAllTime 、Callback_DlnaDmrGetCurrTime

  该接口用于请求当前媒体播放的总时间以及当前时间，格式为：“00:00:00”。

- 接口 Callback_DlnaDmrSetVol

```
HI_UCHAR * Callback_DlnaDmrSetVol
(
    HI_S32      ulInstanceID,
    HI_U32      ulDesiredVol,
    HI_CHAR      *pcChannel
)
```

  该接口用于 DMC 设置机顶盒的音量，参数 ulDesiredVol 即为音量值，取值（0-100），当收到该命令时，机顶盒播放器需要根据该音量值去设置音量。

- Callback_DlnaDmrSetMute

```
HI_UCHAR * Callback_DlnaDmrSetMute
(
    HI_S32    ulInstanceID,
    HI_BOOL   bDesiredMute,
    HI_CHAR   *pcChannel
)
```

  该接口用于 DMC 设置机顶盒的静音，参数 bDesiredMute 即为静音属性，取值 0 用于取消静音，1 用于设置静音。

场景六 遥控器控制接口

该接口仅适用于启动播放器后，使用遥控器的控制行为，例如当前 DMC 推送给 DMR 正在播放，不使用 DMC 退出播放，而使用遥控器退出播放，则退出时需要调用该接口：

```
HI_DLNA_SetDmrCurrPlayState(HI_DLNA_DMR_ACTION_E dmrState);
```

其中可设置的状态定义如下：

```
typedef enum hiDLNA_DMR_ACTION_E
{
    HI_DLNA_DMR_ACTION_INVALID = -1,
    HI_DLNA_DMR_ACTION_PLAY,
    HI_DLNA_DMR_ACTION_STOP,
    HI_DLNA_DMR_ACTION_PAUSE,
    HI_DLNA_DMR_ACTION_FINISH,
    HI_DLNA_DMR_ACTION_BUTT = HI_ENUM_END
}HI_DLNA_DMR_ACTION_E
```

5.3.1.3 DMP 调用场景

浏览 DMS 设备和文件。主要使用步骤如下：

步骤 1 启动并初始化 DMP 协议栈：

```
HI_DLNA_InitStack(HI_DLNA_INIT_MODE_DMP)
```

步骤 2 获取 DMS 设备列表：

```
HI_DLNA_DMP_GetDmsList(ppDmsList);
```

步骤 3 浏览目录：

```
HI_DLNA_DMP_BrowseDmsCD(devnum, ObjectID, in_beg, in_num);
```

步骤 4 等待搜索结束：

```
HI_DLNA_DMP_GetBrowseRspFlag();
```

步骤 5 通过函数获取目录和文件列表：

```
HI_DLNA_DMP_GetDmsContainList();
HI_DLNA_DMP_GetDmsMediaList();
```

步骤 6 获取文件的 url 之后，调用本地播放 API 进行播放即可。

----结束

5.3.2 HiDLNA Linux Sample Code 使用说明

本章节介绍了 Linux 环境下 HiDLNA 示例代码的安装和使用方法，介绍的文件包括 sample_dms.c、sample_dmr.c、sample_dms_dmr_dmp.c 和 sample_dmp.c。

5.3.2.1 Sample Code 的编译方法

将 Linux 版组件发布包解压，得到 hidlna 目录。其目录结构如表 5-1 所示。

表5-1 HiDLNA 目录结构

名称	功能
include	目录下包含编译 sample 所需的 hidlna 头文件
lib	目录下包含编译和运行 sample 所需的 hidlna 库文件
Makefile	编译 sample 的 Makefile
sample	目录下包含了 sample 源码文件和其他相关文件

另外，在发布包中有一个 component_common 目录，其下面包含了 HiDLNA 编译所需要的公共库文件。

HiDLNA Sample Code 在 Linux SDK 下的编译可参考如下步骤：

步骤 1 将发布的 SDK 包解压缩，以 Hi3716XV100R001C00SPC0A1 为例，tar 后 SDK 目录为 Hi3716XV100R001C00SPC0A1。如果有补丁包，需先合入补丁包，然后根据发布包的说明执行 `make build` 编译好 SDK 包。

步骤 2 拷贝 hidlna 目录到 sdk 发布包 sample 目录下，以 Hi3716XV100R001C00SPC0A1 为例，拷贝到 Hi3716XV100R001C00SPC0A1/sample/目录下。

步骤 3 将 hidlna/include 下头文件拷贝到 sdk 发布包 pub/include 目录下，将 hidlna/lib/arm-hisiv200-linux-debug 下的动态库库文件拷贝到 SDK 发布包 pub/lib/share 目录下，如：Hi3716XV100R001C00SPC0A1/pub/include，Hi3716XV100R001C00SPC0A1/pub/lib/share。

步骤 4 将 component_common 目录下的 libcomponent_common.so 库文件拷贝到 SDK 发布包 pub/lib/share 目录下。

步骤 5 修改 SDK/sample 目录下的 makefile 文件，在 `all:` 这一行下添加 `make -C hidlna`，以 Hi3716XV100R001C00SPC0A1 为例：修改 Hi3716XV100R001C00SPC0A1/sample/Makefile 文件。

步骤 6 在 sdk 发布包根目录下执行 `make sample`，编译 HiDLNA sample。

----结束

5.3.2.2 Sample Code 运行前的配置

编译好的 HiDLNA sample 在开发板上的运行可参考如下步骤：

步骤 1 将 Linux SDK 的运行环境烧写到单板上，并能进入到 Shell 提示符。

步骤 2 配置单板的 IP。由于默认情况下 Linux 系统并没有联网，需要给系统配置一个 IP：

```
ifconfig eth0 xxx.xxx.xxx.xxx
```

另外，sample_dms 运行需要有 lo(127.0.0.1)，可以通过下面的命令获得：

```
ifconfig lo 127.0.0.1
```

步骤 3 编译出的 Sample Code 依赖 hidlna/lib/arm-hisiv200-linux-debug 下的动态库文件，可以将这些文件拷贝到/usr/lib 目录下。另外一种方法是通过 NFS 挂载的方法将依赖的动态库文件（hidlna/lib/arm-hisiv200-linux-debug）添加到 LD_LIBRARY_PATH 中，以 NFS 挂载后的共享库目录为/mnt/rootfs_full/dlnalib 为例：

```
export LD_LIBRARY_PATH="/mnt/rootfs_full/dlnalib:$LD_LIBRARY_PATH"
```

同时，需要将 component_common 目录下的 libcomponent_common.so 库文件拷贝到 /usr/lib 目录下。或者通过 NFS 挂载的方法共享出来。hidlna 的 sample 文件运行时会依赖此库。

步骤 4 将 sample 文件传到开发板的任意目录下，并对文件添加可执行权限：

```
chmod u+x sample_dms sample_dmr sample_dmp sample_dms_dmr_dmp
```

----结束

5.3.2.3 Sample DMS 测试方法

在 sample_dms 可执行文件所在的目录下建立 data 和 uploaddir 目录，并在 data 目录中放入要共享的媒体文件。运行 sample_dms 命令后，会有如下提示：

```
# ./sample_dms
*****************************
* DMS Help Info *
*****************************
a:dms init
b:change dms friendly name
c:add "./data" as shared directory
h:add shared directory by hand,such as an Absolute Dir Path from SD card
u:set "./uploaddir" as new upload directory
p:set "/usr/share/dlna/dlnaUploadDir as upload directory
r:remove "./data"
q:dms deinit
>>
```

提示的说明如下：

- a：dms init。dms 协议栈初始化
- b：change dms friendly name。设定 dms 的显示名称，在此为“This is a Test”
- c：add "./data" as shared directory。将当前目录下的“data”文件夹共享
- h：add shared directory by hand,such as an Absolute Dir Path from SD card。由用户指定共享的文件夹路径，如，输入“/mnt”，则 DMS 会在用权限访问的条件下共享“/mnt”目录中的内容
- u：set "./uploaddir" as new upload directory。将当前目录的 uploaddir 设为上传目录。此设置需要重启 DMS 生效
- p：set "/usr/share/dlna/dlnaUploadDir as upload directory。恢复默认上传目录。此设置需要重启 DMS 生效
- r：remove "./data"。将“data”文件夹取消共享
- q：dms deinit。关闭 dms

测试 sample_dms 可参考如下步骤：

步骤 1 测试启动，更改名称，添加目录的功能：

依次输入命令 a、b、c；完成后会在 dmp 或 dmc 设备中看到 dms 设备，名称为（This is a Test）。

步骤 2 输入 h 命令来手动添加另一个目录为共享目录。输入 h，提示输入目录的路径：

```
Please input the directory path or "quit" to back:
```

输入要共享的路径，如，`/mnt` 然后回车。

步骤 3 输入命令 r,sample_dms 会移除 data 这个共享目录，但不会移除通过 h 命令添加的其它目录。

步骤 4 输入命令 u，上传目录修改为当前目录下的 uploaddir 并在下次启动 sample_dms 后生效。

步骤 5 输入命令 p，上传目录会被恢复为协议栈自己设置的上传目录（/usr/share/dlna/dlnaUploadDir），并在下次启动 sample_dms 后生效。

还有一种修改上传目录的方法为直接修改配置文件，其路径为/usr/share/dlna/dlna.ini。

步骤 6 输入命令 q 会退出 sample_dms，退出前程序会释放协议栈占用的资源。

----结束

5.3.2.4 Sample DMR 测试方法

运行 sample_dmr 命令后，会有如下提示：

```
# ./sample_dmr
*****************************
* DMR Help Info *
*****************************
a:dmr init
b:set dmr friendly name
c:set dmr  state
d:dmr ip change
q:dmr deinit
```

提示的说明如下：

- a：dmr init。初始化 dmr，启动一个名为 Hi MediaRender 的 DMR
- b：set dmr friendly name。设置 dmr 名称为：This is a Test
- c：set dmr state。设置 dmr 状态
- d：dmr ip change。如果 ip 地址变化，可以通过此选项重启 dmr
- q：dmr deinit。退出 dmr

启动 dmr 后，只能对 dmr 部分参数测试，并通过打印信息显示出来。

可以使用 upnp 的认证工具 UPnP Certification Test Tool 进行单项测试，无法全部自动测试，部分用例无法测试。

5.3.2.5 Sample DMP 测试方法

运行 sample_dmp 命令后，会有如下提示：

```
# ./sample_dmp
*****************************
* DMP Help Info *
a:dmp start
b:Get dms list
c:browse directory
d:exit the device
```

```
q:quit
```

提示的说明如下：

- a：dmp start。启动 dmp
- b：Get dms list。获取能搜索到的 DMS 设备列表
- c：browse directory。根据获取到的 DMS 列表，先选择设备。然后输入‘0’浏览此设备第一级目录下的内容，之后每次输出‘c’命令都可以根据提示的 id 浏览相应的目录或媒体信息
- d：exit the device。退出之前浏览的 DMS 设备，退出后就可以重新选择另一个 DMS 设备并浏览其中的内容
- q：quit。退出 dmp

测试 sample_dmp 可参考如下步骤：

步骤 1 输入命令 a，启动 DMP。

步骤 2 输入命令 b，获取设备列表，此时会有设备列表的信息打印如下，记住要进入的设备 ID：

```
DMS:
   1 -- uuid:bb5e21ce-2222-11b2-f918-8E2D1302F7BE
FriendlyName HiMediaServer(jiang)
   2 -- uuid:55076f6e-6b79-4d65-6469-0022a10453e2
FriendlyName TwonkyServer [TestServer_NAS]
   3 -- uuid:0751b700-b25a-4998-9e98-50838396b62e
FriendlyName W00215319-PC: w00215319:
   4 -- uuid:624e5ba7-5871-4fd3-8e15-64eff64a3f71
FriendlyName TIANJIA-HP: t00204177:
   5 -- uuid:818b429c-9fc6-4e66-994b-3e6337550918
FriendlyName Z002096281A: z00209628:
   6 -- uuid:832a616e-cc8a-43c4-8122-e3f4c7170135
FriendlyName J00209609-VM1A: j00209609:
```

步骤 3 输入命令 c，浏览设备目录，会有提示输入“设备 ID”和“目录的 ID”，如果是第一次进入设备，最高层目录的 id 为 0，深层目录可根据打印获取到 ID ：

```
please choose one dms device: 1
Input dir id: 0
Input  id:0
DlnaParseActionResp:u:BrowseResponse
<OBJID>0$2</OBJID><NAME>Photos</NAME>
<OBJID>0$3</OBJID><NAME>Videos</NAME>
Browse Result
TotalMatches:2
NumberReturned:2
RealNumReturned:2
```

步骤 4 输入命令 d，退出设备，可以进入另一个 DMS 设备。

步骤 5 输入命令 q，退出。

----结束

5.3.2.6 Sample DMS_DMR_DMP 测试方法

运行 sample_dms_dmr_dmp 命令后，会有如下提示：

```
/*****************************/
a:dms_dmr_dmp start
q:dms_dmr_dmp quit
/** DMP Help Info **/
b:Get dms list
c:browse directory
d:exit the device
/** DMR Help Info **/
h:set dmr&dms friendly name
i:set dmr  state
j:dmr ip change
/** DMS Help Info **/
r:add "./data" as shared directory
s:add shared directory by hand,such as an Absolute Dir Path from SD card
t:set "./uploaddir" as new upload directory
u:set "/usr/share/dlna/dlnaUploadDir as upload directory
v:remove "./data"
/*****************************/
```

sample_dms_dmr_dmp 在同一个进程中启动了 DMP、DMR、DMS 服务，而之前的 sample_dmp、sample_dmr、sample_dms 在同一个进程中仅启动了其中一个服务。其测试方法请参考 sample_dmp、sample_dmr、sample_dms 的测试方法。以下几点需要特殊说明：

- a:dms_dmr_dmp start。会同时启动 DMS、DMR、DMP 三个服务。
- q:dms_dmr_dmp quit。会将 DMS、DMR、DMP 三个服务同时关闭。
- h:set dmr&dms friendly name。修改 DMR 和 DMS 的设备名称，DMR 和 DMS 的名称同时修改。

----结束

5.4 HiDLNA 开发常见问题汇总

5.4.1 Android 开发常见问题

5.4.1.1 如何更新 DMR、DMS 的设备描述？

问题描述

如何支持由外部传入设备描述文件对设备信息控制？

问题分析

DMC 端可以看到 DMR 和 DMS 的设备制造商信息为“Hisilicon Technologies Co.,Ltd”，如果用户需要自己定制此类信息可按如下方法操作。

解决办法

首先找到 HiDLNASettings/res/values/strings.xml 文件，其中有“dmrdescription”和“dmsdescription”两个长字符串。用户可以修改如下几行：

```
<manufacturer>Hisilicon Technologies Co.,Ltd</manufacturer>
        <manufacturerURL>http://www.Hisilicon.com</manufacturerURL>
        <modelName>Hisilicon MediaXXXX</modelName>
        <modelNumber>1.1</modelNumber>
        <modelURL>http://www.Hisilicon.com</modelURL>
```

修改完成后，需要重新编译 HiDLNASettings 并更新系统的 HiDLNASettings.apk。重启系统后修改才能生效。

5.4.1.2 DLNA 的可维可测手段有哪些？

问题描述

当 DLNA 推送出现异常错误时，有什么手段可以定位问题发生的原因？

问题分析

初步定位问题可以根据 PROC 和 LOG，疑难问题的定位可以使用 tcpdump 抓包分析。

解决办法

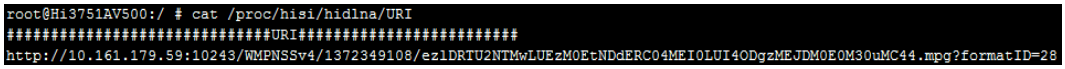
【方法一】通过 DLNA PROC

使用该功能前，请确认已经完成 DMR 的推送。使用 PROC 可以获取以下信息：

- URI：串口中输入 cat /proc/hisi/hidlna/URI

  得到如图 5-2 所示打印，此 URL 为 SetAVTransportURI 消息中默认携带的 URI 信息。可以浏览器或者 VLC 工具打开此 URI，判断资源的合法性。

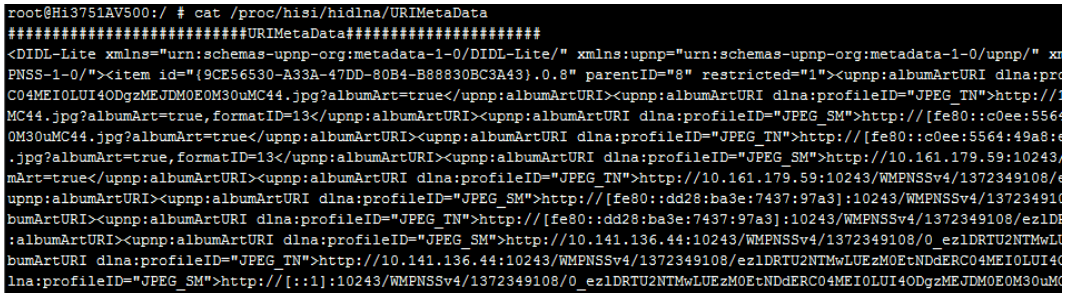
图5-2 PROC 中 URI 打印



- URIMetaData：串口中输入 cat /proc/hisi/hidlna/URIMetaData

  得到如图 5-3 所示的打印，打印内容为 SetAVTransportURI 消息中默认携带的 MetaData 信息，帮助查看推送类型，是否支持 seek 等信息。

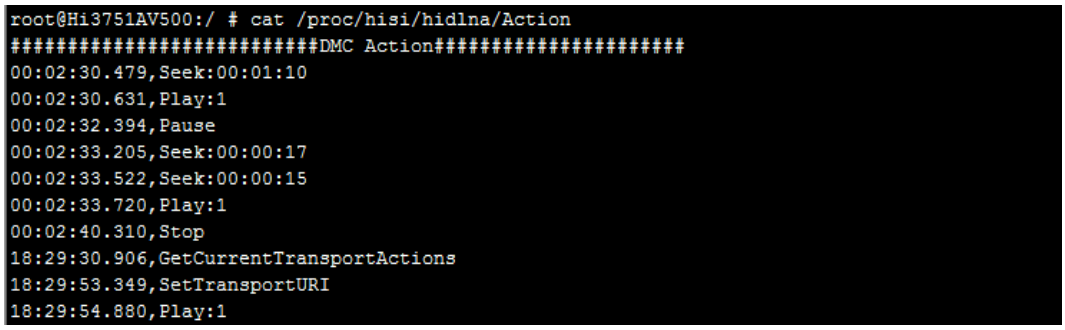
图5-3 PROC 中 URIMetaData 打印



- Action：串口中输入 cat /proc/hisi/hidlna/Action

  得到如图 5-4 中的打印，打印内容为近 10 次推送消息简单记录，用于判断播放状态是否正确。当播放状态发生错误时，辅助判断是那些操作引起了状态错误。

图5-4 PROC 中 Action 打印



- PlayerTimer：串口中输入 cat /proc/hisi/hidlna/PlayerTimer

  得到如图 5-5 中的打印，内容记录调用 MediaPlayer 接口的时间记录。

图5-5 PROC 中 PlayerTimer 打印

```
root@Hi3751AV500:/ # cat /proc/hisi/hidlna/PlayerTimer
##########################HiPlayer Time####################
#old:
00:01:56.917,playurl:http://10.161.179.101:51121/74366056-0cb7-402e-9e3a-f32d94207a68.mp4
00:02:30.364,before seek
00:02:30.488,after seek
00:02:30.491,before seek
00:02:30.671,after seek
00:02:33.230,before seek
00:02:33.599,after seek
00:02:33.604,before seek
00:02:33.853,after seek
18:29:55.481,before reset
18:29:55.482,after reset
#new:
18:29:55.486,playurl:http://127.0.0.1:10243/WMPNSSv4/1372349108/0_ezlDRTU2NTMwLUEzM0EtNDdERC04MEI0LUI4ODgzMEJDM0E0M30uMC44.mp4
,
```

【方法二】通过 DLNA LOG

DLNA log 系统使用 Android 标准 Logcat，对协议关键流程及接收的网络命令进行打印，客户可通过标准 Logcat 命令从串口或者电脑端 adb 连接抓取 Log。

【方法三】通过网络抓包

单板提供网络抓包命令，兼容性相关问题还可以通过网络抓包进行定位解决。

以工作在 eth0 上为例，在串口输入命令：

```
tcpdump -p -vv -s 0 -i eth0 -w /sdcard/1.pcap
```

客户可抓取问题发生时的网络包状态文件，供海思开发人员分析。

5.4.1.3 DLNA 服务是否正常启动？

问题描述

单板启动后，怎么判断 DLNA 的服务是否正常启动？

问题分析

通过 LOG 打印，查看是否有 UPnP（Universal Plug and Play，通用即插即用协议）协议栈启动的打印，从这里可以看出 DLNA 服务是否正常启动。

解决办法

步骤 1 先在串口中输入： ps | grep dlna

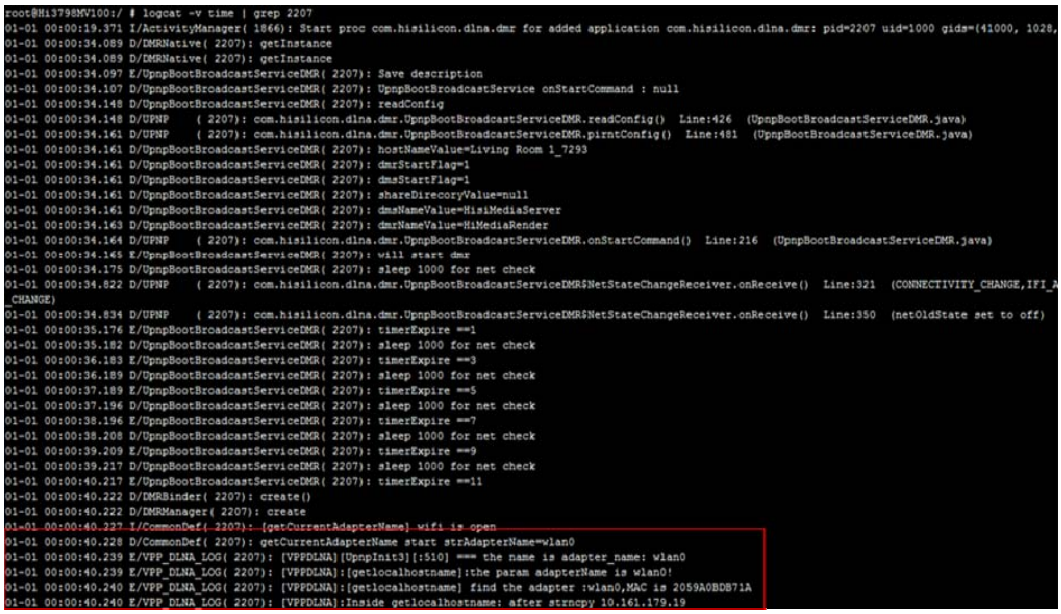
图5-6 查询 DLNA 服务进程号

```
130|root@Hi3798MV100:/ # ps | grep dlna
system    2194  1364  539756 20336 ffffffff 2b1387f4 S com.hisilicon.dlna.dms
system    2207  1364  534664 20268 ffffffff 2b1387f4 S com.hisilicon.dlna.dmr
```

步骤 2 根据 dlna 的进程号再在串口中输入：logcat –v time | grep PID

如图 4-5 所示，此时 PID 的值为 2207。在如图 5-7 的打印中查看是否包含红框中包含的打印，如果包含此部分打印，表明协议栈已正常启动。

图5-7 查看 DLNA 的 LOG



----结束

5.4.1.4 DLNA 设备为何概率性无法发现？

问题描述

DLNA 设备已经正常启动，使用手机客户端概率性无法发现设备。

问题分析

只有手机 DLNA 客户端与 DLNA 设备完成 SSDP（Simple Service Discovery Protocol，简单设备发现协议）的交互过程，并能正确下载并解析设备和服务的描述文档，才能完成设备发现。

解决办法

设备发现过程涉及到 SSDP 消息分发、设备和服务信息下载解析。SSDP 消息分发使用的是 UDP（User Datagram Protocol），可能存在丢包，但海思的服务端已对此做了优化，增加了 SSDP alive 消息的发送频率，由于 UDP 丢包导致设备发现问题的情况较少。

较大概率是由于网络不稳定，导致客户端无法正常下载设备和服务的描述文档，从而导致设备发现问题。可以通过 ping 工具和 wifi 分析仪工具来检测网络状况。建议单板使用有线网络，或者 5G 网络，排除网络拥塞和临频干扰的影响。

5.4.1.5 DLNA 设备为何一直无法发现？

问题描述

DLNA 服务已经正常启动，但是有客户端仍发现不了 DLNA 设备。

问题分析

海思的 DLNA 只支持在一个网卡上工作。如 5.4.1.4 中所述，DLNA 服务端和客户端正确配合，才能完成设备发现。

解决办法

步骤 1 确认客户端和服务端是否处于同一个局域网。如果不在同一个局域网，需要将盒子和手机连接到同一个局域网。

步骤 2 观察手机上客户端是否能发现其他的 DLNA 设备

- 如果所有 DLNA 设备都不能发现，可以判断为第三方手机客户端问题，建议换一个 DLNA 客户端。
- 如果只有海思 DLNA 设备不能发现，需要 LOG 和网络抓包做进一步分析。

----结束

5.4.1.6 手机客户端为何推送失败？

问题描述

手机客户端选择推送，但是 DLNA 无任何反应或者播放失败。

问题分析

可能导致此类问题原因较多，主要可分为三类：

- 由于网络问题、客户端问题原因，DLNA 服务未收到客户端的消息。
- 客户端原因，推送的资源无效，导致播放失败。
- 媒体播放失败、Android 平台不支持问题。

解决办法

- DLNA 服务未收到客户端的消息

  DMR 只有收到 DMC 的消息，才能执行播放。一旦有网络状况差、第三方 DLNA 客户端实现不正确、海思 DLNA 服务端状态错误等原因，都可能导致 DMR 不能收到 Action 消息。

  可以通过查看 proc，看到最近几次 DMR 收到的消息，串口中输入：

  ```
  cat /proc/hisi/hidlna/Action
  ```

  观察如图 5-4 的 PROC 打印。若是推送操作，则查看：

  – 是否收到 SetTransportURI、Play 消息；

- 如果未收到消息，检查网络是否正常，客户端是否连接指定的 DMR 设备。

- 资源无效

  串口中输入：

  ```
  cat /proc/hisi/hidlna/URI
  ```

  得到如图 5-2 所示打印信息，此 URL 为 SetAVTransportURI 消息中默认携带的 URI 信息。

  - 若为图片推送，可以使用电脑上的浏览器打开此 URL；
  - 若为音视频推送，可以使用电脑上的 VLC（VideoLAN 客户端）打开此 URL，观察是否播放正常。如果播放失败，可初步判断问题原因是资源的不存在，为第三方 DMS 或者第三方手机客户端的问题。

    如果对比浏览器或者视频播放正常，则可继续对比 Android 平台本地播放的情况。

- 媒体播放失败、Android 平台不支持问题
  - 对于图片在浏览器中播放正常，使用 DLNA 推送失败，可以使用浏览器将图片下载下来，放到单板 sdcard 目录下，用 android 本地图片播放，观察是否能正常播放。如果 android 本地播放也失败，则说明是图片格式在 Android 平台上不支持。如果成功，则说明是 DLNA 下载解码出错，可进一步抓取 LOG 分析错误原因。
  - 对于音视频在 VLC 中播放正常，使用 DLNA 推送失败，可以使用本地的 VideoPlayer 验证是否播放正常。在串口中输入：

    am start -n com.hisilicon.android.videoplayer/.activity.VideoActivity URL

    这里的 URL 为推送对应的 cat /proc/hisi/hidlna/URI 的打印 URL 地址。

    - 如果 VideoPlayer 播放失败，可以初步判断为媒体问题。
    - 如果 VideoPlayer 播放成功则可进一步抓取 LOG 分析 DLNA 播放出错原因。

5.4.1.7 为何推送手机上的资源较慢、播放卡顿？

问题描述

推送手机上拍摄的图像、视频到单板上播放时，发现视频出现卡顿，图片加载缓慢。

问题分析

目前手机拍摄的图片分辨率大多在 1080P 以上，视频分辨率大多在 720P 以上。要想在手机上流畅地播放 720P 的视频，需要大于 1MB 的带宽。在网络状况不稳定时，DMR 从手机上的 DMS HTTP 下载媒体资源的速度通常达不到 1MB。此时会导致视频播放卡顿，图片加载速度缓慢。

解决办法

建议排除网络干扰，使用稳定的网络环境。如单板使用有线网络，或者单板和手机都使用稳定的无线网络环境：单板和手机都切换到 5G 频段；使用 wifi 分析仪分析当前网络，避免临频干扰等。

5.4.1.8 退出处理策略

问题描述

DMC 客户端推送一个资源后，在不退出播放的情况下，推送下一个资源，播放界面退出后又立即启动播放。

问题分析

在 DLNA 协议里：客户端停止播放时，客户端会向服务端发送 STOP 消息；客户端在一个推送正在执行播放时，再次推送，这两次推送之间也会发送 STOP 消息，这个 STOP 消息目的是 DMR 状态机的复位，并不是要退出播放。这两种情况下，客户端发送的 STOP 消息完全一样，对于服务端来说是无法区分收到的 STOP 消息是停止播放还是状态置位。

- 如果是停止播放消息，DLNA 应该退出播放；
- 如果不是，DLNA 就不应该退出播放，等待下一个推送到来，继续播放；不然 DLNA 播放会闪退后又启动播放，就是所谓的闪屏现象，这种现象会影响推送性能，降低用户体验。

解决办法

应对闪屏问题，海思 DLNA 的处理策略是，收到 STOP 消息时，延时三秒退出，如果此间收到后续的推送消息，则中止退出流程，继续播放。三秒延时退出带来的代价是退出时间变长。但相对于性能、推送体验的提升，我们认为此代价可控。

5.4.1.9 URL 中包含的特殊转义字符怎么处理？

问题描述

使用 skifta 推送资源到 DMR 播放失败。

问题分析

推送资源的 URI 中包含了“%20”，为了解决之前的兼容性问题，处理中将“%20”当做转义字符（转换为空格）。在 skifta 中“%20”实际代表的就是字符串“%20”。

解决办法

这里不转换“20%”为空格，使用 skifta 客户端推送播放成功。可能会引起其他把“%20”用作转义字符的应用推送失败。DLNA 的有不少类似的问题，只能折中考虑，兼容最常用的、大多数的客户端。

5.4.1.10 无线/有线网络绑定策略设置

问题描述

在系统策略为有线优先的情况，无法搜索到无线网络下的 DLNA 设备。

问题分析

海思的 DLNA 只支持在一个网卡上工作，默认情况下是跟随系统网络优先级，因此在多网络环境下会搜索不到其他网络下的 DLNA 设备。

解决办法

在上层应用开源代码中，修改网络优先级，DMS/DMR/DMP 是分开进行选择的，修改文件分别如下：

```
DMS：
android/packages/apps/HiMediaShare/src/com/hisilicon/dlna/dms/MediaService.java
DMR：
android/packages/apps/HiMediaRender/src/com/hisilicon/dlna/dmr/UpnpBootBroadcastSe
rviceDMR.java
DMP：
android/packages/apps/HiMediaCenter/src/com/hisilicon/dlna/mediacenter/MediaCenterA
ctivity.java
android/packages/apps/HiMediaPlayer/src/com/hisilicon/dlna/player/MainPlayerActivity.jav
a
```

修改点如下：

```
private final static String DEFAULT_USE_ADAPTER_WIFI_ETHERNET = "eth";     //wlan
```

5.4.2 该值为 eth 代表有线网络优先，为 wlan 代表无线网络优先。Linux 开发常见问题

5.4.2.1 Seek 功能中，DMR 端在回调中如何处理 seek 的 event？

问题描述

DMC 是如何控制 DMR 端 seek 的，请举例说明按时间 seek 是如何实现的？

问题分析

DMC 端控制播放进度时，会通过回调函数通知 DMR 端进行响应，使 DMR 播放器的进度与 DMC 端保持一致，这由 DMC 端传送的参数来控制如何响应。

解决办法

Seek 回调函数的定义为：

```
typedef HI_U32 (*HI_DLNA_DMR_SEEK_IND_PTR)
```

```
(
    HI_U32          ulInstanceID,
    HI_DLNA_SEEKMODE_E enSeekMode,
    HI_DLNA_PARATYPE_E enSeekDataType,
    HI_VOID         *pvSeekTarget,
    HI_VOID         *pvAuxData
)
```

执行 seek 操作时，DMR 指明 seek 类型以及 seek 参数，由于目前不支持播放列表，仅支持的 seek 操作为：HI_DLNA_SEEK_MODE_ABS_TIME 和 HI_DLNA_SEEK_MODE_REL_TIME。

enSeekDataType 的定义为：

```
typedef enum hiDLNA_PARATYPE_E
{
    HI_DLNA_PARA_TYPE_INVALID = -1,
    HI_DLNA_PARA_TYPE_STRING =0,
    HI_DLNA_PARA_TYPE_INT32,
    HI_DLNA_PARA_TYPE_UINT32,
    HI_DLNA_PARA_TYPE_FLOAT,
    HI_DLNA_PARA_TYPE_BUTT = HI_ENUM_END
}HI_DLNA_PARATYPE_E
```

即当参数enSeekMode指明了seek的类型，而参数enSeekDataType指定对应该seek的数据类型，该类型定义为参考DlnaParaTypeEn 结构

对于：HI_DLNA_SEEK_MODE_ABS_TIME 和 HI_DLNA_SEEK_MODE_REL_TIME，则收到的 enSeekDataType 类型为 0（HI_DLNA_PARA_TYPE_STRING）。

可参考如下解析：

```
typedef union _DmrSeekTargetValUn
{
        HI_DLNA_STRING_S stStrVal;
        HI_U32 ulVal;
        HI_S32  iVal;
        HI_FLOAT  fVal;
}DmrSeekTargetValUn;

HI_U32 Callback_DlnaDmrSeekInd
(
    HI_U32      ulInstanceID,
    HI_DLNA_SEEKMODE_E enSeekMode,
    HI_DLNA_PARATYPE_E enSeekDataType,
    HI_VOID         *pvSeekTarget,
    HI_VOID         *pvAuxData
)
```

```
{
    Dlna_Print("Received the Seek Indication\n");
    int len=0;
    HI_S32 s32Ret = 0;
    DmrSeekTargetValUn uSeekTargetVal;
    HI_U32 mseconds;
    HI_U32 hh,mm,ss;
    if(!pvSeekTarget)
    {
        printf("no seek data\n");
        return -1;
    }
    memset(&uSeekTargetVal,0,sizeof(DmrSeekTargetValUn));
    memcpy(&uSeekTargetVal,(DmrSeekTargetValUn
*)pvSeekTarget,sizeof((DmrSeekTargetValUn *)pvSeekTarget));
    printf("Seek mode:%d,dataType:%d\n",enSeekMode,enSeekDataType);
    switch(enSeekDataType)
    {
        case HI_DLNA_PARA_TYPE_STRING:
            printf("Seek
value:%s\n",(HI_DLNA_STRING_S)uSeekTargetVal.stStrVal);
            sscanf(uSeekTargetVal.stStrVal.pucBuf,"%d:%d:%d",&hh,&mm,&ss);
            mseconds = (hh * 3600 + mm * 60 + ss) * 1000;
            printf("second:%d\n",mseconds);
            s32Ret = HI_SVR_PLAYER_Seek(hPlayer, mseconds);
            if (HI_SUCCESS != s32Ret)
               printf("\e[31m ERR: seek fail, ret = 0x%x \e[0m \n", s32Ret);
Dlna_Print("ERR: seek status is : %d\n",stPlayerInfo.eStatus);
     break;
        case HI_DLNA_PARA_TYPE_INT32:
            printf("Seek value:%d\n",uSeekTargetVal.iVal);
            //to do:
            break;
        case HI_DLNA_PARA_TYPE_UINT32:
            printf("Seek value:%u\n",uSeekTargetVal.ulVal);
            break;
        case HI_DLNA_PARA_TYPE_FLOAT:
            printf("Seek value:%f\n",uSeekTargetVal.fVal);
            break;
        default:
            printf("no SeekDataType\n");
            return -1;
    }
    //HI_SVR_PLAYER_Seek(hPlayer, u32SeekTime);
```

```
    return HI_DLNA_RET_SUCCESS;
}
```

5.4.2.2 DMP 如何要返回目录列表的上一级？

问题描述

DMP 在获得某个设备的目录和文件列表时，如何要返回上一级的目录？

问题分析

DMP 遍历 DMS 共享内容时，是通过 ID 来进行的，因此需要对 ID 进行记录

解决办法

以如下目录结构为例，假设 video 的 ID 为 1，其有两个子目录 all video，by folder，ID 分别为 2 和 3

```
— video
    ├—— all video
    └—— by folder
```

要进入 video 目录则输入 ID 为 1，此时结果会有两个目录 all video 和 by folder，要进入 all video，则要输入 ID 2，返回的结果是 all video 下的内容。

如果此时要返回到 video 目录，则要输入 ID 1，正确的返回的结果是 all video 和 by folder，其他参数都要正确输入，参考如下信息：

```
HI_S32 HI_DLNA_DMP_BrowseDmsCD
(
    HI_S32  devnum,
    HI_CHAR *ObjectID,
    HI_S16  in_beg,
    HI_S16  in_num
);
```

5.4.2.3 如何使用 HI_DLNA_DEVICE_NODE_S 链表？

问题描述

DMP 搜索 DMS 时使用函数 HI_DLNA_DMP_GetDmsList，得到的链表类型为 HI_DLNA_DEVICE_NODE_S，这个返回的链表，是每次获取后就释放还是等应用程序退出时释放一次？是否会被函数 HI_DLNA_DMP_DeleteList ()释放掉？测试发现，如果有 DMS 新增，则可以在链表中得到 DMS 的信息；但如果有 DMS 退出，则不能从链表中自动删除 DMS 的信息？

解决办法

该链表在每次获取之后不一定要释放，是否释放根据程序需要决定。

HI_DLNA_DMP_DeleteList()不会释放该链表。

DMS 退出时，如果 DMP 收到 DMS 设备的 BYEBYE 消息，会删除 DMS 信息；通过再次调用 DlnaDmcGetDmsList()刷新。

5.4.2.4 如何指定协议栈输出 Log 的路径和名称？

问题描述

默认情况下 HiDLNA 协议栈的 log 文件是输出到当前目录下的。Error 日志输出到 DlnaLogError.txt 中，Info 日志输出到 lnaLogInfo.txt 中。如果由于某种原因希望 Log 输出到其它路径下该如何设置，例如，输出到“/tmp/dlnalog.txt”文件中？

解决办法

可以通过配置程序运行的环境变量的方法指定 HiDLNA 协议栈 Log 输出的路径和文件名，环境变量的名称为 DLNA_LOGFILE_PATH。例如，要输出到“/tmp/dlnalog.txt”文件中可以这样操作：

```
export DLNA_LOGFILE_PATH="/tmp/dlnalog.txt"
```

然后在此环境变量存在的情况下运行程序，就会输出到“/tmp/dlnalog.txt”文件中。

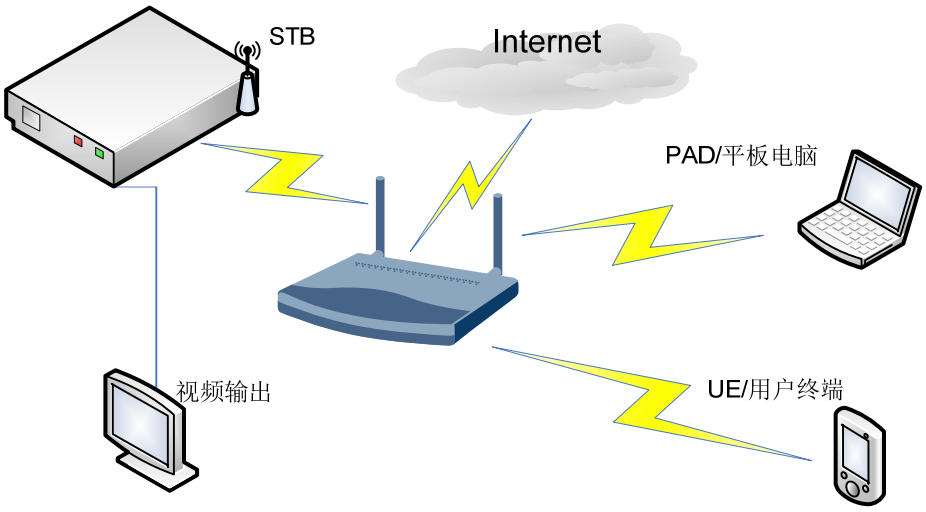
需要注意的是，指定协议栈 Log 的输出路径后将不再区分 Error 和 Info，全部都会输出到指定的这个文件中。

6 HiMultiScreen

6.1 概述

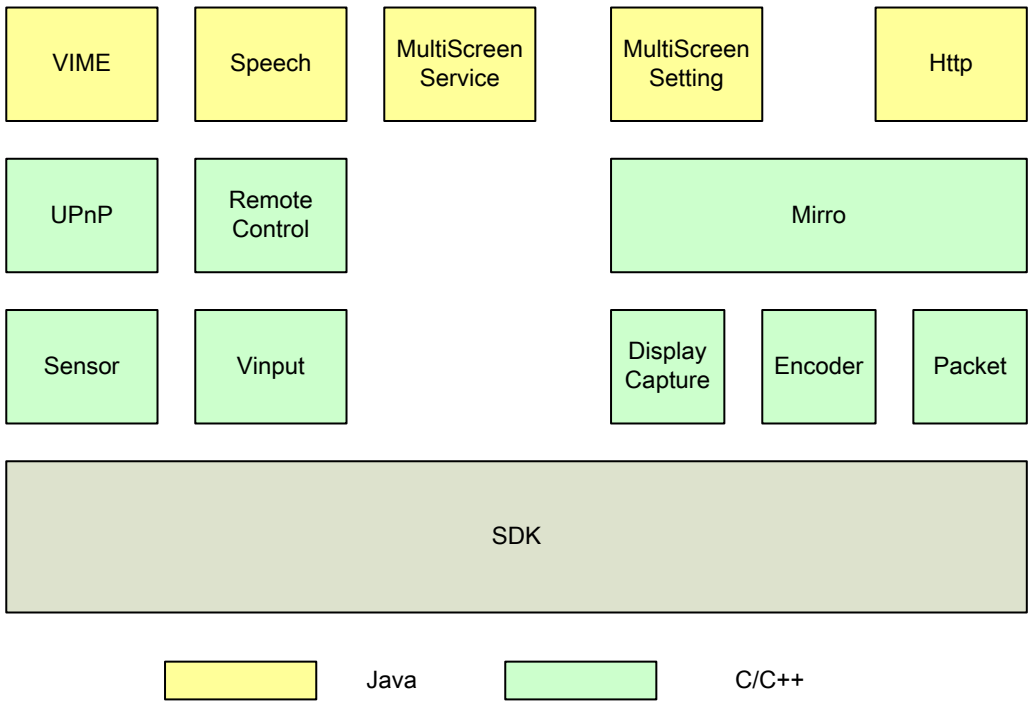
HiMultiScreen 是海思提供的应用组件之一，主要用于局域网范围内在手持设备上显示 STB 屏幕输出，并对 STB 进行远程控制。其软件包括客户端软件和服务端软件两部分，其中客户端软件运行在手持设备侧，服务端软件运行在 STB 侧，使用组网如图 6-1 所示。其中语音控制 Speech 需要组网环境具有 Internet 访问权限。

图6-1 HiMultiScreen 组网图



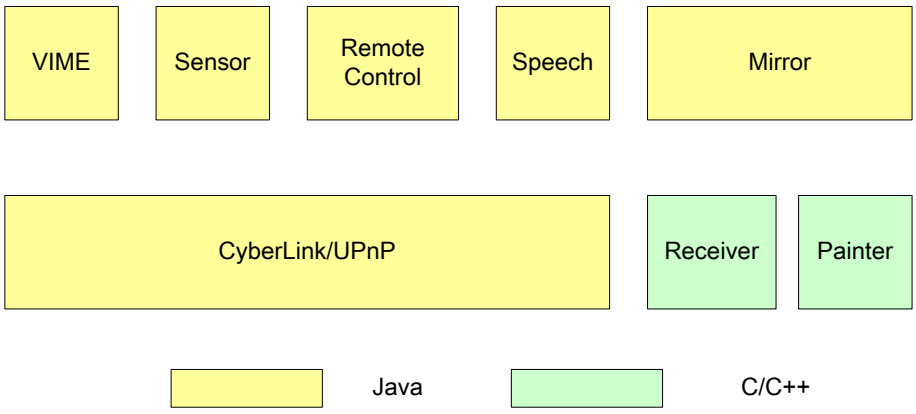
HiMultiScreen STB 服务器端架构框图如图 6-2 所示。图中黄色部分 Java 代码源码交付；图中绿色部分 C/C++代码以 so 库形式提供。

图6-2 HiMuliscreen STB 端架构框图



HiMultiScreen STB 手持设备客户端框图如图 6-3 所示。图中黄色部分，CyberLink/UPnP 为开源库，以源码形式提供，其余模块为应用层代码，源码交付；图中绿色部分以 so 库形式提供。

图6-3 HiMultiscreen 手持设备端架构框图



6.2 重要概念

【UPnP】

通用即插即用（Universal Plug and Play，简称 UPnP）是一套网络协议。该协议的目标是使家庭网络（数据共享、通信和娱乐）和公司网络中的各种设备能够相互无缝连接，并简化相关网络的实现。详细请参考 http://www.upnp.org/

【Mirror】

镜像，特指将机顶盒侧图形及视频显示传输到手持设备侧显示。

6.3 功能描述

6.3.1 主要特点

目前，HiMultiScreen 由设备搜索发现模块、Mirror 模块、RemoteControl 模块、VIME 模块、Sensor 模块以及 Speech 模块组成。各个模块的具体功能如下。

- 设备搜索发现：

  通过 UPnP 协议，手持客户端发现局域网范围内提供多屏服务的 STB 设备，并以“设备名+IP”地址的形式来描述一个受控 STB 设备。

- Mirror

  该模块显示和受控 STB 设备相同的图形和视频信息，并将用户的操控命令传送给 STB 设备服务端，这些操控命令可以方便用户对受控的 STB 进行模拟触摸屏的操作。

- RemoteControl

  模拟海思公板红外遥控器大部分常用按键，用户借助于 RemoteControl 即可实现类似于硬件遥控器的操作体验。

- VIME

  VIME（Virtual Input Method Editor）中文意为虚拟输入法，完成使用手机输入法对 STB 侧进行文本输入功能。客户端部署在手持设备侧，服务器端部署在 STB 机顶盒侧，客户端接收用户输入，并将输入提交给服务器端，由服务器端将输入最终提交到 STB 侧的当前文本框中，完成对 STB 的输入。

- Sensor

  该模块使用手持设备侧的 Sensor（当前支持 G-sensor），对 STB 侧的体感游戏进行操控。

- Speech

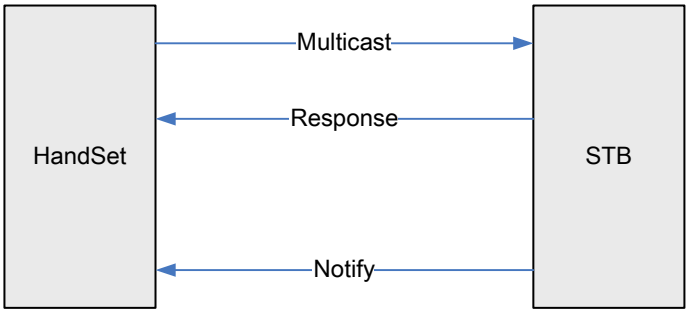
  该模块利用客户端 MIC 作为语音输入源， 通过因特网与讯飞语音云交互得到语义识别结果，并将语义识别结果提交给 STB 服务端。用户借助 Speech 模块实现对 STB 服务端的语音操控。例如可以实现智能对话、语音启动 STB 应用、语音搜片以及语音网页搜索等功能。

6.3.2 模块原理

6.3.2.1 设备发现

设备发现符合 UPnP 协议 1.0 版本。手持设备客户端应用中采用 CyberLink UPnP 开发包。设备发现手持设备与 STB 端交互流程见图 6-4。

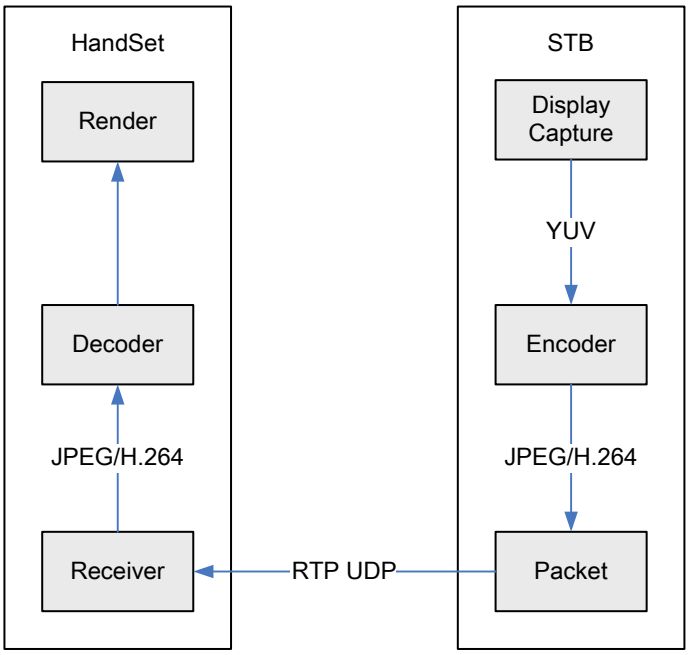
图6-4 设备发现交互



6.3.2.2 Mirror

Mirror 手持设备与 STB 端交互流程见图 6-5。

图6-5 Mirror 交互

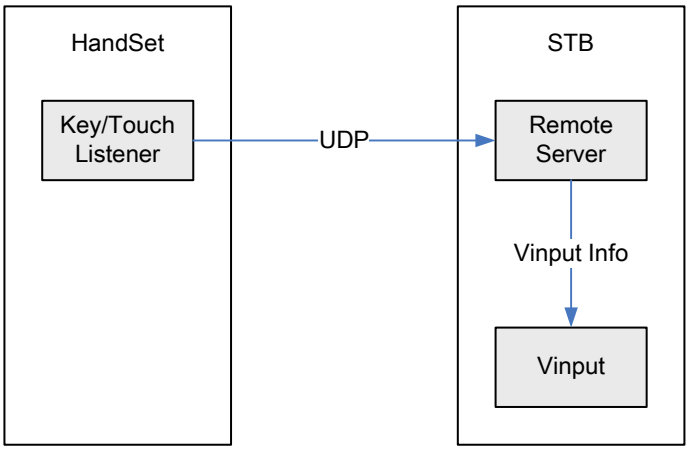


Mirror STB 侧编码发送与手持设备侧接收解码流程部分代码以 so 形式提供。

6.3.2.3 RemoteControl

RemoteControl 使用 UDP 包传输点击，按键信息给 STB 侧，RemoteControl 手持设备与 STB 端交互流程见图 6-6。

图6-6 RemoteControl 交互



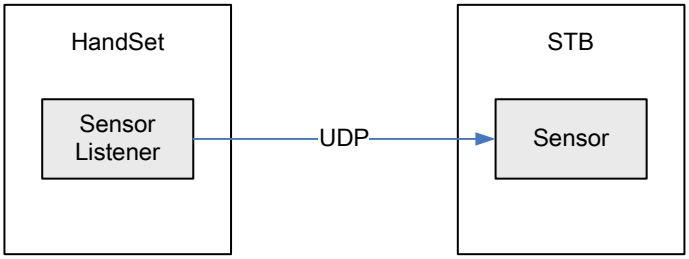
RemoteControl 交互中使用私有协议。

RemoteControl STB 服务器端默认端口 8822。

6.3.2.4 Sensor

Sensor 通过 UDP 向指定端口发送数据包，数据包中的体感类型根据手持设备支持选择相应类型。交互流程如图 6-7 所示。

图6-7 Sensor 交互

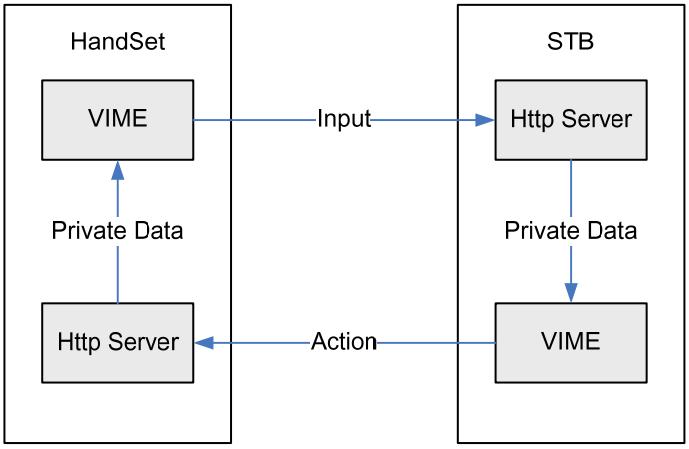


Sensor STB 服务端默认端口 10021。

6.3.2.5 VIME

VIME 通过手持设备与 STB 两侧的 HTTP Server 传输 XML 进行交互，VIME 手持设备与 STB 端交互流程见图 6-8

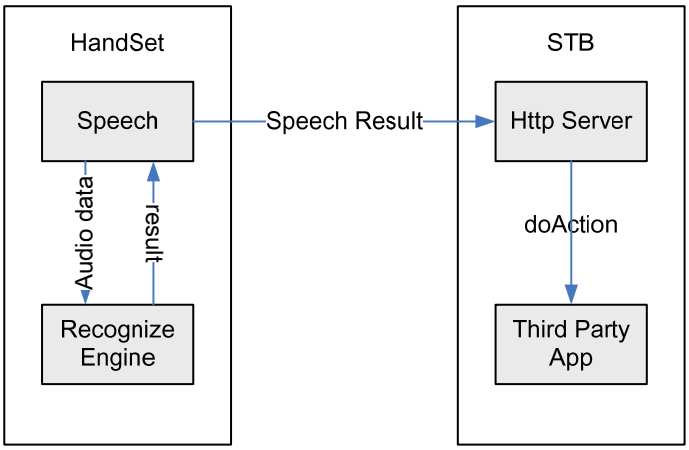
图6-8 VIME 交互



6.3.2.6 Speech

Speech 在手持设备侧通过因特网与语音识别引擎交互后得到语义识别结果，然后将语义识别结果发送到 STB 侧的 HTTP Server，从而完成对 STB 的语音控制。Speech 手持设备与 STB 端交互流程见图 6-9。

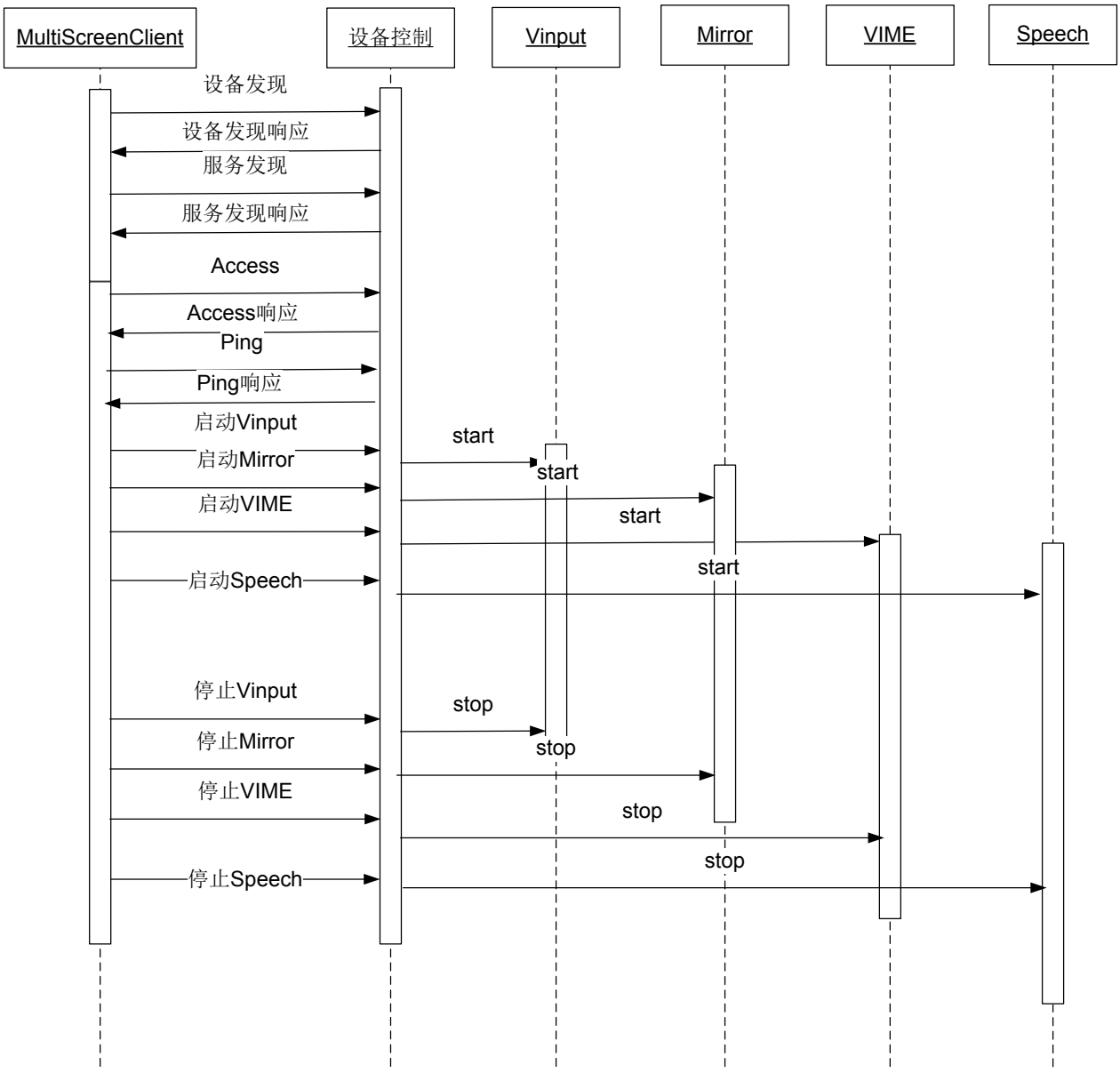
图6-9 Speech 交互



6.3.2.7 运行流程

多屏客户端与 STB 服务端的简要运行流程见图 6-10。

图6-10 整体运行流程图



6.4 开发指引

说明

本章节以 Hi3798MV100 为例，其他芯片类似。

6.4.1 整体说明

不同于其它接口交付组件，HiMultiScreen 作为应用层组件整体交付，并在应用层开放源码，源码路径如下：

- STB 端：
  device\hisilicon\bigfish\hidolphin\component\himultiscreen\android\packages\apps
- 手持设备端：
  device\hisilicon\bigfish\hidolphin\component\himultiscreen\handset_software\MultiScreen

由于相关协议交互(Mirror，VIME，RemoteControl，Sensor)实现部分以 so 形式交付，客户使用 HiMultiScreen 组件的工作量主要集中在 UI 定制化开发。后续章节中主要对多屏应用编译安装方法与 UI 相关代码进行介绍，支持客户快速熟悉相关开源代码，完成 UI 定制工作。

6.4.2 STB 端的编译与安装

STB 端 HiMultiScreen 相关应用在 Android 系统全编译时默认编译。镜像中默认包含 HiMultiScreen 组件。

6.4.2.1 增量编译

用户对应用进行修改后，也可以进行增量编译。首先确认已经按照要求搭建好了服务器编译环境，并完成一次 Android 系统全编译。以 MultiScreenServer 为例，在路径 device\hisilicon\bigfish\hidolphin\component\himultiscreen\android\packages\apps\MultiScreenServer 下执行 mm 命令，后续将在\out\target\product\Hi3798MV100\system\app 下生成 MultiScreenServer.apk。

6.4.2.2 部署

Apk 编译后，有两种部署方式。

- 生成镜像，直接烧写
- 使用 adb push 命令安装 apk

下面假设 STB IP 为 10.161.179.2，MultiScreenServer.apk 在 PC 上的路径为 H:\MultiScreenServer.apk

```
adb connect 10.161.179.2

adb remount

adb push H:\MultiScreenServer.apk system/app
```

6.4.3 STB 端应用

STB 端应用包含 MultiScreenServer，VIME 两个应用。

6.4.3.1 MultiScreenServer

MultiScreenServer 主要包含 MultiScreenService。

- MultiScreenService：MultiScreenServcie 在系统开机时启动，完成 MultiScreen 初始化并将 STB 作为 UPnP 根设备加入到局域网。并在后续接受底层回调启动 VIME 与 Speech。该部分没有界面

6.4.3.2 VIME

应用代码实现一个虚拟输入法注册入 STB 端，接受对应客户端消息完成输入。该部分代码只涉及一个提示界面，消息处理逻辑固定，建议客户不做修改。

6.4.4 STB 端定制开发

6.4.4.1 界面定制

该部分定制集中在语音显示界面 SpeechDialog.。该 SpeechDialog 继承自 Dialog，用户可在该代码基础上替换资源文件对界面进行定制。Android UI 开发相关请参考：http://developer.android.com/guide/components/index.html

6.4.4.2 Mirror 传输格式配置

Mirror 支持 JPEG / H264 两种格式传输，可以通过修改位于 himultiscreen/android/packages/apps/MultiScreenServer/res/raw 目录下的 Mirror 传输格式配置文件 compath264list.xml 配置。当 compath264list.xml 文件中包含的手持设备接入多屏服务时，多屏 Mirror 采用 H264 格式传输图像，非 compath264list.xml 文件中的设备接入多屏服务时，多屏 Mirror 则采用 JPEG 格式传输。假如需要配置手持设备华为 Mate 1 的 Mirror 传输格式为 H264，则在文件 compath264list.xml 中添加华为 Mate 1 的相关信息作为一个设备节点。

- 设备节点说明

  在文件 compath264list.xml 中，每一个手持设备的信息由一个设备节点表示，设备节点如下所示：

```
<device>
    <buildModel>HUAWEI MT1-U06</buildModel>
    <sdkVersion>4.1.2</sdkVersion>
</device>
```

  其中：

  - buidlModel 代表设备型号，例如华为 mate 1 的型号为 HUAWEI MT1-U06。
  - sdkVersion 代表设备的手持设备的 android 版本号。

- Mirror 传输格式两种配置方式：
  - 修改兼容性配置文件 compath264list.xml，用 adb push 到 `data/data/com.hisilicon.multiscreen.server/cache/`路径下。
  - 修改 compath264list.xml，编译镜像并烧写镜像。

6.4.5 客户端的编译和安装

6.4.5.1 总体要求

客户端编译环境搭建请参考“环境配置”章节。

客户端的编译和安装总体要求如下：

- 运行系统
  客户端程序必须安装在 Android2.3 及其以上系统的手持设备侧。
- 编译要求
  客户端程序的编译环境的 SDK 配置要求是基于 Android 4.0 的 SDK。

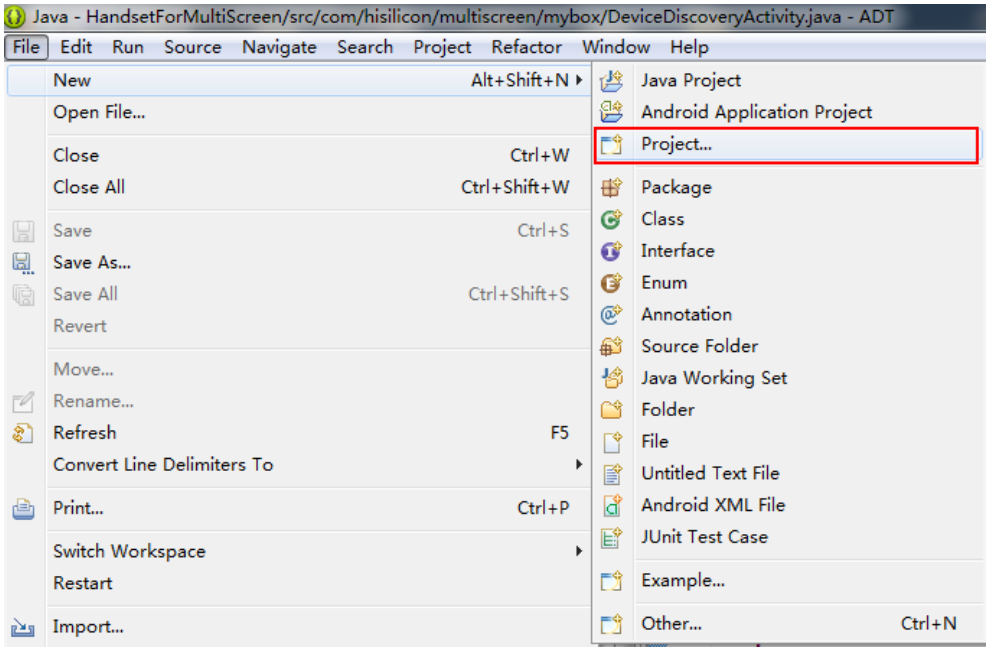
6.4.5.2 客户端使用

在此以 Eclipse 4.2.0, ADT 21.0.0 为例来说明客户端如何编译安装：

步骤 1 在 SDK 源码目录下，找到 device\hisilicon\bigfish\hidolphin\component\himultiscreen\handset_software\MultiScreen 目录。

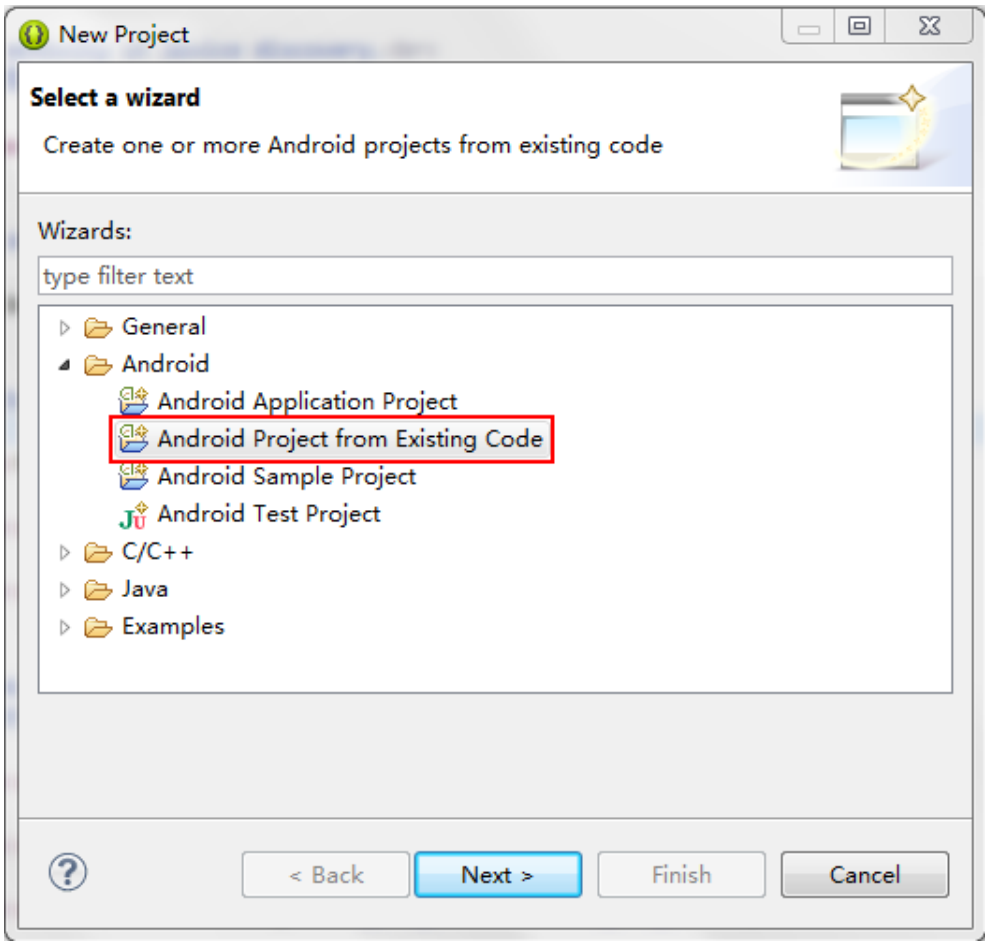
步骤 2 新建 Android 工程，如图 6-11 所示，选择 New->Project。

图6-11 新建 Android 工程



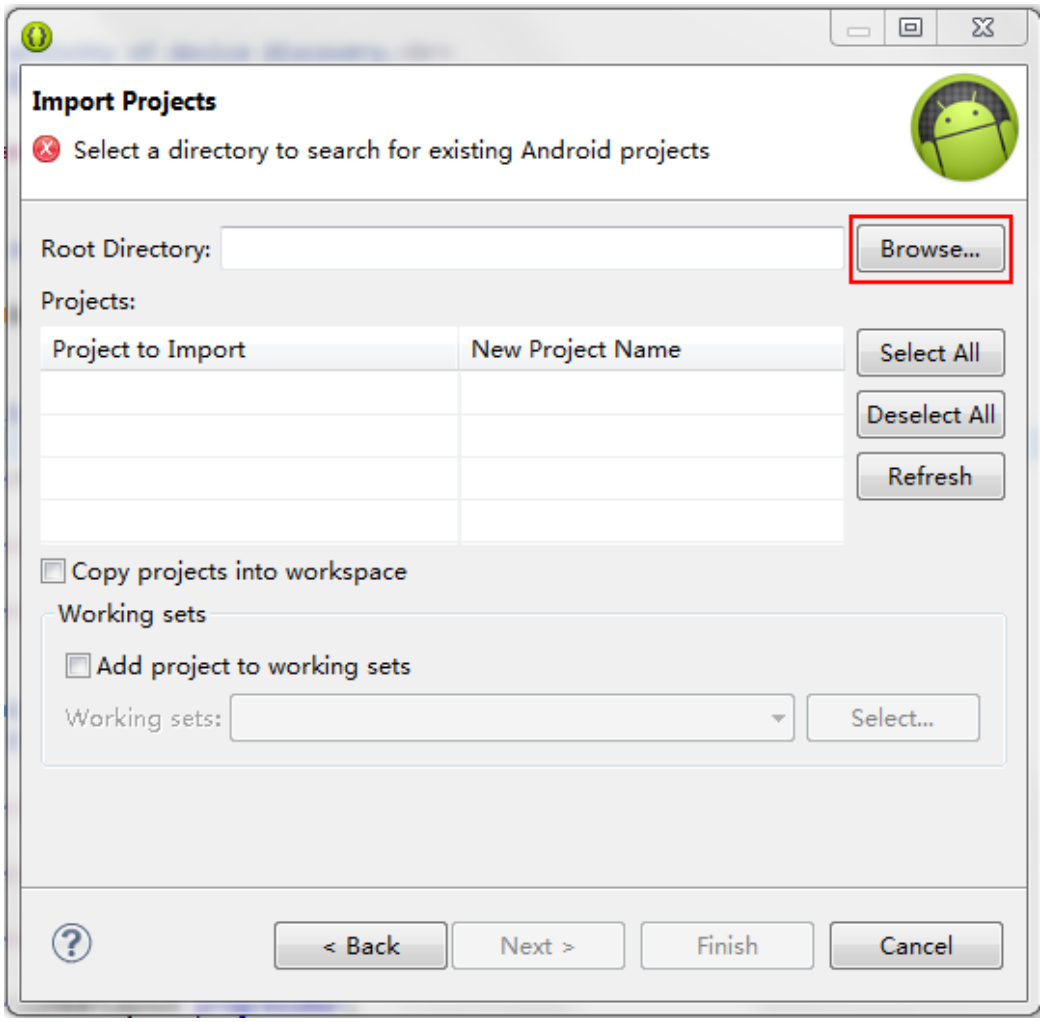
步骤 3 从现有工程中创建，如图 6-12 所示，选择 Android Project from Existing Code，点击“Next”。

图6-12 选择 Android Project from Existing Code



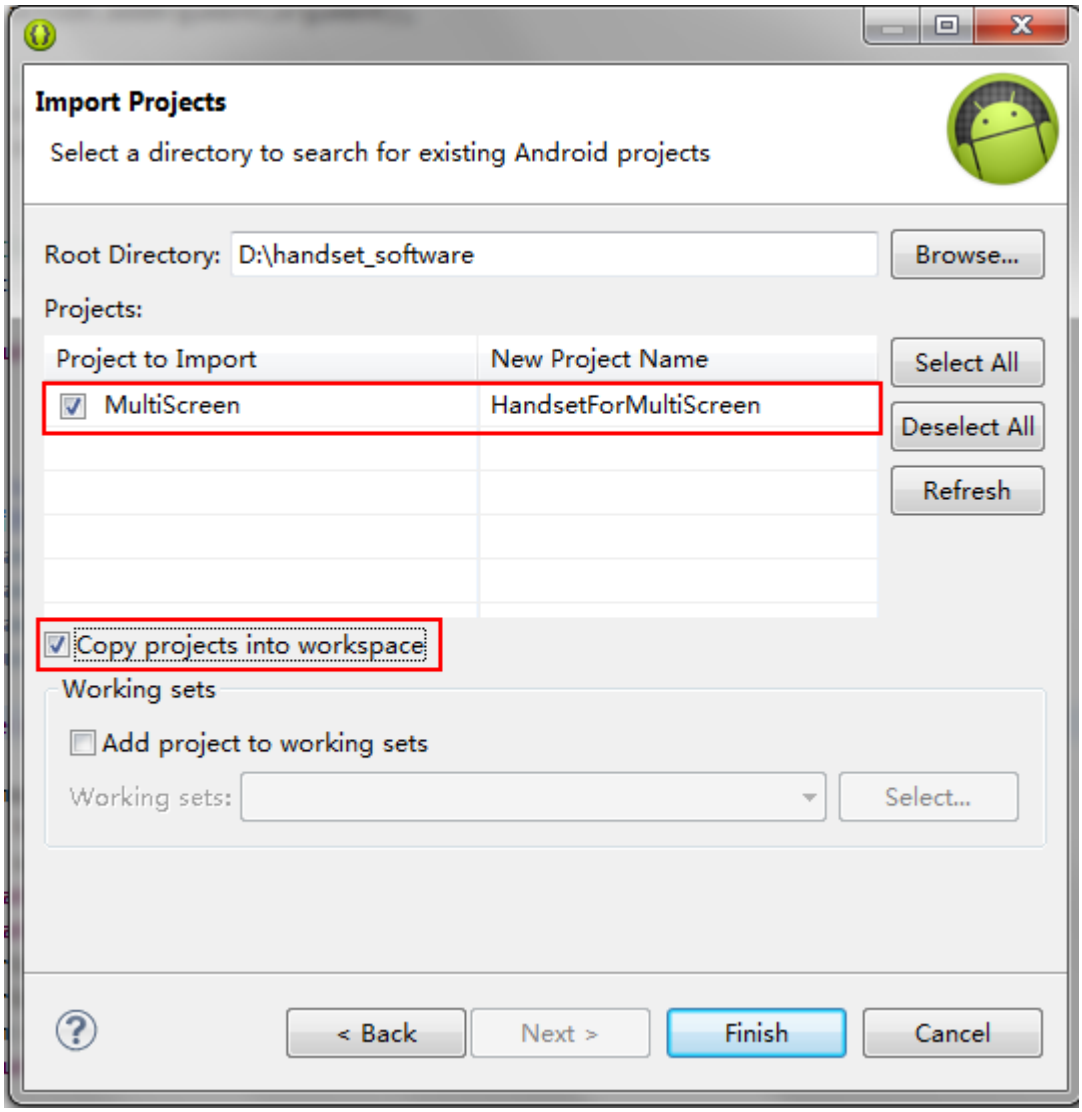
步骤 4 选择工程文件导入，如图 6-13 所示，点击“Browse”，选择解压出来 MultiScreen 工程。

图6-13 导入工程文件



步骤 5 导入工程设置，图 6-14 所，将 Projects 中的工程名选中，同时选中“Copy projects into workspace”选项。

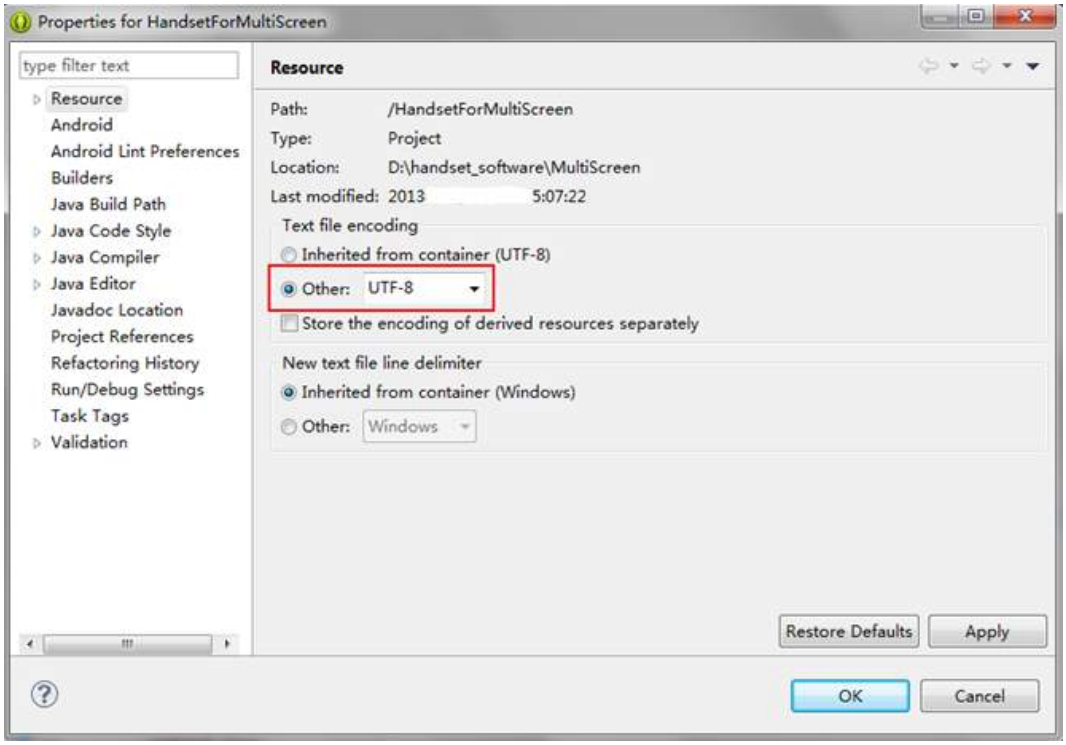
图6-14 导入工程设置



步骤 6 工程编码格式设置。

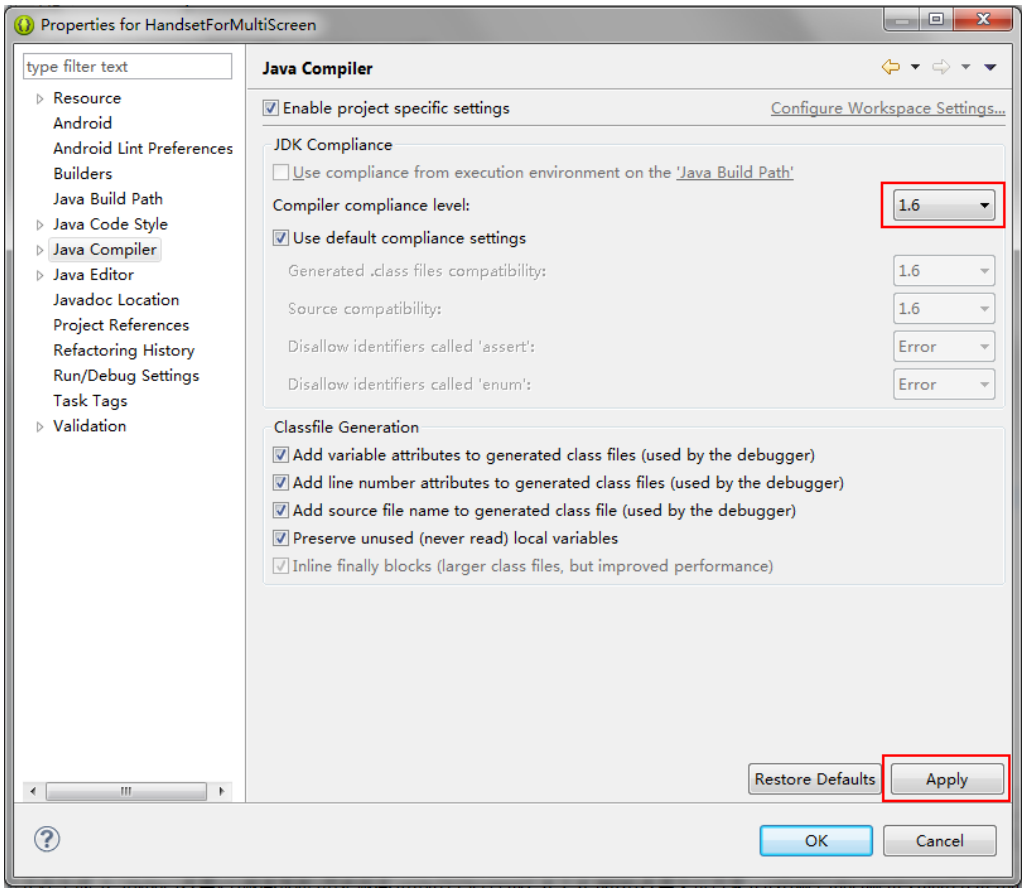
1. 点击工程，选择 Project-->Properties，如图 6-15 所示。
2. 打开后，选择 Resource 对应的页面，将 Text file encoding 的 Other 设置为 UTF-8。
3. 点击“Apply”按钮，使之生效。
4. 生效后，点击“OK”，退出。

图6-15 编码格式设置



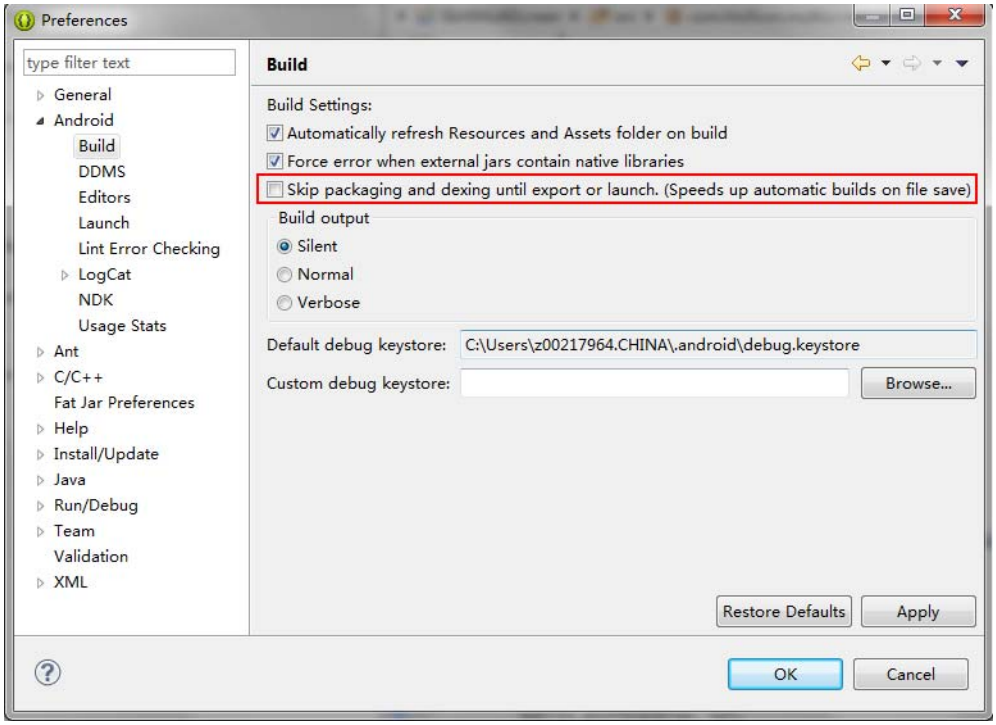
步骤 7 JDK 版本设置，Project-->Properties，打开后，如图 6-16 所示，选择 Java Compiler 对应的页面，将 JDK Compliance 设置为 1.6。

图6-16 JDK 版本设置



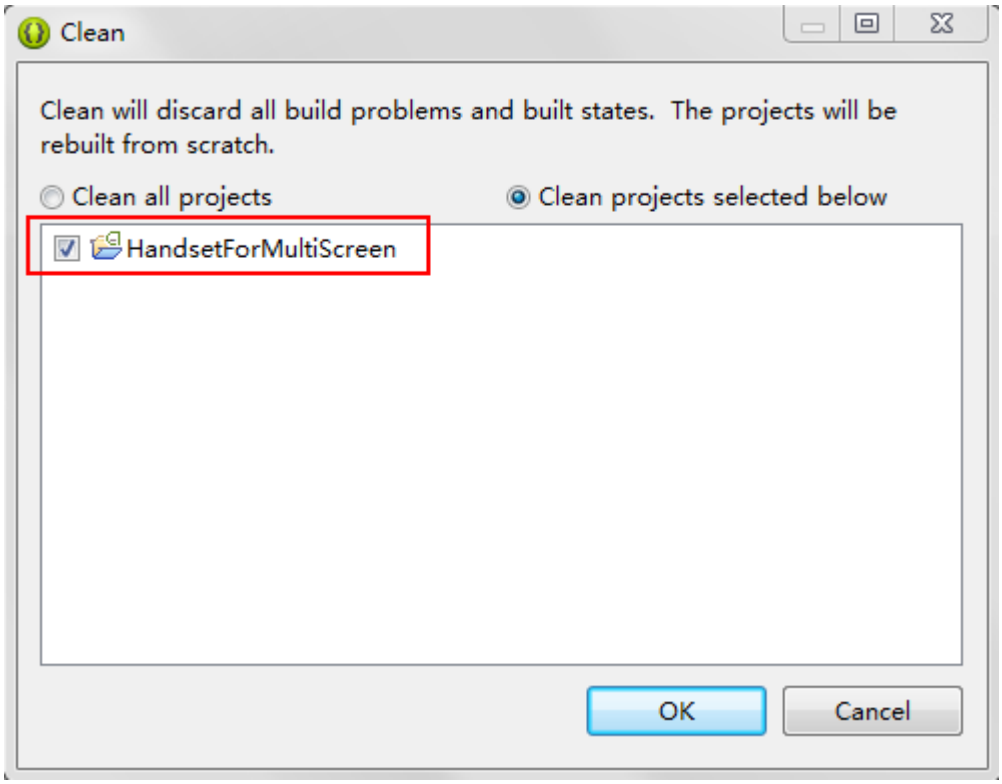
步骤 8 APK 生成时机设置，点击 Window->Preferences，打开后，选择 Android 下的 Build，如图 6-17 所示，确保 Skip packaging and dexing until export or launch 不被选中。

图6-17 APK 生成设置



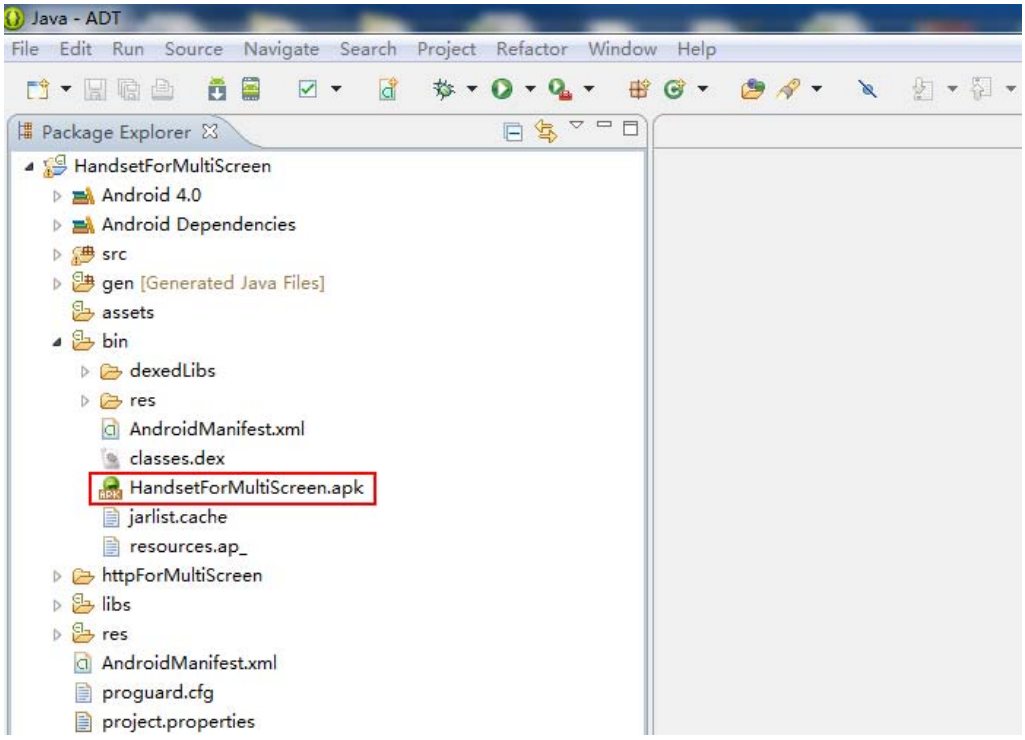
步骤 9 编译工程，选择 Project-->Clean，打开后，如图 6-18 所示，确保所有勾选项和图 6-18 保持一致，之后点击“OK”，进行编译。

图6-18 工程编译



生成 apk 文件，步骤 9 编译完成后，生成的 apk 文件所在目录如图 6-19 所示，生成 HandsetForMultiScreen.apk。

图6-19 APK 文件生成



步骤 10 安装，假设 HandsetForMultiScreen.apk 放在 H 盘的根目录下，则执行如下命令：

```
adb install H:\HandsetForMultiScreen.apk
```

如果原本已经安装，可执行如下命令：

```
adb uninstall com.hisilicon.multiscreen.mybox
```

进行卸载，然后再执行安装。

----结束

6.4.6 客户端应用

客户端应用安装后名为 MyBox（我的盒子），主要包含欢迎介绍界面，设备发现界面，Mirror 界面，设置界面，RemoteControl 界面，RemoteAPP 界面。

6.4.6.1 界面类

界面代码集中在 src\com\hisilicon\multiscreen\mybox 目录下。

- 欢迎介绍界面

  该部分主要包含 GuideActivity.java，GuideGallery.java。在应用首次启动时由设备发现界面弹出，给出多屏应用使用方法提示，用户可通过滑动浏览提示贴图。

- 设备发现界面

该部分主要包含 DeviceDiscoveryActivity.java。为应用启动后的默认跳转界面，在该界面中将定时刷新可用的 STB 端设备，同时也可以手动发起搜索。搜索完成或者接到 STB 服务端通知后，显示搜索到的设备的信息列表，供用户选择和连接

- Mirror 界面

  该部分主要包含 MultiScreenActivity.java。由设备发现界面选择特定的 STB 多屏服务端后跳转，Mirror 界面为多屏应用的主功能界面，在该界面 onCreate 时将对网络状态检测，Mirror，Touch，Sensor，VIME 等模块进行初始化。实现 Mirror，RemoteControl 中 Touch 事件监听传送，并提供设置界面，RemoteControl 界面和 RemoteAPP 界面的跳转入口。

- RemoteAPP 界面

  该部分主要包含 RemoteAppActivity.java。该界面获取机顶盒 APP 应用的标志与应用名，按列表显示，并可以启动机顶盒某个 APP。

- RemoteControl 界面

  该部分主要包含 RemoteControlActivity.java。该界面为遥控器界面，显示遥控器的按钮。

- 设置界面

  该部分主要包含 SettingActivity.java。该界面显示 Gsensor 以及虚拟输入法的设置开关。

- 虚拟输入法输入界面

  该部分主要包含 ContentInputActivity.java。路径：src\com\hisilicon\multiscreen\vime\，为远程输入界面，调用手持设备输入法，进行输入。

- BaseActivity

  BaseActivity.java 多屏应用 Activity 的基类，负责完成控制界面一些共同的特性。

6.4.6.2 服务类

客户端服务类是 Service 的子类实现，主要负责客户端后台业务逻辑实现，这部分代码建议客户不要修改。

- MultiScreenControlService

  代码 src\com\hisilicon\multiscreen\mybox\MultiScreenControlService.java，承载 UPnP 中的 controlPoint 类，完成设备发现，接入控制，网络状态检测以及 mirror 控制的逻辑。

- SensorService

  代码 src\com\hisilicon\multiscreen\gsensor\ SensorService.java，启动手持设备体感服务，获取客户端的体感数据发送到 STB 端。

- VIMEClientControlService

  代码 src\com\hisilicon\multiscreen\vime\ VIMEClientControlService.java，虚拟输入法控制服务，接收，发送、处理虚拟输入法的控制消息，与 STB 端状态同步。

6.4.7 客户端定制开发

6.4.7.1 界面定制

客户可对上述介绍的界面类进行界面定制。Android UI 开发相关请参考：
http://developer.android.com/guide/components/index.html

如果修改了界面，需要对欢迎介绍界面的介绍图片进行刷新。

6.4.7.2 定制远程控制键值

建议客户对 RemoteControl 界面根据客户所使用的红外遥控器进行定制美化。原始界面是对公版海思遥控器键值进行对应。代码中已经提供了公版遥控器所使用的相关键值，如果客户在 STB 端有扩展键值，可参考下面方法进行添加。

在 KeyInfo 类中添加所需支持键值。路径 src\com\hisilicon\multiscreen\protocol\message\KeyInfo.java

```
public static final int KEYCODE_XX = 0xXX;
```

在 RemoteControlActivity.java 中添加 Button，并在其点击事件监听中添加处理

仅支持短按处理：

```
if (event.getAction() == MotionEvent.ACTION_DOWN)
{
   //修改Button贴图
}
else if (event.getAction() == MotionEvent.ACTION_UP)
{
   //修改Button贴图
   mKeyboard.sendDownAndUpKeyCode(KEYCODE_XX);
}
```

支持长按处理：

```
if (event.getAction() == MotionEvent.ACTION_DOWN)
{
if (mStartTime == 0)
   {
       …
       mStartTime = SystemClock.elapsedRealtime();
       if (mStartTime != 0)
       {
sendLongPressKeyRequest(KeyInfo.KEYCODE_XX,
                                KeyInfo.KEY_EVENT_DOWN);
       }
   }
   else
   {
        LogTool.e("wrong order");
```

```
        mStartTime = 0;
sendLongPressKeyRequest(KeyInfo.KEYCODE_XX,
                            KeyInfo.KEY_EVENT_UP);
    }
}
else if (event.getAction() == MotionEvent.ACTION_UP)
{
   …
   mStartTime = 0;
sendLongPressKeyRequest(KeyInfo.KEYCODE_XX,
KeyInfo.KEY_EVENT_UP);
}
else if (event.getAction() == MotionEvent.ACTION_MOVE)
{
if (mStartTime != 0)
   {
      if (mCount < 20)
      {
         mCount++;
      }
      else
      {
         sendLongPressKeyRequest(KeyInfo.KEYCODE_XX,
                                 KeyInfo.KEY_EVENT_DOWN);
         mCount = 0;
      }
    }

}
```

6.4.7.3 客户端应用图标与应用名称修改

准备定制的 ICON.png，替换 res\drawable，res\drawable-hdpi，res\ drawable-ldpi，res\drawable-mdpi 中的 icon.png 图片。

修改 res\values\string.xml，res\values-en-rUS\string.xml，res\values-zh-rCH\string.xml 中 <string name="app_name">XXXX</string>

6.5 调试指引

6.5.1 日志

使用 Android 原生日志系统 Logcat。用户调试时可以在 DDMS 中使用类名作为 TAG 进行日志过滤。

6.5.2 连接

当客户端无法连接 STB 端时，建议首先在 STB 端使用 PING 命名测试是否正常连接手持设备。

7 HiTranscoder

7.1 概述

HiTranscoder 模块以 SDK 的硬件能力为基础，提供包括编码、同步、封装、缓冲、协议等多个模块，保证从电脑/手持设备上可以获取并播放来自于单板侧的媒体数据。模块可以提供多用户，多封装的数据访问。

根据视频和音频编码码率不同，内存使用的情况也不尽相同。其中起决定作用的是视频码率，对应关系如下：

- 当视频码率为 256Kb 时，内存使用 560KB。
- 当视频码率为 512Kb 时，内存使用 560KB。
- 当视频码率为 1Mb 时，内存使用 1060KB。
- 当视频码率为 2Mb 时，内存使用为 2MB。

网络带宽占用情况同音视频码率有关，相当于视频码率和音频码率相加。对应关系基本如下：

- 当视频码率为 256Kb 时，音频 48KHz，带宽需求为 280KB。
- 当视频码率为 512Kb 时，音频 48KHz，带宽需求为 536KB。
- 当视频码率为 1Mb 时，音频 48KHz，带宽需求为 1MB。
- 当视频码率为 2Mb 时，音频 48KHz，带宽需求为 2MB。

CPU 占用情况基本稳定，例如在 Hi3798MV100 单板上，全业务 CPU 占用 3%～8%，为 30MHz～80MHz 主频。

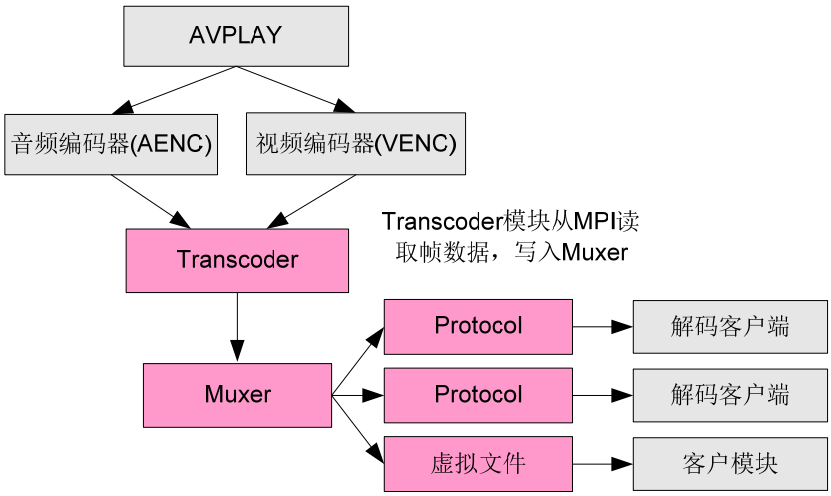
7.2 功能特点

HiTranscoder 模块的主要特性如下：

- 支持 FLV 及 TS 的文件本地封装。
- 客户可以根据其他协议来扩展动态库，添加客户定制功能。
- 支持多路客户端接入协议服务器。

HiTranscoder 模块的功能和结构如图 7-1 所示。

图7-1 HiTranscoder 模块功能结构图



从结构上看，HiTranscoder 可划分为以下 3 个模块：

- Transcoder
  用来处理原始编码数据
- Protocol
  用来处理协议服务器的响应，数据的组织和对源端控制的导入。
- Muxer
  对媒体数据进行文件封装或协议封装

图7-2 HiTranscoder 模块数据流程图

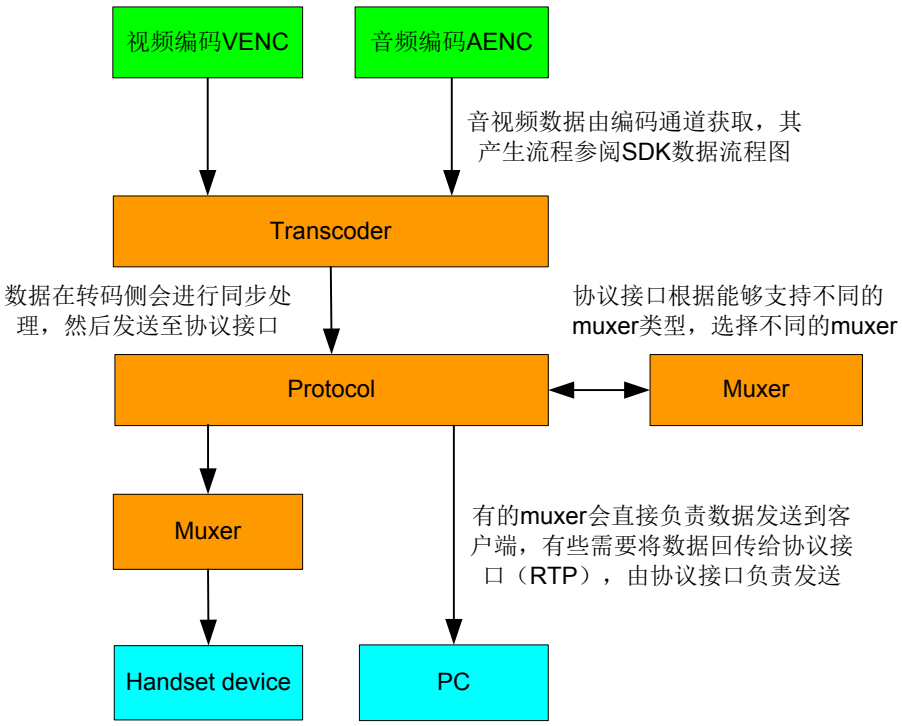


图7-3 HiTranscoder 模块控制流程图

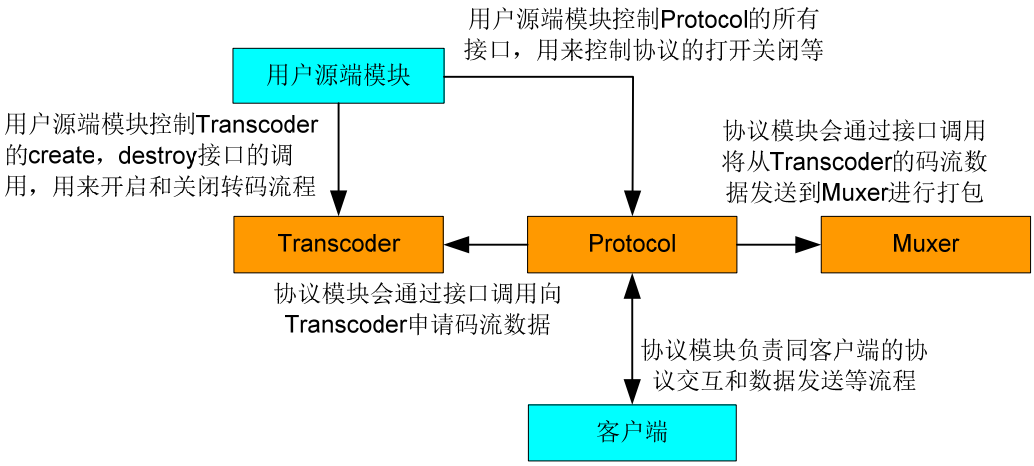
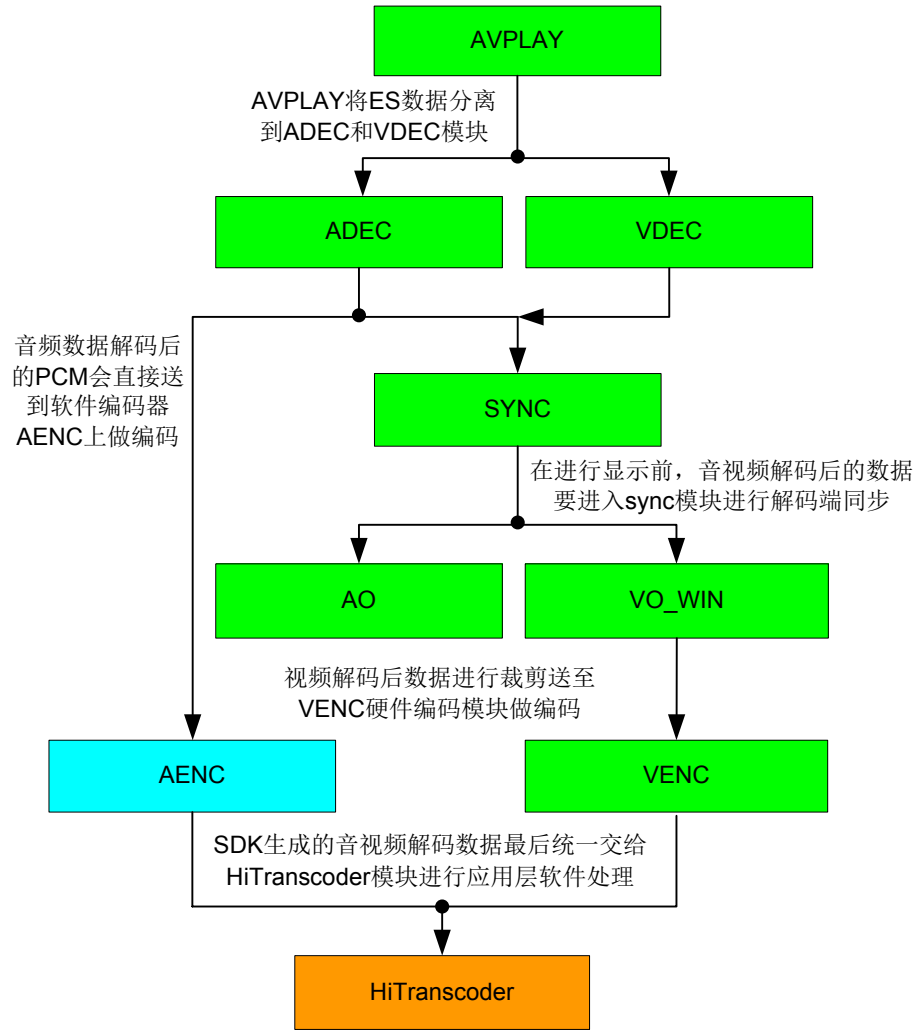


图7-4 HiTranscoder 媒体数据 SDK 整体流程图



7.3 重要概念

7.3.1 Transcoder 概念

Transcoder 是转码的数据处理模块，主要负责通过软件建立从解码侧到编码侧数据通路，同时负责同步，缓冲等音视频编码侧工作。

7.3.2 Protocol 概念

Protocol 是转码模块的协议处理模块，负责在单板上建立一个服务器供手持设备或电脑等客户端接入，保证在转码后的媒体数据按照标准流媒体协议进行通信发送，能够被手持设备或者电脑上的播放器接收并可以正常播放，播放过程中不存在音视频不同步问题，同时尽量降低服务器和播放器间的播放延时。

7.3.3 Muxer 概念

Muxer 是转码模块的数据封装模块，主要负责从 Transcoder 的裸数据到 Protocol 可以使用的数据封装格式的处理。

7.4 开发指引

7.4.1 如何使用模块

HiTranscoder 模块的接口主要包括 Transcoder、Protocol、Muxer 三个部分，需要三个部分配合使用，如果用户不用自己实现 Protocol 的实体，那么 Muxer 模块不用使用。

7.4.1.1 HiTranscoder 模块的初始化、去初始化

编程指导

在调用 HiTranscoder 的其他接口之前，包括客户端的接入及协议启动之前，必须先初始化 HiTranscoder 模块，当不再使用 HiTranscoder 模块时，需要将 HiTranscoder 模块进行去初始化。

去初始化 HiTranscoder 模块会释放 HiTranscoder 模块占用的各种资源，例如：

- 分配的编码通道及虚拟窗口。
- 启动协议所分配的端口及守护线程。

HiTranscoder 的初始化需要包括如下三个接口：

- HI_Transcoder_Init
- HI_Protocol_Init
- HI_Muxer_Init

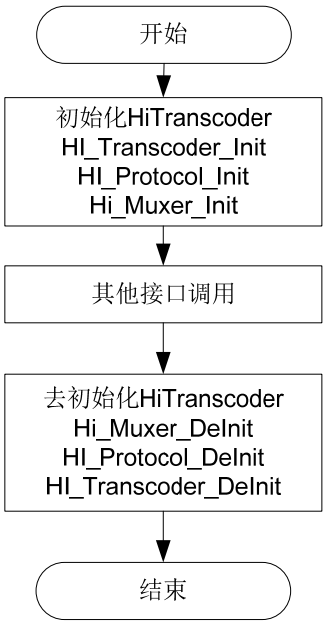
HiTranscoder 的去初始化需要包含如下三个接口：

- HI_Transcoder_DeInit
- HI_Protocol_DeInit
- HI_Muxer_DeInit

工作流程

初始化模块和去初始化模块流程如图 7-5 所示。

图7-5 初始化模块和去初始化模块流程图



7.4.1.2 Transcoder 句柄的创建与销毁

完成 HiTranscoder 的初始化后，需要创建 Transcoder 的句柄，用来获取从 AENC 或 VENC 产生的数据，此接口可以完成从 AENC 或者 VENC 编码数据进行同步后输出到 Protocol 模块的功能。

进行 Transcoder 句柄创建的前置条件是：

- 播放解码的 AVPLAY 初始化和创建流程已经完成。

  AVPLAY 可以处于播放或者未播放状态；该流程使数据源的音视频解码播放，用户可参考第“Sample 使用说明“小节提供的 sample 源码，同时参照平台 SDK 的.h 头文件来理解这一过程。播放的码流必须包括音视频，目前尚不支持对单音频或视频的码流进行转码播放。

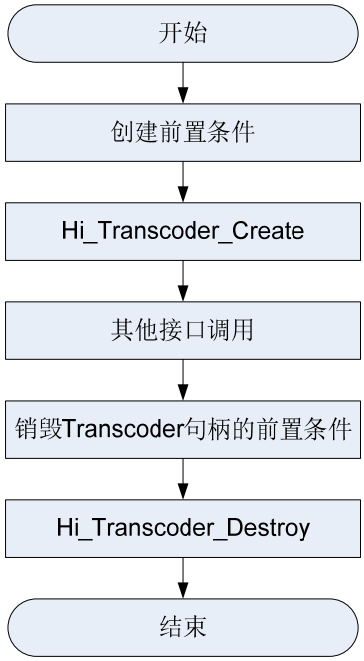
- HiTranscoder 的初始化流程已经完成。

  销毁 Transcoder 的句柄，可以释放用来储存流数据和各种软件标志的内存，此接口的前置条件是：Protocol 的句柄已经同 Transcoder 句柄解绑定。

工作流程

Transcoder 句柄的创建和销毁流程如图 7-6 所示。

图7-6 Transcoder 句柄的创建和销毁流程图



7.4.1.3 Protocol 句柄的创建与销毁

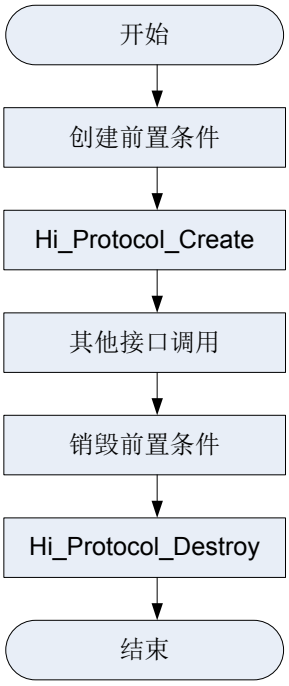
Protocol 句柄的创建和销毁同 Transcoder 句柄的创建和销毁没有必然关系，同解码源端的解码流程也没有关系，此句柄创建的功能是建立协议服务器并启动各类处理守护线程。在该句柄创建时需要使用 muxer 的相关接口，所以前置条件需要完成第“7.4.1.1 HiTranscoder 模块的初始化、去初始化”小节的初始化流程。

- 进行 Protocol 句柄创建的前置条件是：HiTranscoder 的初始化流程已经完成。
- 进行 Protocol 句柄销毁，可以销毁系统使用的协议服务器，守护线程退出，前置条件是：Protocol 的句柄已经同 Transcoder 句柄解绑定。

工作流程

Protocol 句柄的创建和销毁流程如图 7-7 所示。

图7-7 Protocol 句柄的创建和销毁流程图



7.4.1.4 Protocol 句柄的绑定和解绑定

Protocol 句柄在绑定过程中，transcoder 模块的一个句柄会与 Protocol 模块的一个句柄绑定，在 Protocol 的句柄的绑定，Transcoder 会发生和 Protocol 的数据交互，通过绑定的 transcoder 的句柄，Protocol 可控制 transcoder 的开启和关闭，以及从 transcoder 获取数据读取句柄从而读取音视频 ES 数据。

当前默认支持 RTSP 和 HTTP 两种协议，但用户也可定制私有协议，通过实现文件 include/hi_proto_intf.h 中结构体 struct tagHI_ProtocolInfo_S，以 lib*.so 的格式通过接口 HI_S32 HI_Protocol_RegistProtocol(const HI_CHAR* pLibProtocolName);注册私有协议。

进行 Protocol 句柄绑定的前置条件：

- HiTranscoder 的 Protocol 句柄创建完成。
- HiTranscoder 的 Transcoder 句柄创建完成。

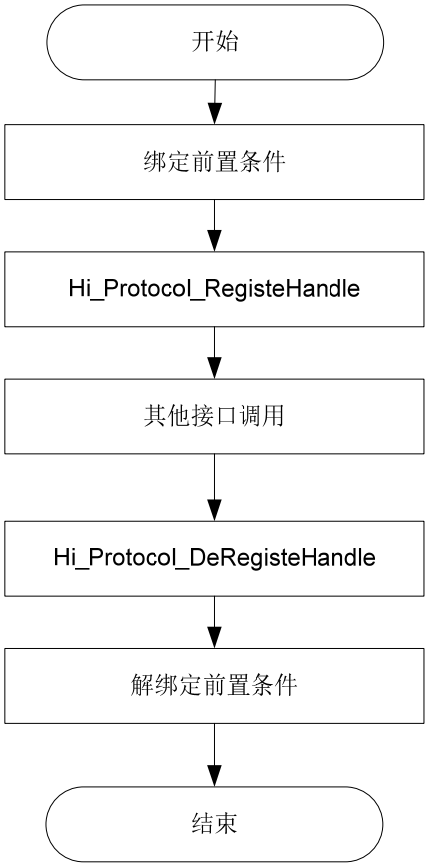
Protocol 句柄的解绑定，在解绑定过程中，会完成从 Protocol 模块中将 Transcoder 的数据读取句柄分离的过程。

进行 Protocol 句柄解绑定的前置条件：HiTranscoder 的 Protocol 句柄已经调用过解绑定接口。

工作流程

Protocol 句柄的绑定和解绑定流程如图 7-8 所示

图7-8 Protocol 句柄的绑定和解绑定流程图



7.4.1.5 Muxer 处理流程

Muxer 处理模块是用于对数据进行封装的，封装后的数据可以直接存为本地文件，也可以通过 Protocol 发送到网络，现在提供的 sample_protocol 代码就是将 HiTranscoder 的转码数据输出到本地文件中。

当前默认封装格式有 RTP, FLV, TS, ES；但支持用户注册私有的数据封装格式，用户可依照 Hi_muxer_intf.h 提供的 HI_MuxerInfo_S 结构体，实现可注册的私有 Muxer，以 lib*.so 的格式通过接口 HI_S32 HI_Muxer_RegistMuxer(const HI_CHAR* pLibMuxerName)进行注册使用。

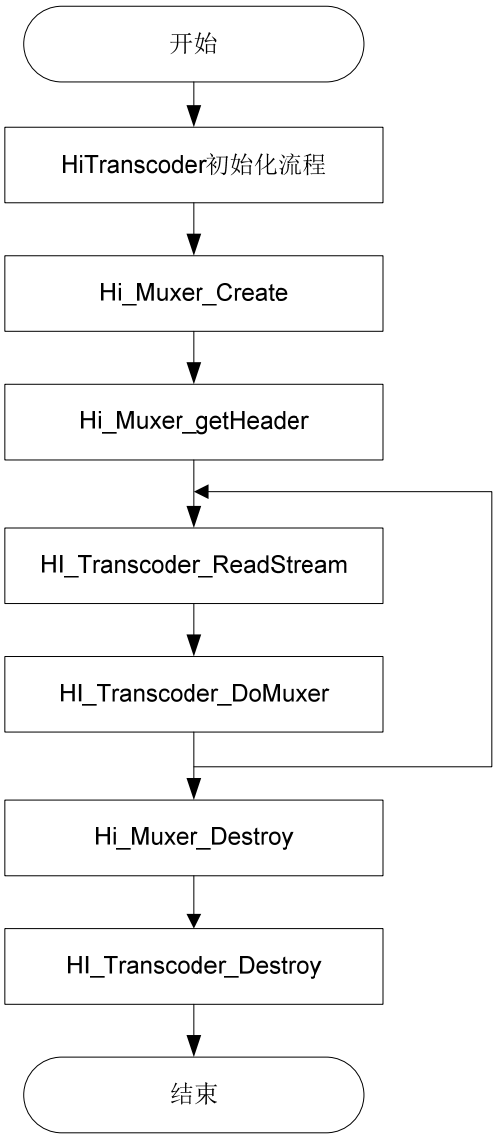
Muxer 模块处理流程的前置条件：

- HiTranscoder 的初始化过程完成，如“7.4.1.1 HiTranscoder 模块的初始化、去初始化”节描述。
- 完成 TranscoderRegistHandle 注册数据读取句柄。

工作流程

Muxer 处理流程如图 7-9 所示

图7-9 Muxer 处理流程图



7.4.1.6 Transcoder 数据处理流程

在 HI_Transcoder_RegisterHandle 调用以后，将 Transcoder 的数据读取出来，把读取出来的 raw 数据封装成其他格式，利用 Protocol 发送至用户的播放器。

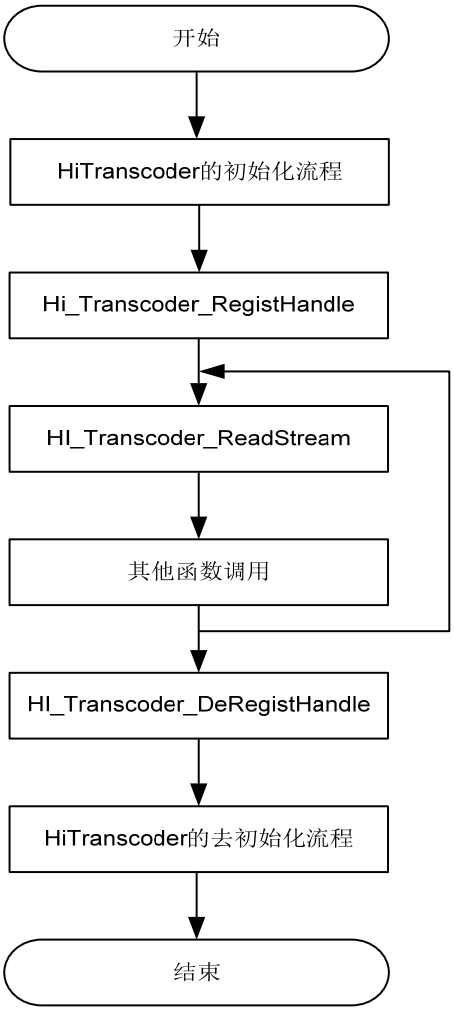
前置条件：

- HiTranscoder 的初始化过程完成，如“7.4.1.1 HiTranscoder 模块的初始化、去初始化”节描述。
- 调用 Hi_Transcoder_create 接口完成 transcoder 句柄的建立。

工作流程

Transcoder 数据处理流程如图 7-10 所示

图7-10 Transcoder 数据处理流程图



7.4.2 配置运行环境

在 Android 和 Linux 平台 HiTranscoder 组件随各自 SDK 发布。

7.4.2.1 配置服务端运行环境

Android 版本

Android 版本配置步骤如下：

步骤 1 Android 平台 SDK 编译

在HiSTBAndroidVX00R001C00SPCXXX(Android) SDK下，请参见“2 开发环境配置”进行全编译，编译成功后按要求烧写单板。

步骤 2 在单板的 com 口命令行模式下执行转码相关 sample：

- TsPlay Sample

```
sample_transcoder_tsplay file=fileNameDir f=25 w=320 h=240 pro=rtsp
b=1024*1024
```

其中fileNameDir为文件路径名，比如/data/app/file.ts，必须是TS文件。

- DvbPlay Sample

```
sample_transcoder_dvbplayer freq=610 srate=6875 qam=64 f=25 w=320
h=240 pro=rtsp b=1024*1024
```

请客户根据自身频点的不同切换 freq=610 的值 eg. freq=235。

- pipplayer Sample

```
sample_transcoder_pipplayer freq=610 srate=6875 qam=64 f=25 w=320
h=240 pro=rtsp b=1024*1024
```

请客户根据自身频点的不同切换 freq=610 的值 eg. freq=235。

- OsdPlayer Sample

```
sample_transcoder_osdplayer f=25 w=320 h=240 pro=rtsp b=1024*1024
```

- HlsPlayer Sample

```
sample_transcoder_hlsplayer file=fileNameDir w=1280 h=720
b=4*1024*1024
```

其中 fileNameDir 为文件路径名，比如/data/app/file.ts，必须是 TS 文件。

----结束

Linux 版本

Linux SDK 下的 source/component/hitranscoder 目录下为转码的相关文件：

- include：转码接口 API 头文件。
- lib：转码相关动态库。
- handset_software：客户端相关代码

在运行应用程序前，请按如下步骤设置运行环境：

Linux 版本配置步骤如下：

步骤 1 Linux 平台 SDK 编译。

在 HiSTBLinuxV100R00XC00SPCXXX (Linux) SDK 下执行

```
make menuconfig
```

在 menuconfig 中确保 hitranscoder 模块已被选中并成功保存退出；然后按照 SDK 根目录下文档《install_notes(chs/eng).txt》进行全编译，编译成功后按要求烧写单板。

步骤 2 编译转码相关 sample

在 SDK 版本（HiSTBLinuxV100R00XC00SPCXXX）根目录下执行 Linux 命令

```
make sample
```

在 SDK_DIR/sample/hitranscoder 文件夹下会生成可执行文件：

```
sample_transcoder_tsplay，sample_transcoder_dvbplayer，
sample_transcoder_pipplayer，sample_transcoder_osdPlayer，
```

`sample_transcoder_hlsplayer`。

步骤 3 使用 tftp 工具下载 sample 到单板

用户自行获取 tftp-server，运行并配置 tftp-server；把步骤 2 中生成的五个 sample 可执行文件使用 tftp 工具下载到单板，在单板的 com 口下，执行：

```
mkdir /root/trans
cd /root/trans
tftp -gr sample_transcoder_tsplay host_IP_addr(pc ip地址)
tftp -gr sample_transcoder_dvbplayer host_IP_addr(pc ip地址)
tftp -gr sample_transcoder_pipplayer host_IP_addr(pc ip地址)
tftp -gr sample_transcoder_osdPlayer host_IP_addr(pc ip地址)
tftp -gr sample_transcoder_hlsplayer host_IP_addr(pc ip地址)
chmod 777 sample_transcoder_*
```

步骤 4 在单板的 com 口命令行模式下，在 sample 文件所在目录下执行转码相关 sample：

- TsPlay Sample

  `./sample_transcoder_tsplay file=fileNameDir f=25 w=320 h=240 pro=rtsp b=1024*1024`

  其中 fileNameDir 为文件路径名，比如/data/app/file.ts, 必须是 TS 文件。
- DvbPlay Sample

  `./sample_transcoder_dvbplayer freq=610 srate=6875 qam=64 f=25 w=320 h=240 pro=rtsp b=1024*1024`

  请客户根据自身频点的不同切换 freq=610 的值 eg. freq=235。
- pipplayer Sample

  `./sample_transcoder_pipplayer freq=610 srate=6875 qam=64 f=25 w=320 h=240 pro=rtsp b=1024*1024`

  请客户根据自身频点的不同切换 freq=610 的值 eg. freq=235。
- OsdPlayer Sample

  `./sample_transcoder_osdplayer f=25 w=320 h=240 pro=rtsp b=1024*1024 band=16`
- HlsPlayer Sample

  `sample_transcoder_hlsplayer file=fileNameDir w=1280 h=720 b=4*1024*1024`

  其中 fileNameDir 为文件路径名，比如/data/app/file.ts, 必须是 TS 文件。

----结束

说明

关于 sample 使用注意事项和参数说明，请参见 0。

Sample 使用说明

Sample 描述如下：

- sample_transcoder_tsplay：TS 文件转码，即服务端播放 TS 文件，客户端可同步转码播放 TS 文件的视频和声音。
- sample_transcoder_dvbplayer：Dvb 播放转码，即服务端播放 Dvb 节目，并支持换台，客户端可同步播放 Dvb 节目，且在服务端换台后，支持续播。
- sample_transcoder_pipplayer：Dvb 画中画播放转码，播放一路全屏可视 Dvb 节目，同时在屏幕左上角以窗口形式播放一路 Dvb 节目，且可换台，客户端可同步转码小窗口 Dvb 节目的音频和视频。
- sample_transcoder_osdplayer：图形层和视频层一同转码，在服务端播放 TS 码流，并同时显示一张图片，客户端可同时转码播放视频和图形，暂时不支持声音的同步转码播放。
- sample_transcoder_hlsplayer：HLS 协议转码支持，在服务端播放 TS 码流，IOS 客户端可实时转码 TS 播放的视频和声音。

注意

在 Android 平台上，上述转码 sample 中 tsplay，dvbplayer，pipplayer，hlsplayer 在运行后，由于默认视频层被图形层遮盖，导致播放的视频图像被隐藏；需要用户手动执行命令以显示或隐藏视频图像。

用户在执行 sample 前，可执行如下命令：

```
echo hide >/proc/msp/hifb0
```

使视频图像显示；

用户在停止 sample 后，可执行如下命令：

```
echo show >/proc/msp/hifb0
```

使视频图像隐藏，图形界面显示。

sample 参数描述如下：

- freq：dvb 播放的频点。(可变，根据实际情况设置)。freq 单位为 MHZ，在频点的带宽内接收 TS 复用流。
- f：视频的帧率。(可变，10/20/25/30)。
- w：视频的宽度。(可变，最小：320，最大：1920，必须是 4 的倍数)。
- h：视频的高度。(可变，最小：240，最大：1080，必须是 4 的倍数)。
- pro：使用服务器的协议。(可变，rtsp/http/local)。此处的 local 协议是指保存 Ts 文件到本地当前路径下，请注意当前路径是否有写文件权限。
- b：视频的码率。(可变，最小 256*1024，最大 8*1024*1024)。
- file：输入的文件路径名称。(可变，根据实际情况设置)。

Sample 注意事项如下：

- 音频库的依赖。

  由于转码功能依赖于 libHA.AUDIO.AAC.encode.so，但涉及版权问题，需用户自行提供；又由于视频播放不同音频格式依赖于不同音频库，比如：

  - libHA.AUDIO.MP3.decode.so
  - libHA.AUDIO.WMA.decode.so
  - libHA.AUDIO.AAC.decode.so 等

  这些音频解码库，需用户自行提供。

  获得这些库后，对于 Android 版本，请用户通过 Android 的 adb 工具推送这些音频库到单板/system/lib 目录下；对于 Linux 版本，请用户通过 tftp 工具下载这些到单板/usr/lib 目录下。

- 对于 sample_transcoder_pipplayer，由于启动两路 avplay，可能会存在 mmz buffer 不足问题，请参照平台 SDK 文档说明适当增大此 buffer 大小，以使 PipPlayer 正常运行。
- 转码 sample 中 dvbplayer，pipplayer 两个 sample，与海思提供的 HiDTVPlayer 业务功能无法一起运行，也就是 DVB 功能无法在两个进程内同时使用；两业务同时启动时，第二个业务会出现无法启动现象；但允许两个业务在一个进程内使用。特此声明，请用户注意。
- 转码 sample_transcoder_osdplayer 所演示的 osd+video 转屏功能，与海思的 HiMultiScreen 业务有资源冲突现象，两个功能无法同时启用；两业务同时启动时，第二个会出现无法启动现象。特此声明，请用户注意。
- 烧写单板后，在 android 版本上有默认图形层显示，而 Linux 版本默认是黑屏状态；因此在 Linux 版本上 osdplayer 执行后，客户端连接会默认显示黑屏；用户可使用 Linux 版本 Sdk_Dir/sample 目录下 tsplay，dvbplay 等 sample 另起一个进程播放视频或显示图形，如此在客户端连接后就有可显示的内容。

Proc 使用指南

1. HiTranscoder

HiTranscoder 在/proc/hisi 目录下创建 Hitranscoder 目录以供本模块设备文件使用。

Hitranscoder 模块包括两种 entry，也就是两种设备文件：Hitranscoder、mbuf。这两种设备文件之间存在一一对应关系，即存在一个 Hitransoder 设备文件，必存在对应的另外一个 mbuf 设备文件。由于 HiTranscoder 模块支持多个 HiTranscoder 实例共存，也就是允许多个不同转码数据流通路同时运行；因此可同时存在多对 transcoder 和 mbuf 设备文件。

由于多实例的存在，且考虑到多进程同时调用，因此对 entry 命名如下：

hitranscoder0X_PID，mbuf0X_PID

其中 PID 是当前进程 pid。

在 HiTranscoder 模块通过 HiTranscoder 接口 create、start 运行起来后，可进行如下设备 Proc 打印。

- Hitranscoder 设备
  - cat 打印

在串口输入命令：

```
cat /proc/hisi/hitranscoder/hitranscoder0X_PID
```

会在串口打印 Hitranscoder 相关编码参数，当前状态，读写信息等。

- echo 命令

```
echo print=0/1 >/proc/hisi/hitranscoder/hitranscoder0X_PID
```

其中 print=1 会开启 mbuf write 打印，打印写入 mbuf 的音视频时间戳，负载类型信息。print=0 会关闭该打印。默认该打印不开启。

```
echo v=0/1 vidEsFileName audEsFileName
>/proc/hisi/hitranscoder/hitranscoder0X_PID
```

其中 v=1 可启动 ES 文件写入，v=0 可停止文件写入；vidEsFileName 为视频 ES 流写入路径，比如/data/app/vid.h264；audEsFileName 为音频 ES 流写入路径，比如/data/app/aud.aac。其中 vidEsFileName 和 audEsFileName 是必备参数。

- mbuf 设备

cat 打印：

在串口输入以下命令：

```
cat /proc/hisi/hitranscoder/mbuf0X_PID
```

会在串口打印 mbuf 相关信息，包括音视频同步 buf 大小，位置，当前读 handle 个数，以及读 handle 相关读指针等信息。

2. HiMuxer

HiMuxer 在/proc/hisi 目录下创建 himuxer 目录以供本模块设备文件使用。本模块包括一个 proc 设备文件，命名为 himuxer。

- cat 打印

```
cat /proc/hisi/himuxer/himuxer
```

会在串口打印当前创建的 muxer 实例个数，以及每个实例的类型和句柄值（0xXXXXXXX）。

打印举例如下：

```
muxerNum: 1
muxerType:ts_default muxerHanle 0xb7fb80f0
```

- echo 命令

```
echo v=0/1 muxHandle FileName >/proc/hisi/himuxer/himuxer
```

其中 v=1 可启动 Mux 后文件写入，v=0 可停止文件写入；muxHandle 为 cat 中打印出的某个 muxer 实例的 handle 值，FileName 为 mux 后数据文件写入路径，比如/data/app/local.ts；

3. HiProtocol

- 当前 HiProtocol 支持 Rtsp，Http 两种协议。HiProtocol 在/proc/hisi 目录下创建 HiProtocol 目录以供本模块设备文件使用。由于 Http 和 Rtsp 都支持多个客户端同时接入，因此分别对每个接入 Session 创建对应设备。Rtsp 设备

Rtsp 设备命名为 rtspSess0X。

cat 打印

```
cat /proc/hisi/hiprotocol/rtspSess0X
```

会在串口打印当前该客户端 Session 相关信息，包括客户端 IP，连接端口，RTP/RTCP 端口号，以及音视频数据发送相关信息。

- Http 设备

  Http 设备命名为 httpSess0X。

  cat 打印

  ```
  cat /proc/hisi/hiprotocol/httpSess0X
  ```

  会在串口打印当前该客户端 Session 相关信息，包括客户端 IP，连接端口，端口号，以及封装后数据发送相关信息。

7.4.2.2 配置客户端运行环境

转码客户端是海思自主开发的低延时客户端，并以开放源码的方式发布，转码客户端支持 rtsp 协议。

- 在 Android 平台版本中，客户端源码位于平台 SDK 根目录下的 device/hisilicon/bigfish/hidolphin/component/hitranscoder/handset_software 下。
- 在 Linux 平台版本中，客户端源码位于平台 SDK 根目录下的 source/component/hitranscoder/handset_software 下

其中 handset_software 下包含两个子目录：

- libHiTransPlayer_jni：客户端 C 层源码，用以生成 player 依赖的动态库。
- HiTransPlayer：客户端 java 层源码，用以生成 Android APK。

libHiTransPlayer_jni 编译说明

libHiTransPlayer_jni 目录的编译工作需要使用 android ndk 工具进行编译。在运行应用程序前，请按如下步骤设置运行环境：

步骤 1 在 linux 下命令行模式下，执行以下命令以新建路径：

```
mkdir -p ~/workspace/NDK
```

步骤 2 下载 android-ndk。

用户可以自行到 google android website 的如下路径下载：

```
http://dl.google.com/android/ndk/android-ndk-r7b-linux-x86.tar.bz2
```

该工具为客户端 C 层代码编译工具，下载后的文件放在 Linux 系统环境下。

步骤 3 解压 NDK 编译器。

在 Linux 下命令行模式执行：

```
tar xjf android-ndk-r7b-linux-x86.tar.bz2 -C ~/workspace/NDK
```

解压 ndk-r7b 到~\workspace\NDK 目录下。

步骤 4 配置 NDK 编译器：

在 Linux 下命令行模式执行：

```
vi ~/.bashrc
```

在文件的最后添加两句：

```
export NDK_HOME=~/workspace/NDK/android-ndk-r7b/
export PATH=$PATH:$NDK_HOME
vi下“:wq”
```

保存退出，执行：

```
source ~/.bashrc
```

让设置马上生效。

步骤 5 拷贝转码客户端 libHiTransPlayer_jni 和 envsetup_player.sh 到 ~/ workspace 下。

步骤 6 在 Linux 下命令行模式在目录 workspace 下执行

```
./envsetup_player.sh
```

步骤 7 libHiTransPlayer_jni 目录下执行 `./ndk-build`。

步骤 8 libtest1_jni.so 和 libhi_ffmpeg.so 生成在：目录 ~/workspace/libHiTransPlayer_jni/libs/armeabi-v7a 下。

----结束

HiTransPlayer 编译说明

HiTransPlayer 编译步骤如下：

步骤 1 配置 eclipse and Android SDK windows 环境。

开发环境是基于 Eclipse 4.2.0 和 ADT 21.0.0，请用户自行下载 Eclipse 4.2.0，ADT 21.0.0 和 google Android SDK。

1. 解压 eclipse ， ADT 和 Android SDK；
2. 执行 eclipse;
3. 在 eclipse 的菜单栏 windows-->Preference-->Android-->SDK Location 设置路径为刚解压的 Android SDK。
4. 在 eclipse 的菜单栏 help-->Install New Software-->Add-->在 name 项输入 Android Plugin-->Arichive-->添加 ADT 包-->OK。

步骤 2 拷贝一份 HiTransPlayer 到用户 Windows 环境下。

步骤 3 拷贝从 libHiTransPlayer_jni 下编译出来的 libs 目录到 HiTransPlayer 目录下，并确认 libtest1_jni.so、libhi_ffmpeg.so、libandroid4.4_jni.so 在 HiTransPlayer\libs\armeabi 下。

步骤 4 导入工程，生成 APK。

1. 打开 eclipse，菜单栏中点击 File-->import。
2. 选择 Gerneral-->Existing Project into workspace。
3. 点击 Browser，找到拷贝到 windows 中的 HiTransPlayer 目录。
4. 点击 Finish ，编译生成客户端 EATAPlayer.apk。

步骤 5 在 Android 手机/Pad 上安装此客户端。

此客户端要求 Android 版本最低 Android2.3。

----结束

注意

EATAPlayer 客户端由于兼容 2.3 以上 android 版本，因此需求 JAVA JDK 版本为 1.6，因此请在使用 eclipse 前确认，通过在 eclipse 菜单栏 window-->preference-->Java-->compiler 中的 JDK 版本请选择 1.6，并保证运行 eclipse 的机器已安装 JDK1.6 版本。

客户端使用说明

请按如下步骤设置运行环境：

步骤 1 确认单板上已经上传了 HiTranscoder 相关的 sample 二进制运行程序，并处于运行。

步骤 2 使用音视频线连接电视机与开发板，打开电视机，选择正确的音视频输入模式。

步骤 3 确认已经为单板分配了 IP 地址。

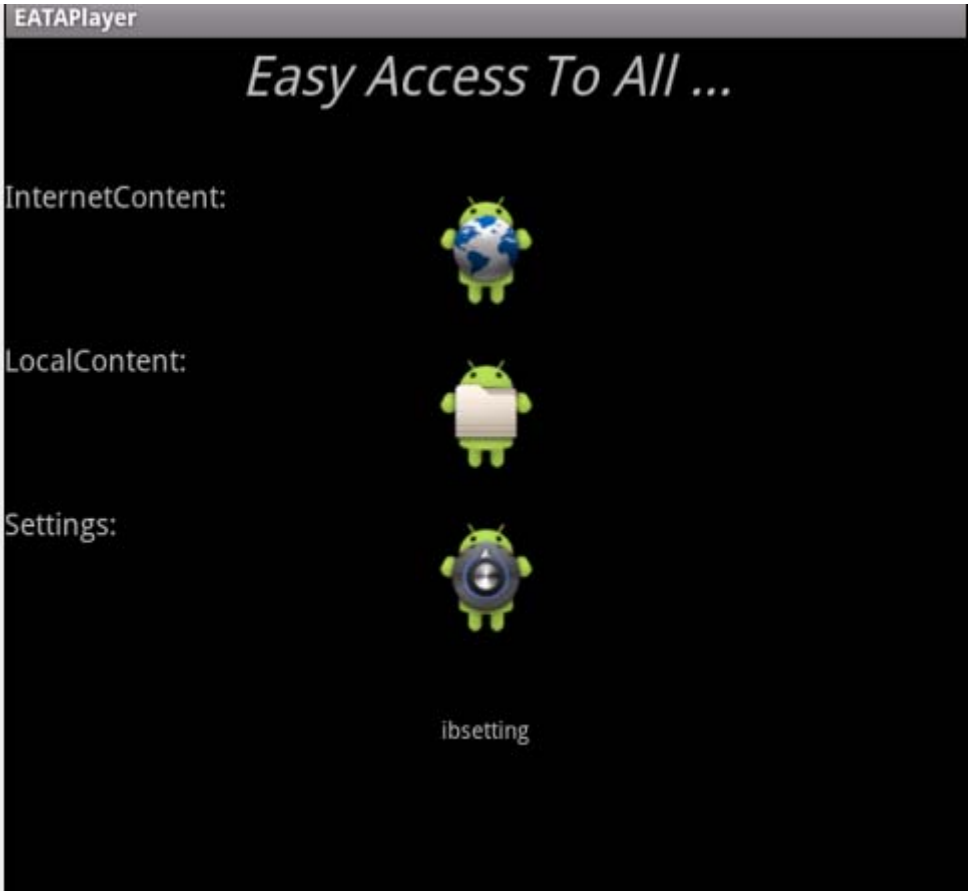
步骤 4 将手持设备通过 WIFI 接入网络，保证手持设备和单板是网络互通的。确认手持设备装有支持 RTSP/HTTP 流媒体协议的播放软件。

电脑上通过有线接入局域网，保证电脑和单板是网络互通的。确认电脑上装有能够支持 RTSP/HTTP 流媒体协议的播放软件。

步骤 5 以低延时客户端做例子，描述如何使用播放软件播放转码数据。

- IP 为单板 IP 地址。
- 确认手持设备已经安装 EATAPlayer.apk。
- port 为用户设置的协议端口，由用户调用设置，在发布的服务端 sample 代码中，此端口设置为 4098.需要注意不要占用已经使用的端口。

图7-11 HiTranscoder 模块低延时客户端使用截图



进入 LocalContent 输入 URL: rtsp://Ip:Port/hisi，接入单板上的服务器。低延时客户端仅支持 rtsp 协议连接。

步骤 6 HiTranscoder 版本同时支持一些其它的协议和封装格式。

其中：

- RTSP 协议：
  点播 rtsp://IP:port/, 不支持暂停操作。
- HTTP 协议：
  - Ts 封装格式：能够点播 http://IP:port/XX.ts。
  - Flv 封装格式：可以点播 http://IP:port/XX.flv。
    (其中 XX 为 2－3 个任意字符，没有特殊限制)
    用户可自行选择其他播放器进行选择播放。

----结束

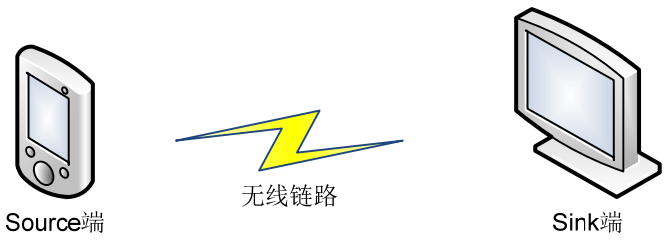
8 HiMiracast

8.1 概述

8.1.1 Miracast 组网图

如图 8-1 所示，Source 端为 Miracast 音视频数据源端，负责音视频数据的编码及发送。而 Sink 端为 Miracast 业务的接收端，负责接收源端发送到 Sink 端的音视频码流，并解码送给显示器显示，当前 Miracast 是为 sink 端设备实现。

图8-1 Miracast 组网图



8.1.2 重要概念

【Wi-Fi Direct】

2010 年 10 月，Wi-Fi Alliance（Wi-Fi 联盟）发布 Wi-Fi Direct 白皮书，白皮书中介绍了有关于这种技术的基本信息、这种技术的特点和这种技术的功能，Wi-Fi Direct 标准是指允许无线网络中的设备无需通过无线路由器即可相互连接。与蓝牙技术类似，这种标准允许无线设备以点对点形式互连，而且在传输速度与传输距离方面则比蓝牙有大幅提升。

【Wi-Fi Display】

Wi-Fi Display 是一种基于 Wi-Fi Direct 标准的技术，可在无线设备与高清显示屏之间实现可靠、对等（P2P）的高清视频与音频流传输。利用这种技术，消费者可以从一个移动设备将音视频内容实时镜像到大型屏幕，随时、随地、在各种设备之间可靠地传输和观看内容。

【Miracast】

Miracast 是 Wi-Fi Alliance（Wi-Fi 联盟）于 2012 年 9 月 19 日宣布启动的 Wi-Fi CERTIFIED Miracast™认证项目，是 Wi-Fi 联盟创立的标准无线显示技术标准。Miracast 基于 Wi-Fi Direct 协议，可直接从一个设备发送内容到另一个设备，而并不通过路由器。

【HDCP】

HDCP（High-bandwidth Digital Content Protection）用于保护音视频数据在 HDCP 发送设备和 HDCP 接收设备之间传输时的安全性。从应用的领域划分，HDCP 技术可分为两种类型：

- 与接口相关的 HDCP 技术，用于保护 HDMI 传输时的音视频数据，当前版本为 HDCP 1.4；
- 与接口无关的 HDCP 技术，用于保护无线传输时的音视频数据，当前版本为 HDCP 2.2。

在 HDCP 的生态系统中，共有三种类型的设备：

- 发送设备（Transmiter）

  发送设备发送音视频数据到转发设备或接收设备；
- 转发设备（Repeater）

  转发设备从发送设备接收数据并转发给下游的接收设备或者显示设备（例如电视机），也可看做一种特殊的接收设备；
- 接收设备（Receiver）。

  接收设备接收数据并显示。

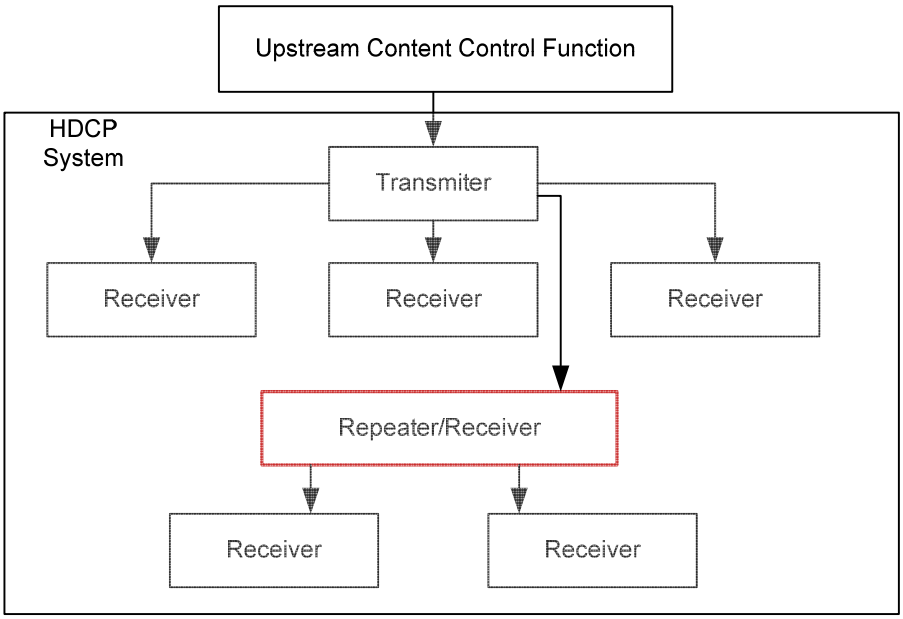
如表 8-1 所示。

表8-1 HDCP 规格

HDCP 规格	规格描述	场景
Transmiter	发送端，加密发送的数据	一般支持 HDCP 的手机作为发送端，连接成功后把音视频数据加密后，通过 TCP/UDP/RST 等协议发送出去
Repeater	接受端，接受数据并转发	一般接收的盒子作为转发中介，把音视频数据解密后再重新加密，发送到其他盒子或者接收端
Receiver	接受端，解密数据	一般显示设备作为接收端，接收发送端手机的被加密的音视频数据，解密后显示出来。

STB 设备属于转发设备，也可以看作是特殊的接受设备。HDCP2.2 包括 3 个基本功能，分别是 HDCP Transmiter，Repeater 和 Receiver。当前海思 Android 平台支持 HDCP2.2 receiver。

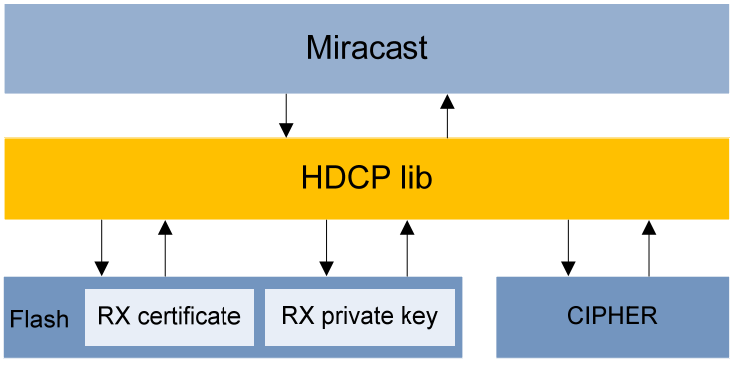
图8-2 HDCP 系统拓扑连接示意图



8.1.3 基本原理

Miracast 将接收到的加密数据传送给 HDCP library，HDCP library 经过解密后，将明文数据返回给 Miracast 处理，如图 8-3 所示。

图8-3 总体框图



整个 HDCP 库工作流程，遵从 HDCP2.2 的规范。

说明

HDCP2.2 的规范可以从 DCP LLC 组织（HDCP 官方组织）的网站 http://www.digital-cp.com/免费下载。

8.2 环境配置

8.2.1 Miracst HDCP 模式配置

目前 Miracast 已支持 HDCP，共有 3 种模式：

- SW HDCP 模式
- HW HDCP 模式
- 非 HDCP 模式

目前全编译默认是非 HDCP 模式，该模式无法解析支持 HDCP 的音视频流。

HW HDCP 模式为海思和第三方公司 Discretix 共同开发的支持 HDCP 的方案，需要依赖 Discretix 公司的库，不是我们主流的方案，一般情况下不要使用。

SW HDCP 模式是我们主流的方案，用户如果要修改为 SW HDCP 模式，需要烧写 HDCP Key 才能使用，SW HDCP 模式下生产开发请参考章节“8.5 SW HDCP 生产开发指导”，如果不支持 SW HDCP 模式，请忽略章节“8.5 SW HDCP 生产开发指导”。

用户可以根据需要自行选择模式编译。具体配置方法如下：

步骤 1 进入源码如下路径，打开 hdcp.xml 文件。
/device/hisilicon/bigfish/hidolphin/component/miracast/android/packages/apps/Miracast/res/raw

步骤 2 根据需要设置模式。

- 如果如果要选择非 HDCP 模式，修改文件如下：

```
<miracast>
    <name>none hdcp</name>
    <mode>1</mode>
</miracast>
```

- 如果要选择 HW HDCP 模式，修改文件如下：

```
<miracast>
    <name>HW hdcp</name>
    <mode>2</mode>
</miracast>
```

- 如果要选择 SW HDCP 模式，修改文件如下：

```
<miracast>
    <name>SW hdcp</name>
    <mode>3</mode>
</miracast>
```

步骤 3 编译烧写。

按照发布包中文档《Android 解决方案 开发指南》中的“2.3 开发编译”进行全编译，编译成功后按要求烧写单板。

----结束

8.3 开发指引

8.3.1 Proc 使用指导

Proc 可以协助开发人员快速的查看 Miracast 当前的状态信息，了解当前运行的状态，并提供了 echo 命令来设置当前设备的参数。

8.3.1.1 cat 查看当前状态

步骤 1 Source 端进行连接请求，开启传屏。

步骤 2 打开串口，进入到 HiMiracast 的 Proc 目录。

命令如下：

```
root@android:/ # cd proc/hisi/himiracast/
```

步骤 3 查看 entry 的数目，如图 8-4 所示。

图8-4 查看 Proc 的 entry

```
root@android:/proc/hisi/himiracast # ls
sink_info
sink_state
```

步骤 4 查看 sink_info entry 内容，如图 8-5 所示。

图8-5 sink_info entry 内容

```
root@android:/proc/hisi/himiracast # cat sink_info
==============================Sink Info=========================
Source IpAddress                 : 192.168.49.242
RTSP Port                        : 7236
Sink Rtp Port                    : 19000
Source Rtp Port                  : none
----------------------------wfd capability--------------------
0:none 1:supported
Wfd_content_protection           : 0
Wfd_coupled_sink                 : 0
Wfd_standby_resume_capability    : 0
==============================Sink Info=========================
```

图 8-5 中的信息参数解释如表 8-2 所示。

表8-2 Sink_info 内容解释

参数	解释
Source IpAddress	Source 端的 IP 地址。
RTSP Port	Source 端建立 RTSP 连接的端口号。

参数	解释
Sink Rtp Port	Sink 端 RTP 的端口号。
Source Rtp Port	Source 端 RTP 的端口号。 None：未获取到端口号。
wfd_content_protection	是否支持 HDCP。 0：不支持； 1：支持。
wfd_coupled_sink	是否支持次级 Sink。 0：不支持； 1：支持。
wfd_standby_resume_capability	是否支持 standby 模式。 0：不支持； 1：支持。

步骤 5　查看 sink_state entry 内容，如图 8-6 所示。

图8-6 sink_state entry 内容。

```
root@android:/proc/hisi/himiracast # cat sink_state
==============================Sink State==============================
0:UNDEFINED 1:CONNECTING 2:CONNECTED 3:PAUSED 4:PLAYING
Sink state                          : 4
----------------------------------------------------------------------
stat lost pkt time                  : 1
lost packets num                    : 0
----------------------------------------------------------------------
0:normal 1:simple
vdec output mode                    : 1
==============================Sink State==============================
```

图中的信息参数解释如表 8-3 所示。

表8-3 sink_state 内容解释

参数	解释
Sink state	Sink 当前所处的状态。 0：未初始化； 1：正在连接； 2：已连接； 3：暂停； 4：正在传屏。
Stat lost pkt time	统计丢包数目的时间间隔，单位是秒。
Lost packets num	在 Stat lost pkt time 间隔内丢包的数目。
Vdec output mode	Vdec 当前所处的模式。 0：快速输出模式； 1：“卡屏”模式。

----结束

8.3.1.2 echo 设置参数

- 传屏状态下，设置 sink 端主动保活（keep alive）时间。
  预制状态：正在传屏
  在串口输入 echo alivetime=X > /proc/hisi/himiracast/sink_info，如图 8-7 所示。

  图8-7 设置 Sink 保活时间

```
root@android:/ # echo alivetime=6 >/proc/hisi/himiracast/sink_info
Arg[0] = alivetime=6
CMD_SINK_KEEPALIVE_TIMEOUT
```

  sink 端主动保活时间默认为 5 秒。设置范围是 5～30 秒，超过这个范围的命令会提示超出范围，重新输入。如图 8-8 所示，表示当前设置的参数超出的范围，需要重新输入。

图8-8 设置保活时间超出范围

```
root@android:/ # echo alivetime=1 >/proc/hisi/himiracast/sink_info
Arg[0] = alivetime=1
too small, please input number between 5 - 30
root@android:/ #
root@android:/ # echo alivetime=31 >/proc/hisi/himiracast/sink_info
Arg[0] = alivetime=31
too big, please input number between 5 - 30
root@android:/ #
```

- 传屏状态下，设置 sink 端统计 RTP 丢包的时间间隔。

  预制状态：正在传屏

  在串口输入 echo lostpkttime=X > /proc/hisi/himiracast/sink_info，如图 8-9 所示。

图8-9 设置 sink 端统计 RTP 丢包的时间间隔

```
root@android:/ # echo lostpkttime=3 >/proc/hisi/himiracast/sink_info
Arg[0] = lostpkttime=3
CMD_SINK_STAT_LOSTPKT_TIMEOUT
```

sink 端统计丢包数目的时间间隔默认为 1 秒。设置范围是 1～10 秒，超过这个范围的命令会提示超出范围，重新输入。如图 8-10 所示，表示当前设置的参数超出的范围，需要重新输入。

图8-10 设置 sink 端统计 RTP 丢包的时间间隔超出范围

```
root@android:/ # echo lostpkttime=0 >/proc/hisi/himiracast/sink_info
Arg[0] = lostpkttime=0
too small, please input number between 1 - 10
root@android:/ #
root@android:/ # echo lostpkttime=11 >/proc/hisi/himiracast/sink_info
Arg[0] = lostpkttime=11
too big, please input number between 1 - 10
```

- 传屏状态下，设置 sink 端 vdec 的输出模式。

  预制状态：正在传屏

  在串口输入 echo mode=X > /proc/hisi/himiracast/sink_info，mode=0 表示设置为固定的 normal 模式，即“卡屏”模式；mode=1 表示设置为固定的 simple 模式；mode=2 表示设置为根据丢包率自动调节 Vdec 信息的模式。如图 8-11 所示。

图8-11 设置 sink 端 vdec 的输出模式

```
root@android:/ # echo mode=2 >/proc/hisi/himiracast/sink_info
Arg[0] = mode=2
CMD_SINK_VDEC_MODE_MIX_NORMAL_SIMPLE
```

8.3.2 修改设备名字

目前启动 Miracast 应用后，UI 中显示的设备名字是 Android_xxxx(Android + 4 位随机数字)，通过 source 端搜索到的设备名字也是 UI 上显示的名字。如果需要修改设备名字，操作步骤如下：

步骤 1 打开源码如下目录的文件 WfdService.java。

/device/hisilicon/bigfish/hidolphin/component/miracast/android/packages/apps/Miracast/src/com/hisilicon/miracast/service

步骤 2 调用接口 mWfdBusiness.setDeviceName(newDeviceName)。

搜索关键字符串“WIFI_P2P_STATE_ENABLED”，在该分支中调用。代码如下：

```
if (isEnabled)
{
LogUtil.i("WIFI_P2P_STATE_ENABLED");
hideWaitingDialog();
mWfdBusiness.setDeviceName(newDeviceName);
}
else
{
LogUtil.i("WIFI_P2P_STATE_DISABLED");
}
```

步骤 3 重新编译 Miracast.apk。

在/device/hisilicon/bigfish/hidolphin/component/miracast/android/packages/apps/Miracast 下执行 mm –B，会生成 out/target/product/Hi3716CV200/system/app/Miracast.apk。

步骤 4 push Miracast.apk 到单板。

下面假设 STB IP 为 10.161.179.2，Miracast.apk 在 PC 上的路径为 D:\ Miracast.apk

```
adb connect 10.161.179.2
adb remount
adb push D:\Miracast.apk system/app
```

----结束

8.4 调试指引

8.4.1 概述

Miracast 连接过程涉及了多个模块的功能业务，包括 Wi-Fi 驱动，Wpa_supplicant，Framework，Wi-Fi Display 协议栈，应用 APP 等。某个环节出错都有可能导致连接出现异常，而定位连接异常问题相当费时耗力。本节内容的主要意义是指导客户可以更快高效、准确的定位 Miracast 业务出现的连接问题。

出现连接问题的原因通过归纳可以分为如下几类：

- Wi-Fi 驱动问题

  由于部分客户不使用海思提供的 Wi-Fi 驱动程序，而使用自己的驱动，如果客户本身对 Miracast 业务测试较少，可能会存在由于 Wi-Fi 驱动适配不完善导致的连接问题。

- Wpa_supplicant 问题

  Miracast 业务的连接对 Wi-Fi Direct 有强依赖关系，Wpa_supplicant 是 Wi-Fi 模块流程控制的中心，本身比较复杂，如果适配不当会导致连接异常。

- Wi-Fi Display 协议问题

  从 Miracast 业务诞生到现在，支持 Miracast 业务的手机或平板设备已经从最初的一两款到现在的几十款甚至上百款。不同的厂家对 Wi-Fi Display 代码实现会存在一定的差异，导致出了一些兼容性问题。

- 手机本身的问题

  有一小部分手机本身存在问题导致连接失败。

- 网络环境问题

  在无线网络环境比较复杂，干扰比较大的情况下，有可能会导致 Wi-Fi Direct 建立的过程中 GO（Group Owner）协商超时，导致连接失败。

- WPS 鉴权失败问题

  在 Miracast 连接过程中，网络环境中存在着另外一台 Source 设备（手机或平板）去连接 Miracast 或做类似 WPS 业务，如手机 A 在连接盒子 A 的同时手机 B 去连接另外一台盒子 B，会导致 WPS 鉴权失败，导致连接失败。

- 用户不正确操作

  由于对 Miracast 业务不熟悉，把 Wi-Fi 直连和 Wi-Fi Display 搞混淆，从 Wi-Fi 直连界面发起连接。

8.4.2 定位准备

确定环境

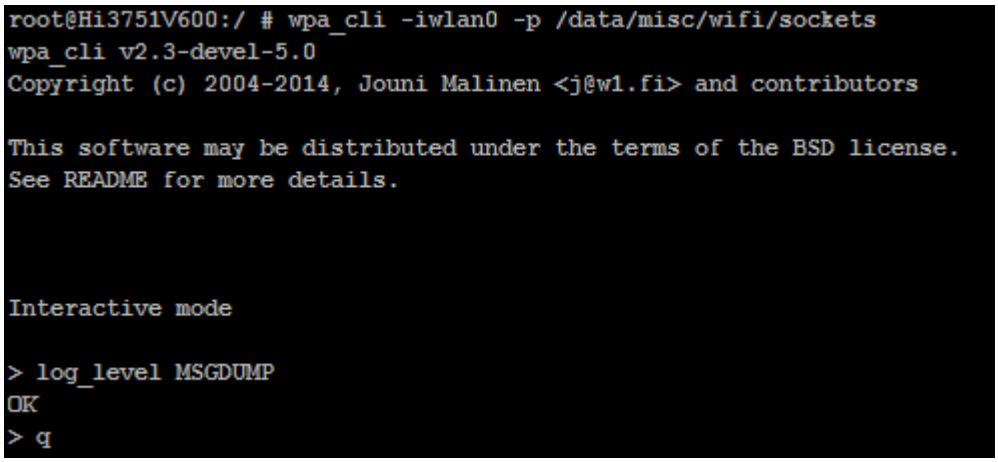
Miracast 业务基于 Wi-Fi Direct 协议，需要在客户的设备上有可以支持 Wi-Fi Direct 的 Wi-Fi 设备，检测客户使用的 Wi-Fi 是否在海思发布的版本兼容性列表中。

抓取日志方法

正确的抓取日志，需要注意如下几点：

- 要在连接前就开启 logcat。
- 抓取日志前需先 logcat –c 清空日志。
- 抓取时要同时记录时间，使用 logcat –v time 抓取。
- 如果出现大量的其他日志刷屏，要把刷屏的日志屏蔽，如：logcat -s XXX:F *:v，XXX 表示大量重复的日志标志。
- 抓取 Wi-Fi Direct 流程日志需要打开 Wpa_supplicant 中的日志开关。打开方法如图 8-12 所示。

图8-12 打开 Wpa_supplicant 日志开关方法



抓取空口包方法

由于 Miracast 是一个 Source 和 Sink 交互的过程，很多时候不确定是哪一端的问题，所以必须抓 sniffer 包来分析，抓取 sniffer 包需要准备：

- 硬件：抓包网卡（推荐 RT5572 芯片的抓包网卡）
- 软件：网卡驱动及抓包软件 omnipeek（推荐 omnipeek 7.5 版本）。

8.4.3 定位过程

Miracast 连接过程主要包括两部分：

- Wi-Fi Direct 连接
- Wi-Fi Display 连接

这两部分有先后次序，在 Wi-Fi Direct 连接建立成功后，才能够进入到 Wi-Fi Display 连接过程。

抓取到日志后，可以根据日志来定位出现连接问题的原因。下面给出一些经典的场景示例。

Wi-Fi Direct 建立流程及中间过程关键日志

Wi-Fi Direct 建立过程要进行二十几条消息的交互，任何一条消息出错都会导致连接失败。从连接建立过程来看，可以将 Wi-Fi Direct 分成 3 个部分来分析：

- GO（Group Owner）协商过程：

  开始消息：

  ```
  p2p0: P2P: Received GO Negotiation Request from 02:11:22:18:9e:74(freq=2412)
  ```

  其中 02:11:22:18:9e:74 为 source 端 mac 地址

  成功消息：

  ```
  p2p0: P2P: GO Negotiation with 02:11:22:18:9e:74 completed (local end will be GO)
  ```

```
p2p0: P2P-GO-NEG-SUCCESS
```

以上日志表明 GO 协商成功，“local end will be GO”表示 Sink 端为 GO，若括号中内容是“peer end will be GO”则表示 Source 端为 GO 设备。

- GROUP-FORMATION 过程

  在 GO 协商成功后，由 GO 建立 Group，通过 PBC 鉴权方式进行 GROUP-FORMATION 过程。

  开始消息：

  ```
  p2p0: CTRL-EVENT-EAP-STARTED
  ```

  表明 WPS 过程开始。

  成功消息：

  ```
  p2p0: P2P: Group Formation completed successfully with 02:11:22:18:9e:74
  p2p0: P2P-GROUP-FORMATION-SUCCESS
  ```

- 连接过程

  在 GROUP-FORMATION 成功后，GC（Group Client）会使用协商好的密钥来连接 GO

  成功日志 1：

  ```
  I/com.hisilicon.miracast.service.WfdService$WfdReceiver( 2953): onReceive L1216
  ---- Wi-Fi Direct connected ----
  ```

  关键日志 2：

  ```
  D/WifiDisplaySink( 2953): init L116
  ```

  - 如果在抓取的一次连接过程中，可以出现上述两个关键日志之一，都可以表示 Wi-Fi Direct 过程建立已 OK。
  - 如果未看到上述关键日志，则表示 Wi-Fi Direct 建立过程中失败。
  - 如果 Wi-Fi Direct 连接失败，可以参考以上日志确定是在哪一个部分失败的，从而缩小定位范围。

如何判断 Wi-Fi Display 已完成建立连接

关键日志：

```
01-01 00:04:18.405 D/WifiDisplaySink( 2953): sendM7 L1369 request = 'PLAY
rtsp://192.168.49.1:7236/wfd1.0/streamid=0 RTSP/1.0
01-01 00:04:18.405 D/WifiDisplaySink( 2953): Date: Wed, 01 Jan 2014 00:04:18 +0000
01-01 00:04:18.405 D/WifiDisplaySink( 2953): CSeq: 3
01-01 00:04:18.405 D/WifiDisplaySink( 2953): Session: 00000060
01-01 00:04:18.405 D/WifiDisplaySink( 2953):
01-01 00:04:18.405 D/WifiDisplaySink( 2953): '
01-01 00:04:18.413 V/WifiDisplaySink( 2953): onReceiveClientData L1014 session 1
received 'RTSP/1.0 200 OK
01-01 00:04:18.413 V/WifiDisplaySink( 2953): cseq: 3
01-01 00:04:18.413 V/WifiDisplaySink( 2953): date: Mon, Mar 30 2015 03:45:16 GMT
01-01 00:04:18.413 V/WifiDisplaySink( 2953):
01-01 00:04:18.413 V/WifiDisplaySink( 2953):
```

如上述关键日志所示，M7 消息是 Wi-Fi Display 连接协商过程中最后一个消息，如果出现 sendM7 消息的日志，表示 Wi-Fi Display 连接已正常建立完成。

HDCP 协商失败导致连接失败

部分手机存在 HDCP 兼容性问题，会出现 Wi-Fi Display 连接建立已完成的情况下，手机发过来一个 Teardown 消息，导致最终连接失败。

单板侧收到的 teardown 消息示例如下：

```
01-01 04:27:26.426 V/WifiDisplaySink( 2953): onReceiveClientData L1014 session 1
received 'SET_PARAMETER rtsp://localhost/wfd1.0 RTSP/1.0
01-01 04:27:26.426 V/WifiDisplaySink( 2953): content-length: 30
01-01 04:27:26.426 V/WifiDisplaySink( 2953): content-type: text/parameters
01-01 04:27:26.426 V/WifiDisplaySink( 2953): cseq: 4
01-01 04:27:26.426 V/WifiDisplaySink( 2953):
01-01 04:27:26.426 V/WifiDisplaySink( 2953): wfd_trigger_method: TEARDOW
01-01 04:27:26.426 V/WifiDisplaySink( 2953):
```

手机侧异常导致连接失败

在 Wi-Fi Display 连接建立过程中，手机侧是 RTSP 的 Server 端，在其发起连接邀请后会启动一个 RTSP Server，处于 Listen 状态，等待盒子侧的 TCP 连接。部分手机在连接时会出现 RTSP Server 启动失败的情况，导致盒子侧请求的 TCP 连接失败，导致 Wi-Fi Display 连接失败。在这样场景下，可以看到盒子的日志如下：

```
01-02 09:31:29.607 D/WifiDisplaySink( 2549): init L115
01-02 09:31:29.607 D/WifiDisplaySink( 2549): sinkProcCreate L283
01-02 09:31:29.608 V/WifiDisplaySink( 2549): initHDCPRptOrRcv L374
01-02 09:31:46.628 D/WifiDisplaySink( 2549): initHDCPRptOrRcv Success
initHDCPRptOrRcv L384
01-02 09:31:46.628 D/WifiDisplaySink( 2549): platformCheck L255
01-02 09:31:46.628 D/WifiDisplaySink( 2549): start L232
01-02 09:31:46.631 E/WifiDisplaySink( 2549): onMessageReceived L468 RTSP error
01-02 09:31:46.631 E/WifiDisplaySink( 2549): onMessageReceived L483 An error
occurred in session 1 (-111, 'Connection failed/Connection refused').
```

如果在连接过程中出现上述 socket 问题，可以尝试断开手机侧已连接的所有的无线路由，然后再重启手机，重新发起连接。

用户操作不正确导致连接失败

用户在 Wi-Fi 直连（Wi-Fi Direct）选项里面发起 Miacast 连接，连接失败的异常如下：

```
01-01 04:01:17.502 D/WifiDisplaySink(16737): init L116
01-01 04:01:17.502 V/NetworkSession(16737): ANetworkSession L663
01-01 04:01:17.503 V/TunnelRenderer(16737): TunnelRenderer L61
01-01 04:01:17.503 V/RTPSink (16737): RTPSink L296
01-01 04:01:17.503 D/RTPSink (16737): start L302
01-01 04:01:17.503 V/NetworkSession(16737): ANetworkSession L663
01-01 04:01:17.503 V/NetworkSession(16737): ANetworkSession L663
01-01 04:01:17.503 D/WifiDisplaySink(16737): sinkProcCreate L284
01-01 04:01:17.503 D/(16737): Sink_PROC_CreateMode L212
01-01 04:01:17.504 D/WifiDisplaySink(16737): platformCheck L256
01-01 04:01:17.504 D/HiSinkJNI(16737):
Java_com_hisilicon_miracast_ndk_SinkNative_startSink 71
01-01 04:01:17.504 D/WifiDisplaySink(16737): start L233
```

```
01-01 04:01:17.509 I/NetworkSession(16737): createClientOrServer L990 connecting
socket 57 to 192.168.49.1:7236
01-01 04:01:17.581 E/NetworkSession(16737): threadLoop L1361 writeMore on socket 57
failed w/ error -111 (Connection refused)
01-01 04:01:17.581 E/WifiDisplaySink(16737): onMessageReceived L469 RTSP error
01-01 04:01:17.581 E/WifiDisplaySink(16737): onMessageReceived L484 An error
occurred in session 1 (-111, 'Connection failed/Connection refused').
01-01 04:01:17.581 I/WifiDisplaySink(16737): onMessageReceived L488 Lost control
connection.
01-01 04:01:17.581 V/NetworkSession(16737): ~Session L211 Session 1 gone
```

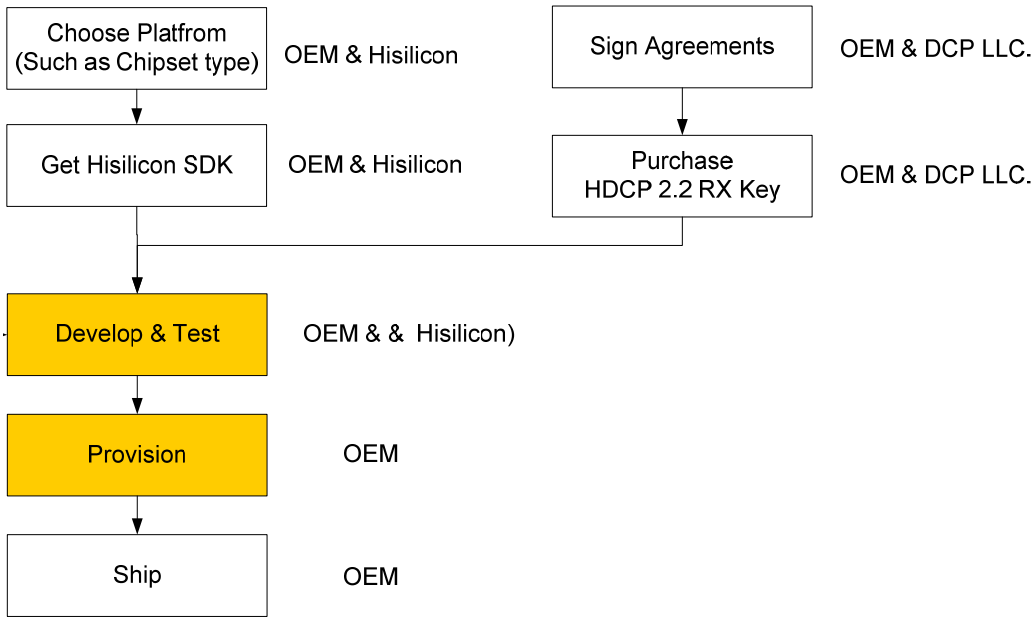
此类问题是 Miracast 使用方式不当造成的，正确的使用 Miracast 业务请参考《Android 解决方案 使用指南》文档中 Miracast 章节。

8.5 SW HDCP 生产开发指导

8.5.1 SW HDCP 开发流程

开发整体流程如图 8-13 所示。

图8-13 SW HDCP 开发流程



如上图所示，过程如下：

步骤 1 OEM 厂商首先向 DCP LLC.组织申请并获取 HDCP2.2 RX Key。

步骤 2 OEM 厂商确定使用的芯片和 SDK 版本，并向海思获取 SDK。

步骤 3 OEM 厂拿到 HDCP2.2 RX Key 和海思支持 HDCP2.2 版本 SDK 后，根据生产环境定制生产工具或者 APP 装备 HDCP Key，并做输出保护方案，完成测试。

步骤 4 OEM 产线生产和出货。

----结束

步骤 3 中，OEM 厂商需要自行定制输出保护方案，在输出保护方案中，一般需要使能 HDMI 的 HDCP 保护特性，此时必须使能 HDCP 1.4。

说明

HDMI 已经支持 HDCP1.4.使能此特性请参考《HDCP KEY 工具使用指南.pdf》和咨询 FAE。

所以，在 SW HDCP 2.2 的开发和测试前，需从 DCP LLC.组织购买 HDCP 2.2 和 HDCP1.4 的密钥。

当前海思 SDK 中已经包含 HDCP2.2 库（libhihdcp.so），但是需要烧写 HDCP2.2 KEY，如何烧写 HDCP2.2 KEY 请参考后面章节。

8.5.2 工厂装备 Key 过程

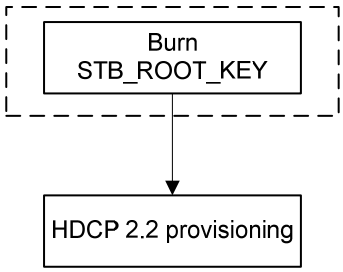
SW HDCP2.方案中， HDCP2.2 Key 是由 STB_ROOT_KEY 加密后再烧写在 flash 中，所以烧写 HDCP Key 之前，OEM 厂商需要烧写 STB_ROOT_KEY，如果 OEM 厂商不另外单独烧写 STB_ROOT_KEY，HDCP 库在烧写前会自动生成一个随机的 STB_ROOT_KEY 并烧写进去，整个 HDCP Key 会烧写在 deviceinfo 分区的尾部位置，占用空间为 4K。如果损坏这个区域请重新操作烧写。

说明

STB_ROOT_KEY 可以由 OEM 厂商生成并管理，烧写在 OTP 后锁住，cpu 就无法从 OTP 中读取到 STB_ROOT_KEY，从而保证安全数据安全。

工厂生产环节简要流程如图 8-14 所示。

图8-14 工厂生产环节简要流程图



根据前面章节的介绍，图 8-14 中虚框部分为 OEM 厂商可选操作，如果 OEM 厂商需要自行管理 STB_ROOT_KEY，则需要完成这部分操作，否则 hdcp 库中会自动随机生成并烧写 STB_ROOT_KEY。烧写 STB_ROOT_KEY 请参考 sample：

- device/hisilicon/bigfish/sdk/sample/otp/sample_otp_setstbrootkey.c
- device/hisilicon/bigfish/sdk/sample/otp/sample_otp_lockstbrootkey.c

OEM 厂商确定完是否需要自行烧 STB_ROOT_KEY 后，请根据如下步骤完成 HDCP Key 烧写，SDK 中已经提供烧写库（libhihdcp_tool.a）和接口 API。

具体步骤使用如下：

步骤 1 使用拆包工具将 HDCP2.2 Key 包拆包，得到单个的明文 key。工具所在目录为：device/hisilicon/bigfish/sdk/tools/windows/HdcpTools/HDCP_Key_Huawei_STB.zip

步骤 2 工厂生产 APK 集成 STB_ROOT_KEY 的烧写。此步骤根据 OEM 厂商需要是否需要完成。如果不需要自行管理 STB_ROOT_KEY，则跳过此步骤。

步骤 3 工厂生产 APK 集成 api 和库，使用 api 烧写。其烧写 API 为：

```
HI_S32  HI_HDCP2_SetHDCPKey(HI_U8 *pKey, HI_U32 u32Length);
```

所在头文件为：

```
device/hisilicon/bigfish/security/hdcp/include/HI_ADP_HDCP_Receiver.h
```

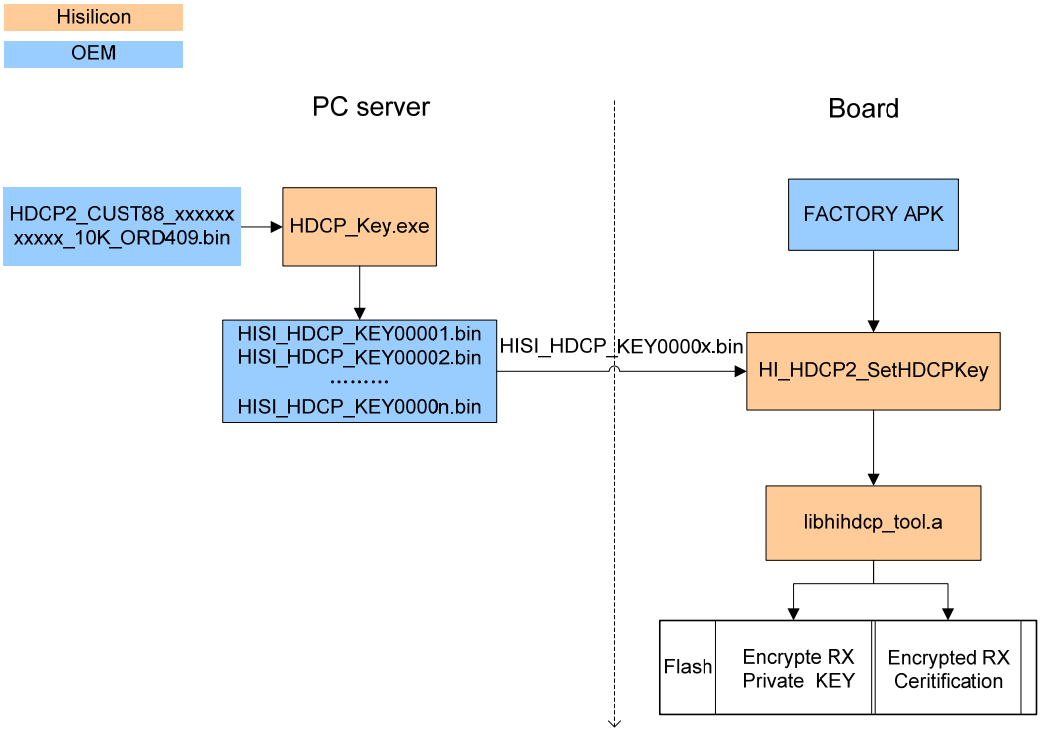
演示 sample 为：

```
device/hisilicon/bigfish/security/hdcp/sample/sample_hdcp_burn.c
手动调试时，可以将编译好的sample_hdcp_burn和拆分好的HDCPkey，
HISI_HDCP_KEYxxxxx.bin下载到/data/目录，然后执行命令：
#./chmod 777 sample_hdcp_burn
#./sample_hdcp_burn HISI_HDCP_KEYxxxxx.bin
```

步骤 4 生产时从服务器端将 Key 下载到单板，生产 APP 将 Key 写入单板。

如果不需要自行烧写 STB_ROOT_KEY，则烧写过程如图 8-15 所示。

图8-15 工厂烧写 HDCP 示意图



烧写 HDCP KEY 后，使用 Mircast 请参考海思发布文档《Android 解决方案 使用指南》中 Miracast 章节。

----结束

9 HiKaraoke

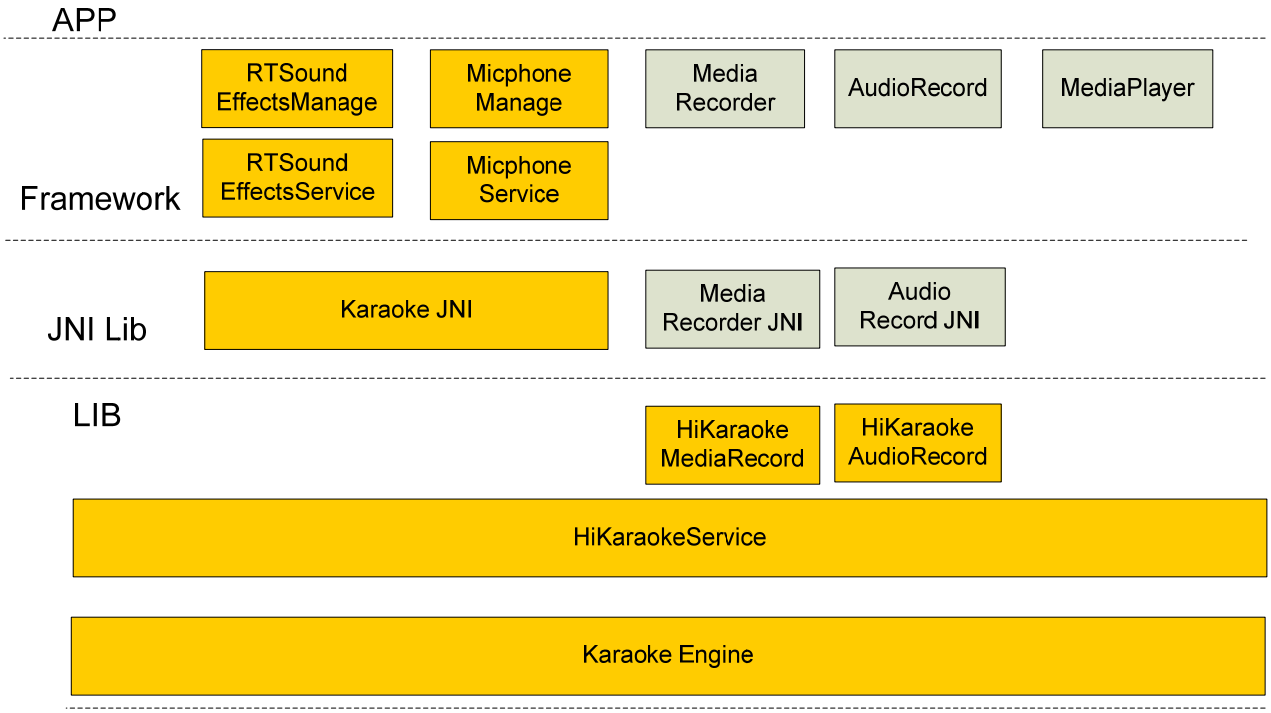
9.1 概述

HiKaraoke 是海思实现的一套卡拉 OK 解决方案，符合中国移动《咪咕家庭音乐卡拉 OK 业务技术规范》。该方案可以实现网络或本地播放伴奏，并将采集到的麦克风数据与伴奏进行混音、音效处理输出，最终实现完美的卡拉 OK 演唱效果。

HiKaraoke 框架如图 9-1 所示。

- 黄色的部分为新增的模块。
- 浅色的模块为对 Android 原声框架修改的模块。
- APP 为应用层，用户可以开发自己的卡拉 OK 应用
- Framework 为框架层，包含 2 个服务和 3 个接口类，主要为卡拉 OK 应用层提供接口
- JNI lib 为过渡层，主要卡拉 OK 的接口提供 Java 到 c++层的过渡
- LIB 层为具体实现层，包括 HiKaraokeService 服务和 Karaoke Engine,引擎部分位于 sdk/source/component/karaoke 目录，以组建库的形式存在，其余的部分都以源码的形式存在

图9-1 HiKaraoke 框架图



9.2 重要概念

【HiKaraoke】

海思卡拉 OK 模块。

【Micphone】

麦克风管理控制服务。

【RTSoundEffects】

音效管理控制服务。

9.3 功能描述

9.3.1 主要功能

HiKaraoke 主要包括实时混音、低延时、音效处理、打分支持、原伴唱切换、混音录制功能。具体如下：

- 实时混音

  实现人声（麦克风输入）与伴奏（MV 或 MP3）实时混音输出。
- 低延时

从麦克风输入到音箱输出的时延小于 30 毫秒。

- 音效处理

  提供录音棚、KTV、演唱会 3 种音效，支持动态切换。

- 打分支持

  支持上层获取实时麦克风数据，支撑上层应用实现打分功能。

- 原伴唱切换

  支持动态切换原伴唱。

- 混音录制

  支持上层获取经过音效处理后的混音数据（AAC 编码），支撑上层应用实现回放、分享等功能。

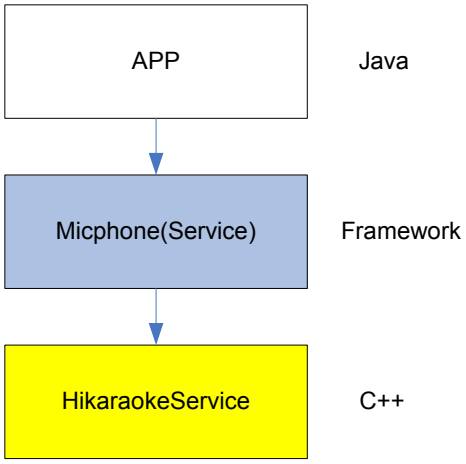
9.3.2 模块原理

9.3.2.1 Micphone 服务

Micphone 服务是 Android AIDL 服务，主要实现麦克风控制的相关接口，包括麦克风打开、关闭、音量控制等。

Micphone 服务的控制流程图图 9-2 所示。

图9-2 Micphone 服务控制流程图

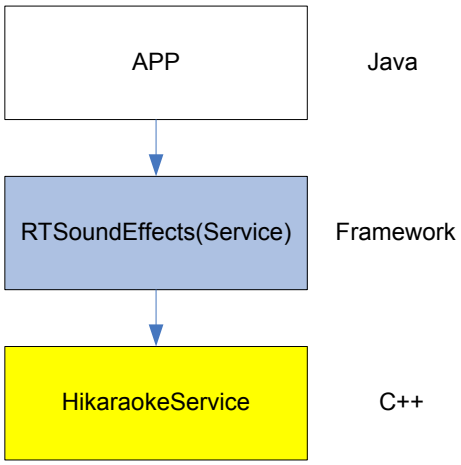


9.3.2.2 RTSoundEffects 服务

RTSoundEffects 服务是 android AIDL 服务，主要实现音效控制的相关接口，包括录音棚、KTV、演唱会三种音效设置。

RTSoundEffects 服务的控制流程图如图 9-3 所示。

图9-3 RTSoundEffects 服务控制流程图



9.3.2.3 HiKaraokeService 服务

HiKaraokeService 服务是 android c++服务，实现上层服务与 Karaoke Engine 的对接。

9.3.2.4 Karaoke Engine

Karaoke Engine 是卡拉 OK 引擎库，实现麦克风控制、实时混音、低延时、音效处理、打分支持、混音录制等功能。

9.4 开发指引

9.4.1 Micphone 服务的使用

Micphone 服务在系统启动时加载，应用可以通过 getSystemService 方法获取服务的接口，对麦克风进行控制。

- 开始播放 MP3 或 MV 时，初始化并打开麦克风

```
Micphone micphone = (Micphone) getSystemService("Micphone");
micphone.initial();
micphone.start();
```

- 演唱过程中，设置麦克风音量

```
micphone. setVolume (volume);
```

- 演唱退出时，关闭麦克风并释放资源

```
micphone. stop ();
micphone. release ();
```

9.4.2 RTSoundEffects 服务的使用

RTSoundEffects 服务在系统启动时加载，应用可以通过 getSystemService 方法获取服务的接口，对音效进行切换。

- 演唱过程中，动态切换音效：

```
RTSoundEffects rtSoundEffects = (RTSoundEffects)
getSystemService("RTSoundEffects");
rtSoundEffects.setReverbMode(RTSoundEffects. REVERB_MODE_STUDIO);
```

9.4.3 原伴唱切换

MediaPlayer 实现了 MediaPlayerEx 类的 switchChannel 接口，可以通过该接口进行原伴唱切换。

- 演唱过程中，切换原伴唱：

```
MediaPlayer mPlayer = new MediaPlayer();
if(mPlayer instanceof MediaPlayerEx){
((MediaPlayerEx) mPlayer). switchChannel (CHANNEL. LEFT);
}
```

9.4.4 麦克风数据的获取

AudioSource 增加了 CMCC_KARAOK_MIC 项，使用该项创建 AudioRecord，并通过 read 接口获取麦克风数据。

- 创建 AudioRecord：

```
int minFramesize = AudioRecord.getMinBufferSize(44100,
AudioFormat.CHANNEL_CONFIGURATION_MONO,
AudioFormat.ENCODING_PCM_16BIT);
AudioRecord mAudioRecord = new
AudioRecord(AudioSource.CMCC_KARAOK_MIC, 44100,
AudioFormat.CHANNEL_CONFIGURATION_MONO,
AudioFormat.ENCODING_PCM_16BIT, minFramesize);
```

- 读取麦克风数据。

  使用 AudioRecord 的 read 方法。

9.4.5 混音的录制

MediaRecorder 实现 MediaRecorderEx 类的 pause、resume 接口用以支持混音录制控制功能。AudioSource 增加了 CMCC_KARAOK_SPEAKER 项，使用该项创建 MediaRecorder，并通过 MediaRecorder 的方法实现录制。

- 创建 MediaRecorder 并启动录制：

```
MediaRecorder mMediaRecorder = new MediaRecorder();
mMediaRecorder.setAudioSource(HikaraokeUtil.KARAOKE_MEDIARECORD_SOURC
E);
mMediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.AAC_ADTS);
mMediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AAC);
mMediaRecorder.setAudioSamplingRate(48000);
mMediaRecorder.setAudioEncodingBitRate(64000);
```

```
mMediaRecorder.setAudioChannels(2);
mMediaRecorder.prepare();
mMediaRecorder.start();
```

- 暂停录制：

```
if(mMediaRecorder instanceof MediaRecorderEx){
((MediaRecorderEx) mMediaRecorder). pause();
}
```

- 恢复录制：

```
if(mMediaRecorder instanceof MediaRecorderEx){
((MediaRecorderEx) mMediaRecorder). resume();
}
```

9.4.6 麦克风热插拔

系统已经添加了麦克风拔插广播：

- 在 USB 麦克风设备插上的时候，系统发送 com.cmcc.intent.action.MICPHONE_PLUG_IN 广播。
- 在 usb 麦克风设备拔出的时候，系统发送 com.cmcc.intent.action.MICPHONE_PLUG_OUT 广播。

应用可以注册这两个广播，监测 USB 麦克风的状态。

9.4.7 遥控器的适配

原伴奏切换、音效选择等功能可以直接集成到应用的菜单上面进行控制，如果需要通过快捷键进行控制，需要平台与应用进行遥控器的适配。

9.4.8 接口测试说明

device/hisilicon/bigfish/ development/app 目录下面，有 HiKaraokeDemo 测试应用可以对各个接口进行测试，在开发过程中也可以根据这个 Demo 进行开发。