android多媒体框架

讲解人：罗彧成

2010年12月

android体系结构

媒体层结构

opencore基本概念

a、目录结构

b、编译

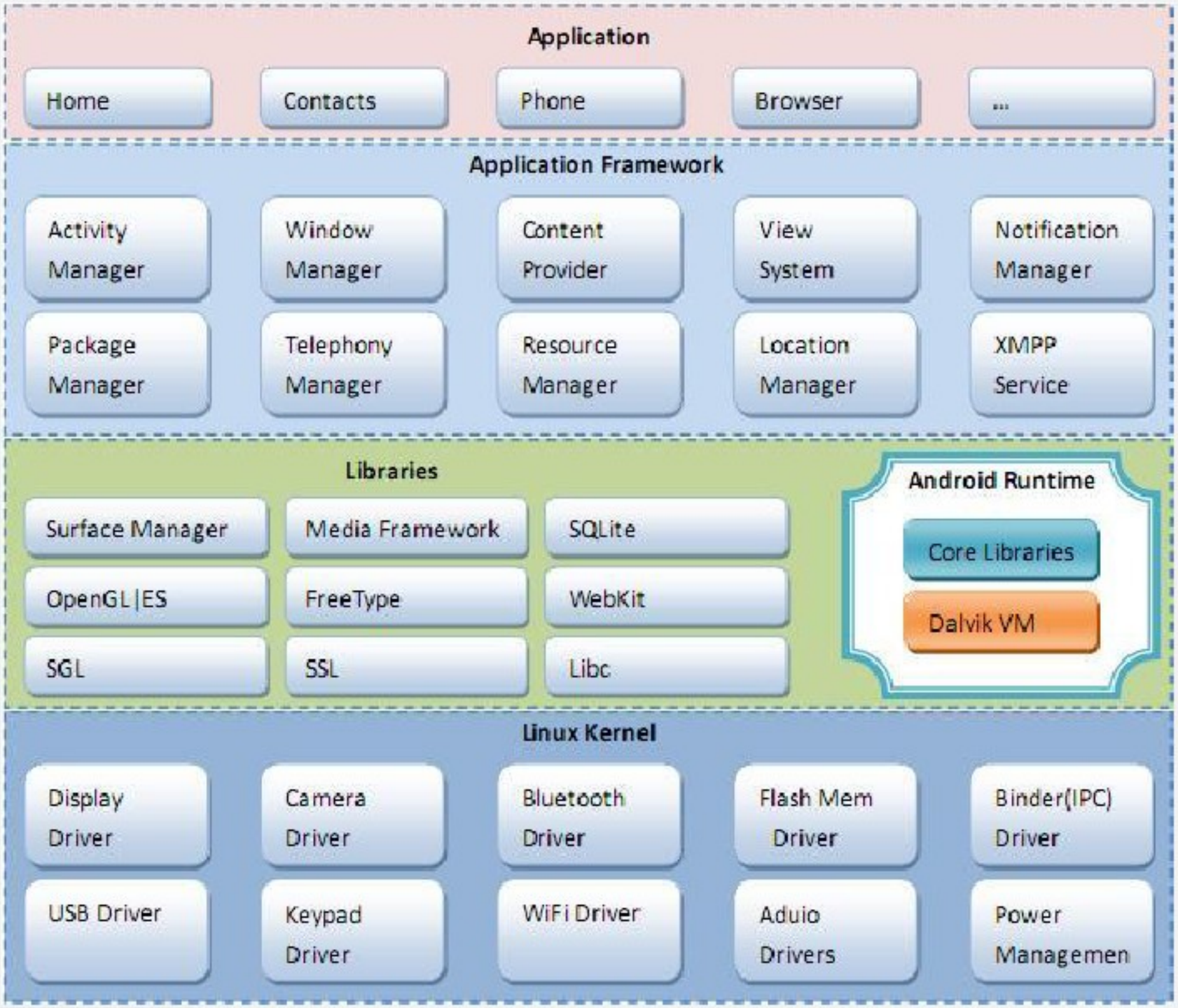
c、调度器(OsclExecScheduler)

d、节点(Node)

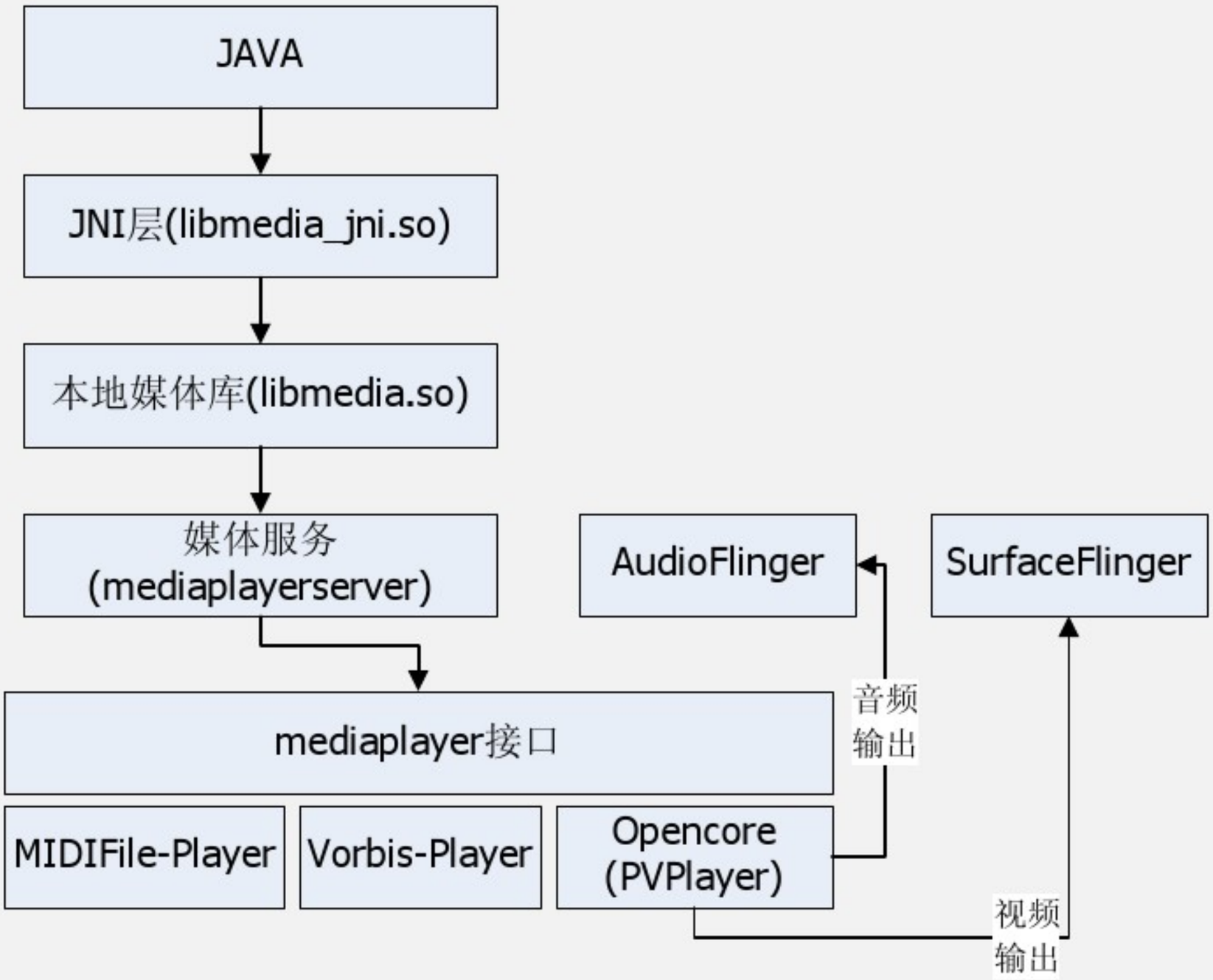
e、端口(Port)

f、消息通讯(Msg)

Android体系结构



媒体层结构


JAVA
JNI层(libmedia_jni.so)
本地媒体库(libmedia.so)
媒体服务
(mediaplayerserver)
AudioFlinger
SurfaceFlinger
mediaplayer接口
音频
输出
MIDIFile-Player
Vorbis-Player
Opencore
(PVPlayer)
视频
输出

媒体层结构

- Android-MediaPlayer上层应用程序使用JAVA语言编写，实现逻辑处理；JAVA程序通过调用底层媒体库来实现具体的音视频文件和网络视频播放；
- JAVA程序通过JNI接口实现对底层媒体库libmedia.so的调用；
- MediaPlayer在运行的时候，可以大致上分成Client和Server两个部分，它们分别在两个进程中运行，它们之间使用Binder机制实现IPC通讯，图中libmediaplayerservice.so是服务端的实现库。
- 对于视频文件的播放，MediaPlayer通过调用Opencore提供的功能媒体播放功能来负责实现；
- 在opencore中负责媒体文件格式解析、音视频数据的解码、以及媒体数据的输出；
- Opencore通过调用SurfaceFlinger的接口实现视频数据的显示；通过调用AudioFlinger的接口来实现音频数据的回放；

媒体层结构

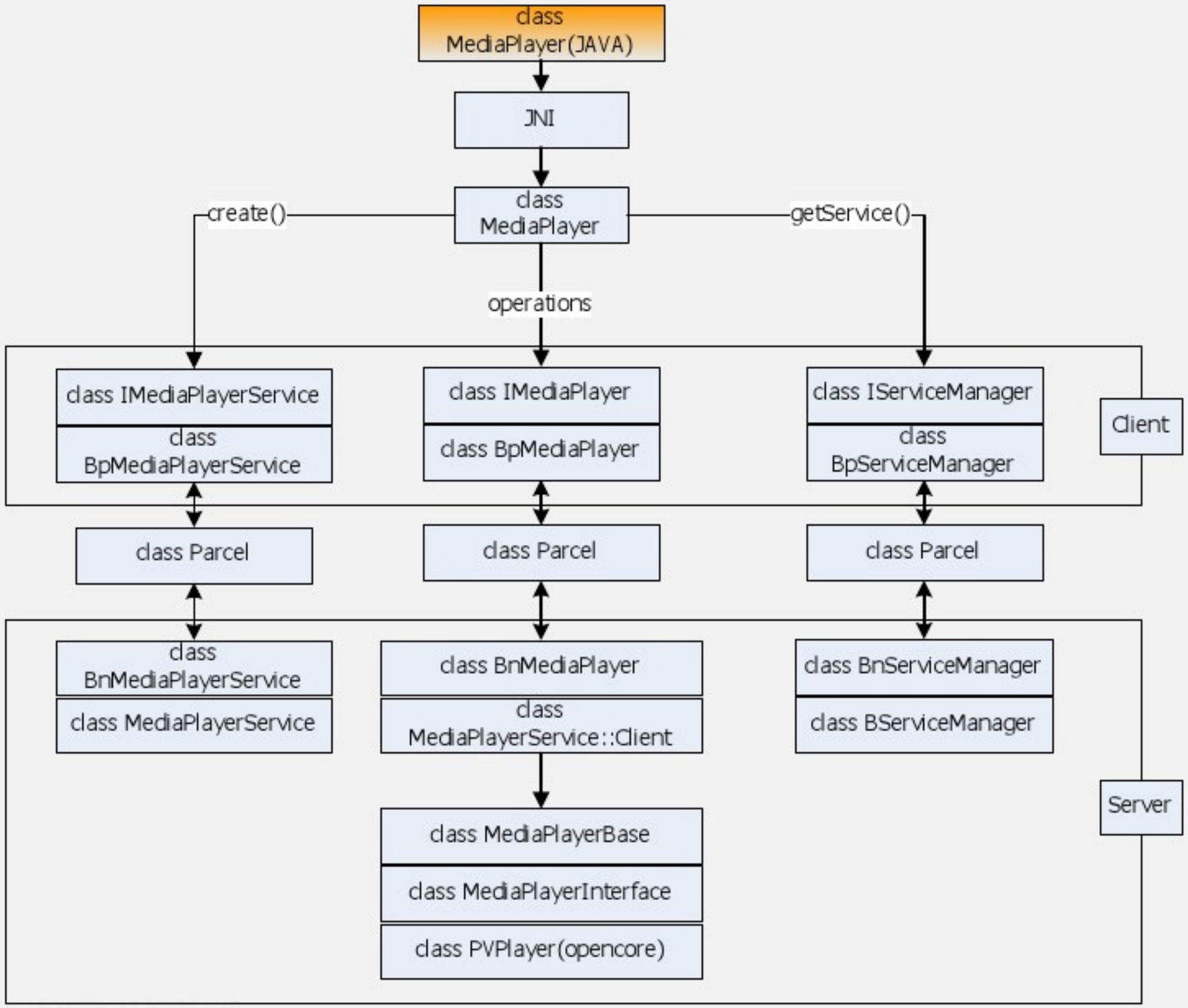
媒体播放器	功能
MIDIFile-Player	主要负责midi音频文件的播放（*.mid、*.midi、*.smf、*.xmf、*.imi、*.rtttl、*.rtx、*.ota）
Vorbis-Player	主要负责ogg格式的音频文件的播放(*.ogg、*.oga)
PVPlayer	负责除以上2种播放器之外的所有媒体的播放

媒体层结构-目录结构

目录名	代码描述
media/java	存放java层媒体相关代码
media/jni	媒体层jni接口代码库
media/libmedia	媒体层client端代码库
media/libmediaplayerservice	媒体层server端代码库
media/mediaserver	媒体层服务端应用程序
libs/audioflinger	媒体层audioFlinger代码库
libs/surfaceflinger	媒体层surfaceFlinger代码库

注：以上目录均是基于frameworks/base

MediaPlayer-类层次结构图



MediaPlayer类说明

- 1、上层的JAVA程序通过JNI调用到底层C++代码中的MediaPlayer类来实现媒体流播放；
- 2、MediaPlayer类首先通过IServiceManager的getService接口获取一个名称为“media.player”的服务，然后通过调用IMediaPlayerService的create接口创建一个mediaplayer播放器，之后所有的操作均通过新创建的这个mediaplayer播放器进行处理，接口为IMediaPlayer；
- 3、图中所有的以BpXXXX的类均是一个代理类，负责将客户端的请求通过Binder机制发送到服务端实现消息功能的转发，在服务端都有对应的一个BnXXXX的子类负责实现具体的功能；
- 4、最终的媒体流的播放是通过MediaPlayerInterface接口，调用底层的opencore库来实现的；

opencore-目录结构

目录名	说明
android/	存放opencore与android系统接口代码主要是一些MIO的实现代码
fileformats/	文件格式解析与合成相关代码
nodes/	opencore节点源码
oscl/	系统api接口库
pvmi/	opencore媒体框架代码
build_config/	编译makefile文件
codecs_v2/	编解码相关代码，其中的omx目录是openMAX的实现代码
engines/	媒体相关引擎实现代码（player, author,2way）
extern_libs_v2/	openMAX头文件
pvplayer.cfg	opencore配置文件，配置编解码库

opencore-编译

- 编译opencore的代码需要到opencore的根目录下执行mm命令编译，不能在opencore的子目录下执行mm编译。

- 重新编译所有代码：

find . -name “*.cpp” –o name “*.h”|xargs touch

opencore-调度器

- OsclExecScheduler是opencore中一个用户级的调度器，其主要功能如下：

A、该调度器负责实现opencore主线程循环

B、调度需要运行的活动对象（ActiveObject）

C、负责对象的定时处理

D、在没有对象需要运行的时候，自动进入休眠状态

注： OsclExecScheduler本身并不创建任何线程

opencore-节点(Node)

在opencore中使用Node来完成某个指定的功能，例如：文件格式解析(ParseNode)，编解码节点（EncodeNode、DecNode）、视频输出节点（MediaOutputNode）

节点通过提供通用的接口函数来供上层调用儿不需要知道具体的节点类型；

节点之间通过port通讯来传递信息：媒体数据；

一般的节点都继承自OsclTimerObject或OsclActiveObject来实现调度，并实现特定的Run()函数来处理消息和命令

opencore-节点(Node)

文件	说明
src/pvmf_node_interface.cpp	Node接口定义源文件
include/pvmf_node_interface.h	Node接口定义头文件

以上文件目录均是基于：pvmi/pvmf/目录

opencore-端口(Port)

1、Port用来实现Node之间的媒体数据传输，每个port中都包含一个输入消息队列和一个输出消息队列用来存放收到和待发送的消息；

2、Port在进行消息传递的时候必须先连接Connect()，这样Port才知道从哪里接收数据或者向哪里发送数据；

3、一般port并不处理消息，而是通过接口通知Node有消息进来，由Node来进行消息的处理；

4、Port之间实现了消息的拥塞处理，在本端输入队列已满的时候会通知对端停止数据发送，而在本端可以接收数据的时候通知对端可以继续发送数据；

opencore-端口(Port)

重要的Port消息：

消息名称	消息说明
PVMF_PORT_ACTIVITY_OUTGOING_MSG	有消息需要发送
PVMF_PORT_ACTIVITY_INCOMING_MSG	有消息已经接收
PVMF_PORT_ACTIVITY_OUTGOING_QUEUE_BUSY	输出队列忙
PVMF_PORT_ACTIVITY_OUTGOING_QUEUE_READY	输出队列准备好，可以存放待发送消息
PVMF_PORT_ACTIVITY_CONNECTED_PORT_BUSY	对端输入队列忙
PVMF_PORT_ACTIVITY_CONNECTED_PORT_READY	对端输入队列准备好，可以接收消息

opencore-端口(Port)

1. 在NodeA调用QueueOutgoingMsg()发送消息PortA正忙时，会返回PVMFErrBusy; 在输出消息队列由空闲时, PortA会通过PVMF_PORT_ACTIVITY_OUTGOING_QUEUE_READY消息通知NodeA, 此时NodeA需要再调用一次QueueOutgoingMsg()函数即可完成消息的发送; PVMF_PORT_ACTIVITY_OUTGOING_QUEUE_READY消息的发出是在PortA->Send()函数中调用的(在iOutgoingQueue.iBusy = true时);

2. 在NodeA调用Send()返回Busy后，PortB会异步的通过调用PortA->ReadyToReceive()函数来通知PortA可以发送消息到PortB(在NodeB调用DequeueIncomingMsg()函数时，PortB会检查是否需要调用PortA->ReadyToReceive());

opencore-端口(Port)

代码实现：

pvmi/pvmf/src/pvmf_port_base_impl.cpp

pvmi/pvmf/include/pvmf_port_interface.h

pvmi/pvmf/include/pvmf_port_base_impl.h

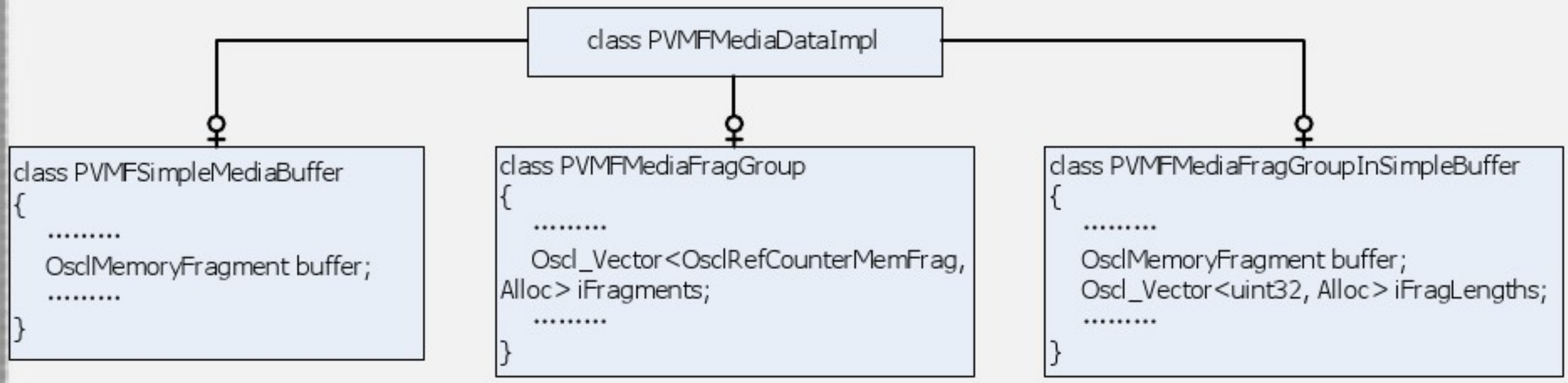
opencore-消息(Msg)

1、在opencore中媒体数据在各个Node之间的传递全部是通过Msg的方式，在传递过程中一般不会进行数据的拷贝，这样可以提高系统效率；

2、在消息处理完成后，会将占用的数据缓冲区归还给发送端，以便发送端可以重复利用；

3、在opencore中提供了多种限制用户内存使用的机制，防止用户无限耗尽系统内存；

opencore-消息(Msg)



说明:

A. class PVMFSimpleMediaBuffer: 单缓冲区类;

B. class PVMFMediaFragGroup: 多缓冲区类, 存在一个缓冲区数组;

C. class PVMFMediaFragGroupInSimpleBuffer: 利用单缓冲区来实现多缓冲区, 原理是在一个单缓冲区中设置一个记录每个单块长度的数组;

D. struct OsclMemoryFragment: 记录申请的缓冲区开始指针和长度信息

```
struct OsclMemoryFragment
{
    void *ptr;
    uint32 len;
};
```

```
class PVMFMediaData
{
    .....
    PVMFMediaMsgHeader* hdr_ptr;
    OsclSharedPtr<PVMFMediaDataImpl> impl_ptr;
    .....
}
```

Thanks!