**UniPlayer独立升级方案**

**背景**

早期的uniplayer为便于快速上线，播放器实现是基于独立jar包，由android framework直接调用，注入到app进程中进行工作的。随着OTA若干个版本迭代及业务需求发展，发现这种方式存在着如下问题：

1.播放器直接在app中工作，如果出现播放器端的fatal exception，

会导致app一起crash

2.目前的uniplayer大部分code实现都是java，与app在同一个进程工作时，发现okhttp网络服务在某些case下存在兼容性问题

3.用户端发现的一些source端兼容性问题以及一些因码流异常导致的av sync问题必须等到下个ROM OTA版本上线后才能解决，但实际播放器端解决问题的周期用不了一个ROM OTA的周期

4.一些来自产品的技术需求方案可能比较紧急，但因为ROM OTA升级周期的原因导致时间拉的比较长，延误品牌影响力宣传

为应对uniplayer与ROM耦合的弊端，提出了uniplayer以apk的形式独立工作、独立升级的想法，并进行技术论证，证明其是可行的，采用uniplayer独立工作、升级，有如下好处：

1.uniplayer的工作空间与app进程解耦，提高各自工作的稳定性

2.可以及时应付部分突发问题，及时推出迭代版本

3.架构清晰，逻辑与实现分开，便于推出独立应用与第三方合作并独立进行feature升级

**技术框架**

**1、简述**

Code路径：\device\whaley\common\UniPlayer\service

把player实例ExoPlayer放在一个app的service中，通过binder与MediaPlayer通信。

UniPlayer独立升级的app是WhaleyPlayerWorker.apk，包名cn.whaley.playerworker，app中的service是PlayerWorkerService。

Binder通信涉及到三个类：UniPlayerServiceImpl和UniPlayerService、UniPlayerServiceClient。

UniPlayerServiceImpl在MediaPlayer进程中，属于UniPlayerService的Proxy端，负责与服务端的UniPlayerService交互。

UniPlayerService在PlayerWorkerService中，属于WhaleyPlayerWorker进程，单例模式。成员变量UniPlayerServiceClient，UniPlayerServiceClient封装ExoPlayer实现，接收UniPlayerServiceImpl传过来的命令和参数。

3个AIDL文件IUniPlayerListener、IUniPlayerService、IUniPlayerServiceClient。

4个序列化对象：TrackInfoParcel、MetadataParcel、InvokeRequestParcel、InvokeReplyParcel。

aidl文件、序列化类等client和service共用部分，打包在jar库：uniplayer.service.aidl.jar中

Service端code打包在静态java库uniplayer.service.stub中，最终被编译进WhaleyPlayerWorker apk

设置系统属性persist.player.bindserver为true，在播放音视频时会走入独立升级code。

下面讲述UniPlayer独立升级后，播放视频时code的调用流程。

**2、3、4节属于MediaPlayer进程，binder的Client端:**

**2、UniPlayer抽象类**

import android.media.UniPlayer

code路径：

frameworks/base/media/java/android/media/UniPlayer.java

MediaPlayer.java中通过反射得到UniPlayer的子类实例如下：

mUniPlayer = (UniPlayer)Class.forName("cn.whaley.UniPlayerImpl").newInstance();

UniPlayerImpl类code路径：device/whaley/common/UniPlayer/cn/whaley/UniPlayerImpl.java

**2.1、UniPlayer创建逻辑**

MediaPlayer.java的无参构造函数中调用bridgePlayerInit()，来判断是否走UniPlayer。

代码逻辑：根据属性persist.player.type判断

playerTypeStr = SystemProperties.get("persist.player.type", "0")

playerTypeStr是1、999、2、1000时playerType = 1 goUniPlayer = true

反射得到UniPlayerImpl对象：

mUniPlayer = (UniPlayer)Class.forName("cn.whaley.UniPlayerImpl").newInstance();

mUniPlayer.createPlayer(playerTypeStr);创建底层exoplayer对象

mUniPlayer.setMediaPlayer(this);

mUniPlayer.setHandler(mEventHandler);设置消息回传handler，用于监听UniPlayer的notify消息。

**2.2、MediaPlayer与UniPlayer接口对接**

setDisplay、setSurface、setDataSource、prepare 、start 、stop 、pause 、getMetaData 、release 、reset、setOnXXXListener等方法直接调用mUniPlayer相应方法。

**2.3、UniPlayer数据成员**

protected MediaPlayer mMediaPlayer = null 保存MediaPlayer对象。

protected Handler mHandler = null 保存MediaPlayer的handler。

重要方法：createPlayer() 创建底层player对象。

**3、UniPlayerImpl**

继承UniPlayer，编译在uniplayer.jar中。

Code路径：\device\whaley\common\UniPlayer\cn\whaley

private UniPlayer player

private String type

createPlayer根据player类型type选择要构建的真实Player：UniPlayerServiceImpl、UniPlayerExoImpl()或UniPlayerExo2Impl等。并把MediaPlayer、Listener和Handler传入构建的真实player中，独立升级走UniPlayerServiceImpl。

当前通过设置属性persist.player.bindserver为true，来进入UniPlayerServiceImpl

**4、UniPlayerServiceImpl**

Code路径：\device\whaley\common\UniPlayer\cn\whaley

**4.1、构造函数**

保存playerType，传到UniPlayerServiceClient端，判断走哪一个Player。

新建Handler：EventHandler，接收UniPlayerServiceClient端的notify消息。

**4.2、DeathRecipient**

mDeathRecipient，死亡通知对象重写binderDied接口，发送MEDIA\_ERROR\_SERVER\_DIED消息到MediaPlayer对象。在获取UniPlayerService的代理后，把mDeathRecipient注册到Service端，监听UniPlayerService的生命周期。

**4.3、setDataSource**

1、Setdatasource中真正建立与service端的连接，通过ServiceManager.getService获取UniPlayerService代理。

2、新建UniPlayerListener，IUniPlayerListener的stub实现，保存4.1中新建的EventHandler，负责接收UniPlayerService端代理发送的通知消息。在EventHandler中用mHandler或Listener把消息转发给MediaPlayer。

3、新建UniPlayerServiceClient，用UniPlayerService.createPlayer(playerType)得到UniPlayerServiceClient的代理，保存在mUniPlayerServiceClient中与UniPlayerServiceClient交互。

4、通过UniPlayerServiceClient.registerCallback把UniPlayerListener的代理注册到UniPlayerServiceClient的RemoteCallbackList中，UniPlayerServiceClient通过RemoteCallbackList发送通知消息给UniPlayerListener。

5、attachNewPlayer，保存新建的IUniPlayerService、UniPlayerListener、IUniPlayerServiceClient对象，并释放之前的。

**4.4、其它方法**

setSurface、start、stop、getTrackInfo、release、getMetaData、invoke等，只是调用IUniPlayerServiceClient相对应的方法，实现在UniPlayerServiceClient中

**下面是service端，属于WhaleyPlayerWorker进程:**

**5、PlayerWorkerService**

把UniPlayerService单例运行在PlayerWorkerService进程中，注册到ServiceManager中，如下：

ServiceManager.addService(UniPlayerService.SVC\_NAME, (IBinder)UniPlayerService.getInstance(this));

**6、aidl文件**

IUniPlayerService.aidl、IUniPlayerServiceClient.aidl、IUniPlayerListener.aidl

IUniPlayerService只有一个方法：createPlayer，新建UniPlayerServiceClient对象并返回，保存在UniPlayerService的成员ArrayList <UniPlayerServiceClient>中。release后从此ArrayList中删除。

UniPlayerServiceClient中的方法与UniPlayer一一对应。

IUniPlayerListener中只有一个notify(int msg, int ext1, int ext2, String s)方法，接收UniPlayerServiceClient发送过来的通知消息，解析后转发给UniPlayerServiceImpl的mEventHandler

序列化类：TrackInfoParcel、MetadataParcel、InvokeRequestParcel、InvokeReplyParcel，封装trackinfo、metadata、invoke数据内容。

**7、UniPlayerService**

是一个binder通信中的servcie，已经注册在ServiceManager中，功能：新建UniPlayerServiceClient实例并保存其引用。

**8、UniPlayerServiceClient**

封装EXOPlayer等真实的播放器对象，成员RemoteCallbackList保存一个IUniPlayerListener的代理。

setSurface：由于binder数据传输时，会在service进程新建一个新的Surface，所以在UniPlayerServiceClient被销毁或再次setSurface时，需要在UniPlayerServiceClient中主动释放这个新建的Surface

HandlerThread：mEventHandler，接收EXOPlayer发送的通知消息，通过IUniPlayerListener发送到UniPlayerServiceImpl

private UniPlayer player = null

注意事项：

1、UniPlayerServiceClient封装的EXOPlayer等需要继承抽象类UniPlayer

2、EXOPlayer等player发送的Message里面的obj需要转换成String类型或null对象，并且String类型有固定结构：

A、必须是JSON结构

B、json结构的第一个key-value必须是：

jsonText.key(UniPlayerServiceName.MSG\_OBJ\_TYPE);

jsonText.value(UniPlayerServiceName.MSG\_OBJ\_xxx);自定义

C、在UniPlayerListener中解析该String结构，复原player中的真实数据结构，传给MediaPlayer。

以上code的具体实现细节请自行查看代码。