互动直播

Android 端集成

产品文档

腾讯云

文档目录

Android 端集成
下载代码

最近更新时间：2018-06-15 18:32:46

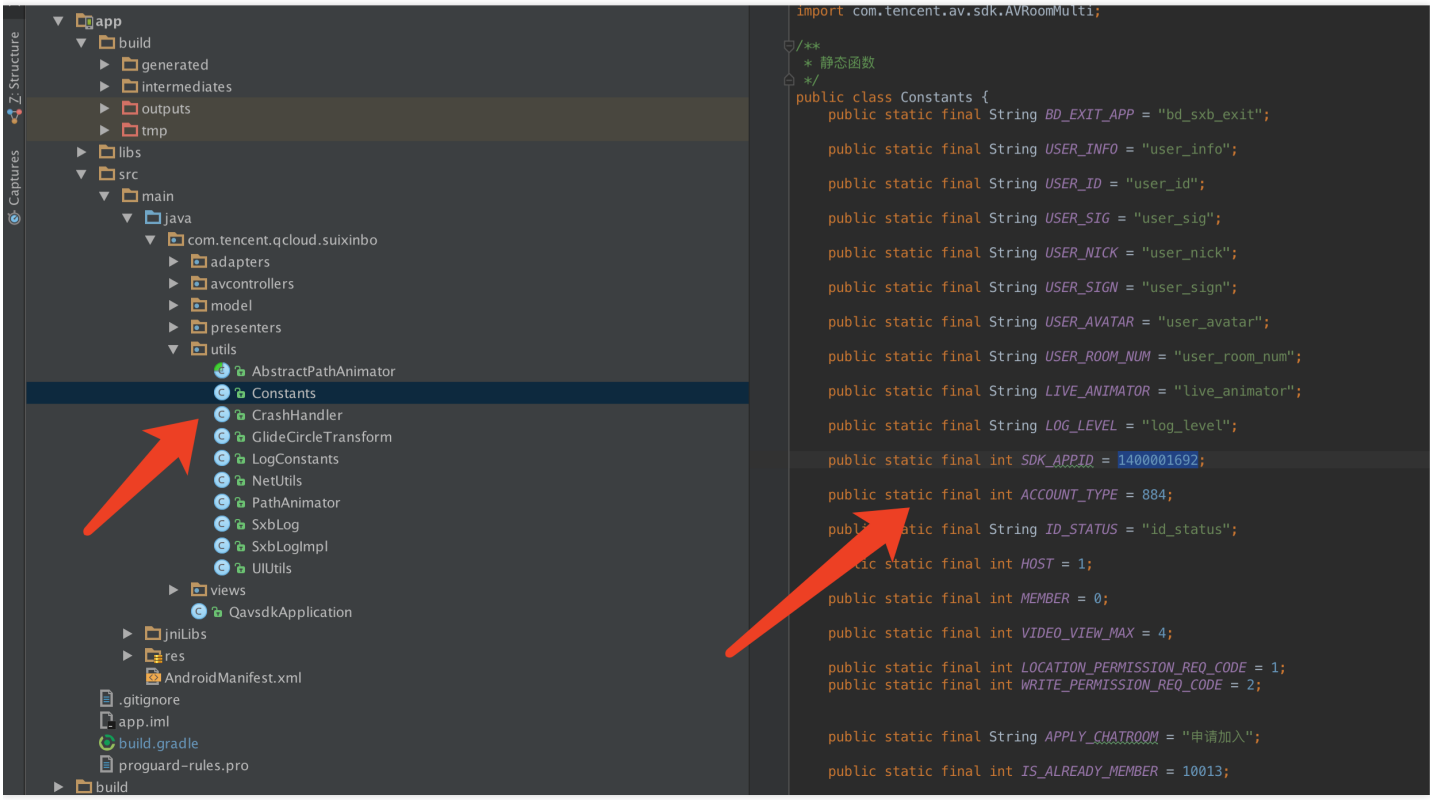
下载 Demo

目前在 GitHub 上提供了两个示例：

- 随心播 演示了包括界面和后台交互的完整的直播流程。
- 简单直播 最简单的互动直播示例，演示了最关键的几个接口的调用。

修改配置

把随心播代码中的 appid 和 accountType 修改成开发者自己的。



运行

编译运行工程，在启动界面选择随心播。

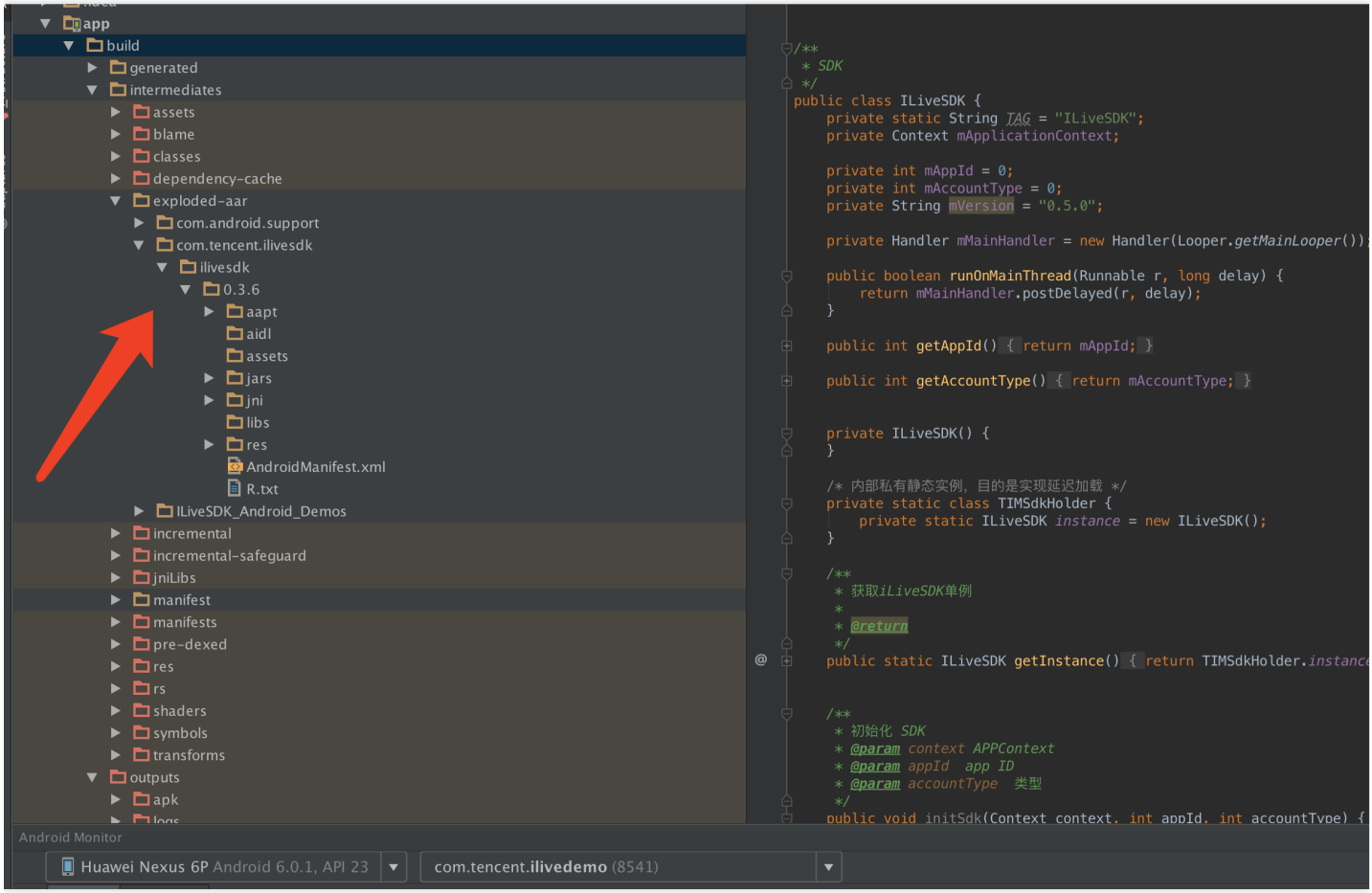
集成到开发者自己的代码工程里

引入 SDK

- **aar 方式集成**（强烈推荐）

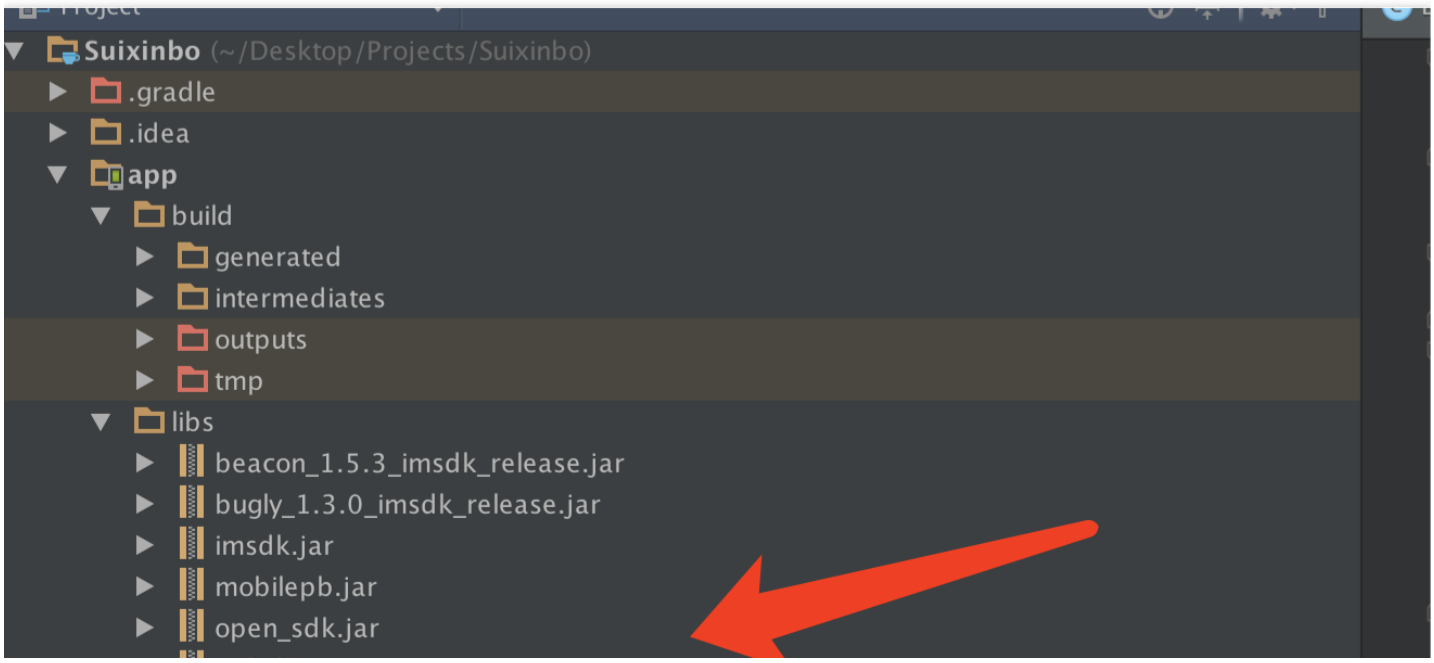
如果您使用的 Android Studio 开发，那么导入 iliveSDK 非常简单。只需两行代码就可以搞定了 。(X.X.X 替换成对应版本号 比如 1.0.1)。同步完成之后可以在 build 文件夹中找到 livesdk 和 ilivesdk 文件夹。

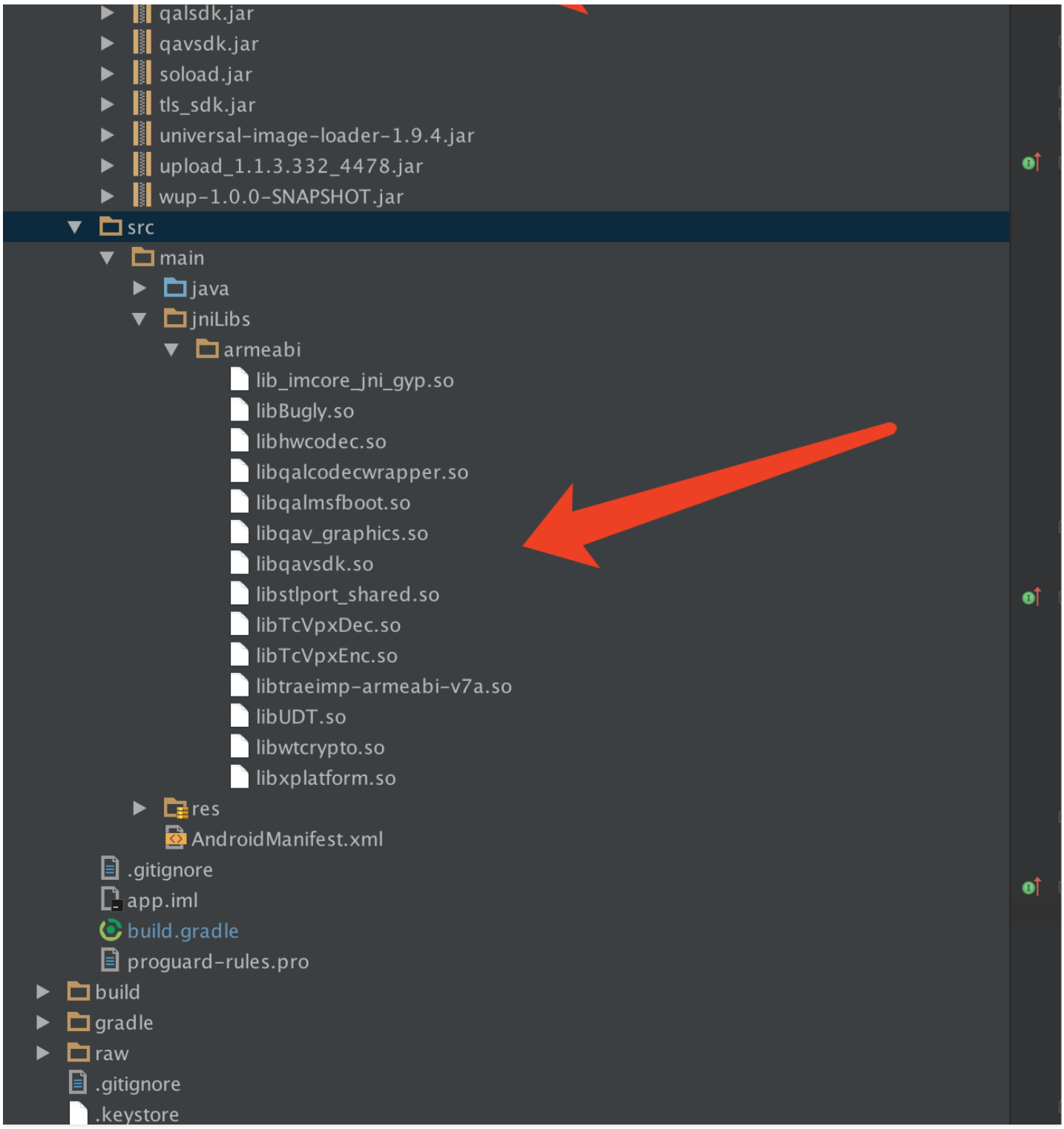
```
直播业务功能
compile 'com.tencent.livesdk:livesdk:X.X.X'
核心功能
compile 'com.tencent.ilivesdk:ilivesdk:X.X.X'
```



- **传统库类方式集成**

在腾讯云官网 下载音视频库类。需要把 so 文件和 jar 包文件分别放到对应 jnilibs 和 libs 里面。





配置服务修改后台 server 地址

目前随心播由业务后台（开源可复用，源码）主要用来维护直播房间列表。如果复用随心播客户端代码，需要修改随心播后台地址为业务方自己部署的服务器地址。

接口	说明

接口	说明
GET_MYROOMID	获取自己分配的房间号
NEW_ROOM_INFO	创建新房间
STOP_ROOM	退出房间
GET_LIVELIST	获取房间列表
SEND_HEARTBEAT	房间心跳
GET_COS_SIG	图片上传相关

```
/**
 * 网络请求类
 */
public class OKhttpHelper {
    private static final String TAG = OKhttpHelper.class.getSimpleName();
    private static OKhttpHelper instance = null;
    public static final String GET_MYROOMID = "http://182.254.234.225/sxb/index.php?svc=user_av_room&cmd=get";
    public static final String NEW_ROOM_INFO = "http://182.254.234.225/sxb/index.php?svc=live&cmd=start";
    public static final String STOP_ROOM = "http://182.254.234.225/sxb/index.php?svc=live&cmd=end";
    public static final String GET_LIVELIST = "http://182.254.234.225/sxb/index.php?svc=live&cmd=list";
    public static final String SEND_HEARTBEAT = "http://182.254.234.225/sxb/index.php?svc=live&cmd=host_heartbeat";
    public static final String GET_COS_SIG = "http://182.254.234.225/sxb/index.php?svc=cos&cmd=get_sign";


    public static OKhttpHelper getInstance() {
        if (instance == null) {
            instance = new OKhttpHelper();
        }
        return instance;
    }


    public static final MediaType JSON
            = MediaType.parse("application/json; charset=utf-8");
```

添加混淆配置

```
-keep class com.tencent.**{*;}
-dontwarn com.tencent.**

-keep class tencent.**{*;}
-dontwarn tencent.**

-keep class qalsdk.**{*;}
-dontwarn qalsdk.**
```

配置 service

```
    <!--TLS Qal 一些服务 -->
    <service
        android:name="com.tencent.qalsdk.service.QalService"
        android:exported="false"
        android:process=":QALSERVICE" />

    <receiver
        android:name="com.tencent.qalsdk.QALBroadcastReceiver"
        android:exported="false">
        <intent-filter>
            <action android:name="com.tencent.qalsdk.broadcast.qal" />
        </intent-filter>
    </receiver>
    <receiver
        android:name="com.tencent.qalsdk.core.NetConnInfoCenter"
        android:process=":QALSERVICE">
        <intent-filter>
            <action android:name="android.intent.action.BOOT_COMPLETED" />
        </intent-filter>
        <intent-filter>
            <action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
        </intent-filter>
        <intent-filter>
            <action android:name="android.intent.action.TIME_SET" />
        </intent-filter>
        <intent-filter>
            <action android:name="android.intent.action.TIMEZONE_CHANGED" />
        </intent-filter>
    </receiver>
</application>
```

配置权限

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
<uses-permission android:name="android.permission.GET_TASKS" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.READ_LOGS" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.BROADCAST_STICKY" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />

<!--<uses-feature android:name="android.hardware.camera" />-->
```

删除非 armeabi 架构 so

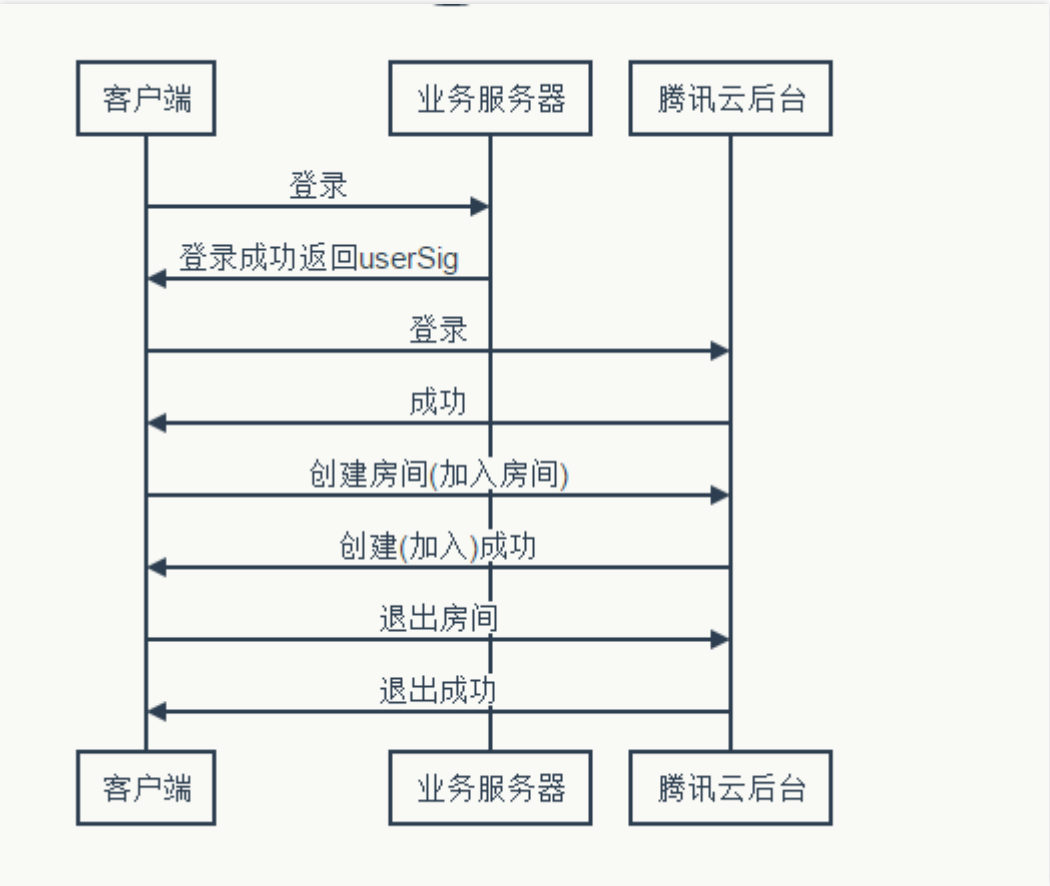
由于目前只支持 armeabi 和 armeabi-v7a 架构，如果工程(或依赖库)中有多架构，需要在 build.gradle 中添加以下配置（如果包含子工程子工程也要加）。

```
android{
defaultConfig{
ndk{
abiFilters 'armeabi', 'armeabi-v7a'
}
}
}
```

直播接口

最近更新时间：2018-09-14 16:11:14

直播流程示例



初始化

接口名	接口描述
initSDK	iLiveSDK 的部分类的预初始化，是所有行为的第一步，告知身份 appId

参数类型	说明
Conext	建议用 AppcalicationContext
int	传入业务方 appid
int	传入业务方 accounttype

示例：

```
//iLiveSDK 初始化
ILiveSDK.getInstance().initSdk(getApplicationContext(), appid, accoutype);
//初始化直播场景
ILVLiveConfig liveConfig = new ILVLiveConfig();
ILVLiveManager.getInstance().init(liveConfig);
```

账号登录

接口名	接口描述
iLiveLogin	使用托管方式或独立模式，在获取到用户的 sig 后，使用登录接口，告知后台音视频模块上线了（包括 avsdk）

参数类型	说明
String	用户 ID，在直播过程中的唯一标识
String	鉴权的密钥 Sig 如果是独立登录方式，是业务方后台计算生成后下发的
ILiveCallBack	帐号登录回调接口。通知上线是否成功

示例：

```
ILiveLoginManager.getInstance().iLiveLogin(ILiveSDK.getInstance().getMyUserId(), "123456", new ILive
CallBack() {
@Override
public void onSuccess(Object data) {
bLogin = true;
Toast.makeText(ContactActivity.this, "login success !", Toast.LENGTH_SHORT).show();
}

@Override
public void onError(String module, int errCode, String errMsg) {
Toast.makeText(ContactActivity.this, module + "|login fail " + errCode + " " + errMsg, Toast.LENGTH_
SHORT).show();
}
});
```

创建房间

接口名	接口描述
createRoom	创建一个直播，只有在初始化和登录成功之后才能创建直播

参数类型	说明
int	房间 ID 房间唯一标识 建议由业务方后台统一分配
ILVLiveRoomOption	房间配置项 可以设置角色 权限 主播 ID 摄像头参数等 具体参考类 ILVLiveRoomOption
ILiveCallBack	创建房间回调接口。通知创建房间是否成功

```
//创建房间配置项
ILVLiveRoomOption hostOption = new ILVLiveRoomOption(null)
.controlRole(Constants.HOST_ROLE)//角色设置
.authBits(AVRoomMulti.AUTH_BITS_DEFAULT)//权限设置
.cameraId(ILiveConstants.FRONT_CAMERA)//摄像头前置后置
.videoRecvMode(AVRoomMulti.VIDEO_RECV_MODE_SEMI_AUTO_RECV_CAMERA_VIDEO);//是否开始半自动接收
//创建房间
ILVLiveManager.getInstance().createRoom(room, hostOption, new ILiveCallBack() {
@Override
public void onSuccess(Object data) {
Toast.makeText(LiveActivity.this, "create room ok", Toast.LENGTH_SHORT).show();
logoutBtn.setVisibility(View.INVISIBLE);
backBtn.setVisibility(View.VISIBLE);
}

@Override
public void onError(String module, int errCode, String errMsg) {
Toast.makeText(LiveActivity.this, module + "|create fail " + errMsg + " " + errMsg, Toast.LENGTH_SHORT).show();
}
});
```

加入房间

接口名	接口描述
joinRoom	观众角色调用加入房间接口

参数类型	说明
int	房间 ID 房间唯一标识 建议由业务方后台统一分配
ILiveRoomOption	房间配置项 可以设置角色 权限 主播 ID 摄像头参数等 具体参考类 ILiveRoomOption
ILiveCallBack	加入房间回调接口。通知加入房间是否成功

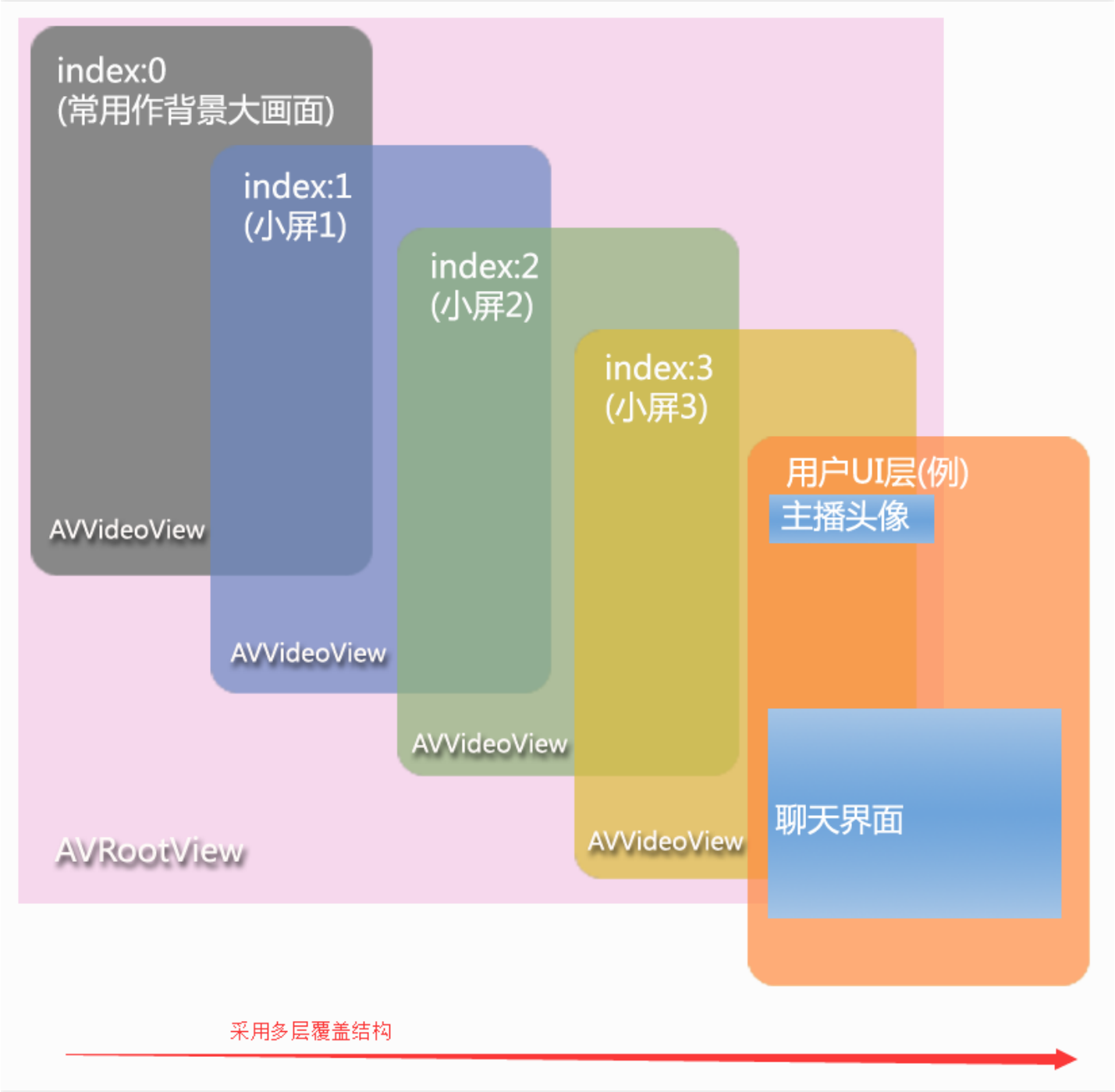
特别注意: 普通观众加入房间时，应该配置authBits无上行权限，仅观看权限。否则会和主播一样走核心机房DC，产生高额费用。

```
//加入房间配置项
ILVLiveRoomOption memberOption = new ILVLiveRoomOption("")
.autoCamera(false) //是否自动打开摄像头
.controlRole(Constants.NORMAL_MEMBER_ROLE) //角色设置
.authBits(AVRoomMulti.AUTH_BITS_JOIN_ROOM | AVRoomMulti.AUTH_BITS_RECV_AUDIO | AVRoom
Multi.AUTH_BITS_RECV_CAMERA_VIDEO | AVRoomMulti.AUTH_BITS_RECV_SCREEN_VIDEO) //权限设置
.videoRecvMode(AVRoomMulti.VIDEO_RECV_MODE_SEMI_AUTO_RECV_CAMERA_VIDEO) //是否开始半
自动接收
.autoMic(false);//是否自动打开 mic
//加入房间
ILVLiveManager.getInstance().joinRoom(room, memberOption, new ILiveCallBack() {
@Override
public void onSuccess(Object data) {
bEnterRoom = true;
Toast.makeText(LiveActivity.this, "join room ok ", Toast.LENGTH_SHORT).show();
logoutBtn.setVisibility(View.INVISIBLE);
backBtn.setVisibility(View.VISIBLE);
}

@Override
public void onError(String module, int errCode, String errMsg) {
Toast.makeText(LiveActivity.this, module + "|join fail " + errMsg + " " + errMsg, Toast.LENGTH_SHOR
T).show();
}
});
```

设置渲染层

渲染层级示例图 在界面层 XML 插入一个 AVRootView，音视频数据最终是通过 AVRootView 渲染出来。考虑多屏互动情况，AVRootView 实际上不是一层 View 而是多层 AVVideoView 的叠加。直播业务默认主播在第 0 层默认最大，其他互动观众分别在 1，2，3 层。每层大小都可以动态调节。



示例：

```
<com.tencent.ilivesdk.view.AVRootView
android:id="@+id/av_root_view"
android:layout_width="match_parent"
```

```
android:layout_height="match_parent"
android:background="@color/white" />
avRootView = (AVRootView) findViewById(R.id.av_root_view);
ILVLiveManager.getInstance().setAvVideoView(avRootView);
```

互动消息和上麦接口

最近更新时间：2018-06-15 18:32:55

发送消息

发送普通文本消息

接口名	接口描述
sendText	发送文本消息 通过消息体类型可以发群消息和 C2C 消息

参数类型	说明
ILVText	文本消息类 可以指定消息类型（群消息或 C2C 消息）消息内容，消息接受者（群则是群 ID，个人则是个人 ID）
ILiveCallBack	发送消息回调接口

发文本示例：

```
//发送消息
ILVText iliveText = new ILVText("ss", "", ILVLiveConstants.GROUP_TYPE);
iliveText.setText("" + textInput.getText());
//发送消息
ILVLiveManager.getInstance().sendText(iliveText, new ILiveCallBack() {
@Override
public void onSuccess(Object data) {
Toast.makeText(LiveActivity.this, "send succ!", Toast.LENGTH_SHORT).show();
}
@Override
public void onError(String module, int errCode, String errMsg) {
}
});
```

发送信令消息

接口名	接口描述
sendCustomCmd	发送信令消息 通过消息体类型可以发群消息和 C2C 消息

参数类型	说明
ILVCustomCmd	信令消息类 可以指定消息类型（群消息或 C2C 消息）消息内容，消息接受者（群则是群 ID，个人则是个人 ID），以及 String 类型参数
ILiveCallBack	发送消息回调接口

上麦邀请示例：

```
//邀请上麦
ILVCustomCmd cmd = new ILVCustomCmd();
cmd.setCmd(ILVLiveConstants.ILVLIVE_CMD_INVITE);
cmd.setType(ILVLiveConstants.C2C_TYPE);
cmd.setDestid("" + memId.getText());
cmd.setParam("");
ILVLiveManager.getInstance().sendCustomCmd(cmd, new ILiveCallBack<TIMMessage>() {
@Override
public void onSuccess(TIMMessage data) {
Toast.makeText(LiveActivity.this, "invite send succ!", Toast.LENGTH_SHORT).show();
}
@Override
public void onError(String module, int errCode, String errMsg) {
}
});
```

为了方便用户自定义信令 iLiveSDk 保留了一个自定义区域。

```
public static final int ILVLIVE_CMD_CUSTOM_LOW_LIMIT = 0x800; //自定义消息段下限
public static final int ILVLIVE_CMD_CUSTOM_UP_LIMIT = 0x900; //自定义消息段上限
```

解析消息

ILVLiveConfig 是全局直播设置，里面可以配置一个消息回调 ILVLiveConfig.ILVLiveMsgListener。通过这个消息回调可以拿到对应的消息类型，然后塞回 ILVLiveManager 进行配置。

参数类型	说明
onNewCmdMsg	iLiveSDK 预定义信令回调，例如上麦、下麦
onNewCustomMsg	用户自定义信令回调
onNewTextMsg	普通文本接收回调

示例：

```
ILVLiveConfig liveConfig = new ILVLiveConfig();
liveConfig.setLiveMsgListener(new ILVLiveConfig.ILVLiveMsgListener() {
@Override
public void onNewTextMsg(String text, String id) {
Toast.makeText(LiveActivity.this, "onNewTextMsg : " + text, Toast.LENGTH_SHORT).show();
}
@Override
public void onNewCmdMsg(int cmd, String param, String id) {
switch (cmd) {
case ILVLiveConstants.ILVLIVE_CMD_INVITE:
Toast.makeText(LiveActivity.this, "onNewCmdMsg : received a invitation! ", Toast.LENGTH_SHORT).sho
w();
ILiveLog.d(TAG, "ILVB-LiveApp|received ");
ILVLiveManager.getInstance().upToVideoMember(ILVLiveConstants.VIDEO_MEMBER_AUTH, ILVLiveCo
nstants.VIDEO_MEMBER_ROLE, new ILiveCallBack() {
@Override
public void onSuccess(Object data) {
}
@Override
public void onError(String module, int errCode, String errMsg) {
}
});
break;
case ILVLiveConstants.ILVLIVE_CMD_INVITE_CANCEL:
break;
case ILVLiveConstants.ILVLIVE_CMD_INVITE_CLOSE:
ILVLiveManager.getInstance().downToNorMember(ILVLiveConstants.NORMAL_MEMBER_AUTH, ILVLiv
eConstants.NORMAL_MEMBER_ROLE, new ILiveCallBack() {
@Override
public void onSuccess(Object data) {
}
@Override
public void onError(String module, int errCode, String errMsg) {
}
});
break;
case ILVLiveConstants.ILVLIVE_CMD_INTERACT_AGREE:
break;
case ILVLiveConstants.ILVLIVE_CMD_INTERACT_REJECT:
break;
}
}
@Override
```

```
public void onNewCustomMsg(int cmd, String param, String id) {
Toast.makeText(LiveActivity.this, "cmd "+ cmd, Toast.LENGTH_SHORT).show();
}
});
```

上麦互动

上麦行为

切换角色 --> 打开摄像头 -->渲染画面（如果开启自动渲染，则省略这一步）。

接口	接口描述
upToVideoMember	包括切换角色，权限，场景，打开摄像头 合成一步

示例：

```
ILVLiveManager.getInstance().upToVideoMember(ILVLiveConstants.VIDEO_MEMBER_AUTH, ILVLiveConstants.VIDEO_MEMBER_ROLE, new ILiveCallBack() {
@Override
public void onSuccess(Object data) {
}
@Override
public void onError(String module, int errCode, String errMsg) {
}
});
```

下麦行为

切换角色 --> 关闭摄像头 -->结束渲染。

接口	接口描述
downToNorMember	包括切换角色、权限、场景、关闭摄像头，合成一步

示例：

```
ILVLiveManager.getInstance().downToNorMember(ILVLiveConstants.NORMAL_MEMBER_AUTH, ILVLiveConstants.NORMAL_MEMBER_ROLE, new ILiveCallBack() {
@Override
public void onSuccess(Object data) {
}
```

```
@Override
public void onError(String module, int errCode, String errMsg) {
}
});
```