微吼直播 SDK for Android

V2.1.1

目录

一、 修订记录

日期	版本号	描述	修订者
2016-04-21	V2.1.1	初稿	xy

二、 简介

本文档为了指导开发者更快使用 Android 系统上的 “自助式网络直播服务 SDK”，默认读者已经熟悉 IDE 的基本使用方法（本文以 Eclipse 为例），以及具有一定的编程知识基础等。

支持的产品特性如下:

分类	特性名称	描述
发起直播	支持编码类型	音频编码：AAC，视频编码：H.264
	支持推流协议	RTMP
	视频分辨率	640*480
	屏幕朝向	横屏、竖屏
	闪光灯	开/关
	静音	开/关
	切换摄像头	前、后置摄像头
	目标码率	使用软编，码率固定在 300-400 之间，暂不可修改
	支持环境	Android 4.0 以上，
观看直播	支持播放协议	RTMP/HLS
	延时	RTMP: 2-4 秒，HLS：20 秒左右
	支持解码	H.264
观看回放	支持协议	HLS
权限	第三方 K 值认证	http://e.vhall.com/home/vhallapi/embed 支持客户自己的权限验证机制来控制观看直播、观看回放的权限
其它	代码安全	支持代码混淆

三、 权限开通申请

请点击 API&SDK 权限申请 立即沟通申请，申请后客户经理会在线上与您直接联系。

审核通过后，可以获取开发应用的权限信息：App_Key、Secret_Key、App Secret_Key， 立即查看。

四、 SDK 使用准备

1、 下载 SDK&DEMO

从 github 下载：https://github.com/vhall20/vhallsdk live android

2、 开发环境要求

Pc 操作系统：64window 系统
JDK: 1.6 以上
Eclipse : 建议使用官方已经集成的 Eclipse ,谨慎使用 Android studio
Android: 4.0 以上
备注： Android 设备操作系统需要 4.0 以上, 需要访问手机硬件,暂不支持模拟器开发

3、 需要导入的 Jar

Vhallsdk.jar

4、 动态库 SO

Libdynload.so

Libffmpeg.so

Libjingle.so

libstlport_shared.so

libVinnyLive.so

5、 权限及配置

```
<uses-permission android:name="android.permission.CAMERA" />

<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

<uses-permission android:name="android.permission.RECORD_AUDIO" />

<uses-permission android:name="android.permission.RECORD_VIDEO" />

<uses-permission android:name="android.permission.INTERNET" />

<uses-permission android:name="android.permission.WRITE_SETTINGS" />

<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />

<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

<uses-permission android:name="android.permission.FLASHLIGHT" />
```

<uses-permission android:name="android.permission.CHANGE_WIFI_MULTICAST_STATE" />

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

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

备注：主要是访问网络.Wife.还有手机硬件的 Camera

6、 代码混淆

若 app 打包需要进行代码混淆，请添加：

-dontwarn - com.vhall.**

-dontwarn - com.vinny.**

-keep class com.vhall.**{*;}

-keep class com.vinny.**{*;}

五、 快速接入介绍

1、 权限认证信息配置

以下信息配置到文件里。文件名称：src\com\example\rtmpdemo\Constants.java

```
public class Constants {
    public static final String APP_KEY = "";
    public static final String APP_SECRET_KEY = "";
}
```

其中：App_Key、App_Secret_Key：从此页面获取到， 立即查看权限信息

2、 发起直播流程

第一步 预览采集 Camera：

在 Activity 的 onCreate 方法中，创建 CameraNewView，并且初始化

第二步 发起直播：

设置开始按钮，在可点击情况下调用 NativeLive.StartPublish(String Url)。

第三步 建立 Socket

发起直播成功后 ，连接 Socket 建立参会 new ZReqEngine().new Attend（）

第四步 停止直播：

设置停止按钮，在可点击情况下调用 NativeLive.StopPublish()

3、 观看直播流程

第一步 创建观看 View

在 Activity 的 onCreate 方法中，创建 PlayView，并且初始化

第二步 开始观看

NativeLive.StartRecv(String watchUrl)

第三步 停止观看

NativeLive.StopRecv()

4、 基础功能说明

备注： 以下功能根据各自需求自行选取使用，设置后即可显示

Step 1 : 初始化播放器

```
PlayView playview = new PlayView(GLSurfaceView)
playview.init(with,height)
playview.updataScreen(byte []  Y ,byte []  U ,byte [] V )
playview.updataScreenAll(byte []  YUV)
```

Step 2 : 初始化音频

```
AudioPlay audio = new AudioPlay()
Audio.init(int sampleRate, int channelConfig, int audioFormat)
Audio.play(byte[] data, int size)
Audio.destory()
```

Step 3 : 使用自定义 CameraNewView

自定义 view 已经对手机摄像头（Camera） 做好了处理，实时采集每一帧数据，包括摄像头的切换，闪光的开启，用户可以自行调用。

```
mCameraView=(CameraNewView) this.findViewById(R.id.cameraview);
```

5、 响应事件（回调）

直播观看过程中的回调 LiveCallback：

在你当前开启直播的页面，初始化直播回调：

```
LiveCallback livecallback = new LiveCallback ;
```

实现其中的回调方法：

```
public void notifyVideoData(byte[] data)
public int notifyAudioData(byte[] data, int size)
public void notifyEvent(int resultCode, String content)
public void onH264Video(byte[] data, int size, int type)
```

在当前 Activity onCreate 方法中加入回调

```
LiveObs.setCallBack(livecallback);
```

其中 notifyEvent（resultCode ,content） 返回的 resultCode 是底层定义的直播状态，需要用户对其自行处理。 状态定义如下：

```
public static final int OK_PublishConnect        = 0;   //直播连接服务器成功
public static final int ERROR_PublishConnect  = 1;   //直播连接服务器失败
```

```
public static final int OK_WatchConnect    = 2; //观看直播连接服务器成功
public static final int ERROR_WatchConnect     = 3; //观看直播连接服务器失败
public static final int StartBuffering         = 4;//开始缓冲
public static final int StopBuffering          = 5;//停止缓冲
public static final int ERROR_Param            = 6;//错误参数
public static final int ERROR_NeedReconnect    = 7;//错误 需要重新连接
public static final int ERROR_Send             = 8;//发送直播流失败
public static final int INFO_Speed_Upload      = 9;//上传速度 Kbps 单位
public static final int INFO_Speed_Download    = 10;//下载速度 Kbps 单位
public static final int INFO_NetWork_Status    = 11;//网络状态
public static final int INFO_Decoded_Video     = 12;//视频解码
public static final int INFO_Decoded_Audio     = 13;//音频解码
public static final int INFO_Record_Audio      = 20;//录音
```

6、 发起直播详解

用户使用 VhallSDK 发起直播　需要首先了解几个重要的类

LiveParam　直播中所需的重要参数(这些参数会被传入底层,错误的参数会让 Activity 报错 ，目前默认使用分辨率 640*480 更高的分辨率暂不支持)

ConnectionChangeReceiver　用来检测网络变化

定义的一些常量

```
public static final int NET_ERROR = 0;
public static final int NET_UNKNOWN = 1;
public static final int NET_2G3G = 2;
public static final int NET_WIFI = 3;
```

第一步　初始化直播信息

创建 Activity ,初始化自定义 view CameraNewView， 此时的 CameraNewView 被创建，开启 PreviewCallback 回调,实现 onPreviewFram(),获取 Camera 采集的每一帧的数据 ，将此数据传递底层处理

```
mCameraView = (CameraNewView) this.findViewById(R.id.cameraview);
mCameraView.init(param, this, new RelativeLayout.LayoutParams(0, 0));
mCameraView.startPublish()   //  此方法须直播回调中返回 OK_PublishConnect 才能调用
```

此处代码必须添加

```
NativeLive.CreateVinnyLive();      //  创建 VinnyLive 对象
LiveObs.setCallback(mLiveCallBack);   // 设置直播回调
NativeLive.EnableDebug(true);      // 是否打开 Debug 模式 （会打印日志）
NativeLive.AddObs();      //添加直播的监听
```

第二步　创建 button ， 开启直播 。

这时调用底层方法，连接推流地址，需要传递一个参数 Path ，Path 为流的地址 。 判断这个方法，如果返回的是 0 ，则连接成功 ，返回非 0，则连接失败

NativeLive.StartPublish（path） // 连接推流地址

第三步 处理直播时的回调

当直播成功之后，处理直播时时返回的信息 ，详细方法可以参考直播过程的回调 LiveCallback ，处理方案根据各自需求自行处理。

第四步 停止直播

CameraView.stopPublish();

NativeLive.StopPublish()

7、 RTMP 观看直播详解

第一步 初始化 PlayView AudioPlay

playView 初始化时需要传入 GLSurfaceView ，使用它需要用户自定义一个渲染器（render）不过这里在初始化时已经定义好，用户可以直接使用

```
mPlayView = new PlayView(glSurfaceview);
mPlayView.init(width , height) // 传入初始化的宽高;
```

此处代码必须添加

```
NativeLive.CreateVinnyLive();     //  创建 VinnyLive 对象
LiveObs.setCallback(mLiveCallBack);   // 设置直播回调
NativeLive.EnableDebug(true);     // 是否打开 Debug 模式 （会打印日志）
NativeLive.AddObs();     //添加直播的监听
```

获取用户的活动 ID（此 ID 需要在 PC 上取得） ，

请求 ZReqEngine.watch(id , APP_KEY , APP_SECRET_KEY , name , email , password , new ReqCallback())

```
@param id                    //  活动 ID 必传
@param APP_KEY           //  app_key
@param APP_SECRET_KEY    //  app_secret_key
@param name              //  必传
@param email             //  必传
@param password          //  活动如果有 K 值需要传
@callback ReqCallback        //  传入回调 获取返回的参数
```

请求成功之后，会在 ReqCallback OnSuccess()返回 Json 参数。

```
@result rtmp_video           //  rtmp 观看直播地址
@result video            //  hls 观看直播回放地址
@result status               //  当前播放状态
@result msg_server           //  建立参会
@result msg_token        //  建立参会
```

请求失败之后，会在 ReqCallback OnFail()返回信息，直接打印即可

建立参会，参会建立成功，可以统计参会人数。

```
ZReqEngine.Attend attend = new ZReqEngine().new Attend(msg_server, msg_token);
```

第二步 创建 button 开始观看

将之前获取的观看地址传入当前方法

Natiive.StarRecv(path)　// 连接接受地址　0 则连接成功 ，非 0 则连接失败

第三步　处理直播时的回调

这时会用到 LiveCallback 中的方法

public void notifyVideoData(byte[] data)　//得到正在直播的视频数据

这时调用 UpdateScreenAll() ，将取得的视频信息传给 PlayView

mPlayView.UpdateScreenAll(data)

public int notifyAudioData(byte[] data, int size)　//得到正在直播的音频数据

这时调用 play() ，将取得的视频信息传给 AudioPlay

备注 ： notifyEvent() 依然需要调用

第四步　停止观看

NativeLive.StopRecv()

stopAudioPlay

attend.disAttend();　// 关闭参会

8、 HLS 观看直播及 HLS 观看回放

初始化 VhallHlsPlayer ，实现 VhallHLSPlayer.Listener

需要用户设置 SurfaceView

```
String userAgent = Util.getUserAgent(this, "VhallAPP");
mMediaPlayer = new VhallHlsPlayer(new HlsRendererBuilder(this, userAgent, path));　// 这里需要传入地址
mMediaPlayer.addListener(mVhallPlayerListener);
mMediaPlayer.setSurface(Surface);　//　设置 SurfaceView
mMediaPlayer.setPlayWhenReady（true）　// 为 true 的时候开始播放
```

实现 VhallHLSPlayer.Listener　所需实现的方法如下

public void onStateChanged(boolean playWhenReady, int playbackState) {} // 当播放状态发生改变的时候

public void onError(Exception e) {}　// 播放错误的时候

public void onVideoSizeChanged(int width, int height, int unappliedRotationDegrees,float pixelWidthHeightRatio) {} // Video 尺寸发生改变的时候

获取播放地址之前需要先获取用户的活动 ID（通过服务器接口获取）

请求 ZReqEngine.watch(id , APP_KEY , APP_SECRET_KEY , name , email , password , new ReqCallback())

```
@param id                    //　活动 ID 必传
@param APP_KEY           //　app_key
@param APP_SECRET_KEY    //　app_secret_key
@param name              //　必传，参会人员姓名，用于统计
```

```
@param email              //  必传，参会人员邮箱，用于统计，并做为用户的唯一标识
@param password           //  可选，活动如果有 K 值需要传
@callback ReqCallback     //  传入回调 获取返回的参数

请求成功之后，会在 ReqCallback OnSuccess()返回 Json 参数
@result rtmp_video        //  rtmp 观看直播地址
@result video             //  hls 观看直播回放地址
@result status            //  当前播放状态
@result msg_server        //  建立参会
@result msg_token         //  建立参会

请求失败之后，会在 ReqCallback OnFail()返回信息，直接打印即可
建立 Socket 连接参会，建立成功后，可以统计参会人数。
ZReqEngine.Attend attend = new ZReqEngine().new Attend(msg_server, msg_token);
```

六、 DEMO 简介

1、 DEMO 简介

DEMO 只针对核心功能进行演示，不包括 UI 界面设计。

2、 主要测试参数说明：

1）活动 ID：指的是客户创建的一个直播活动的唯一标识，Demo 测试时可从 e.vhall.com 的控制台页面上获取到

2) Token ： Demo 测试时可从 http://e.vhall.com/api/test 页面，调用接口 verify/access-token 获取到，有效期为 24 小时

3）码率设置：主要用于视频编码设置，码率与视频的质量成正比，默认值 300，单位 Kbps

4）缓冲时间：延时观看时间

5）分辨率： 640*480

6）K 值： 默认为空，指的是控制直播观看权限的参数，具体使用说明参考第三方 K 值验证

3、 客户 Server 端需提供给 APP 的信息

客户 Server 端需要提供如下信息：

1）Id: 通过客户 Server 端接口获取到，此接口需调用 VHALL 接口 webinar/list 获取。

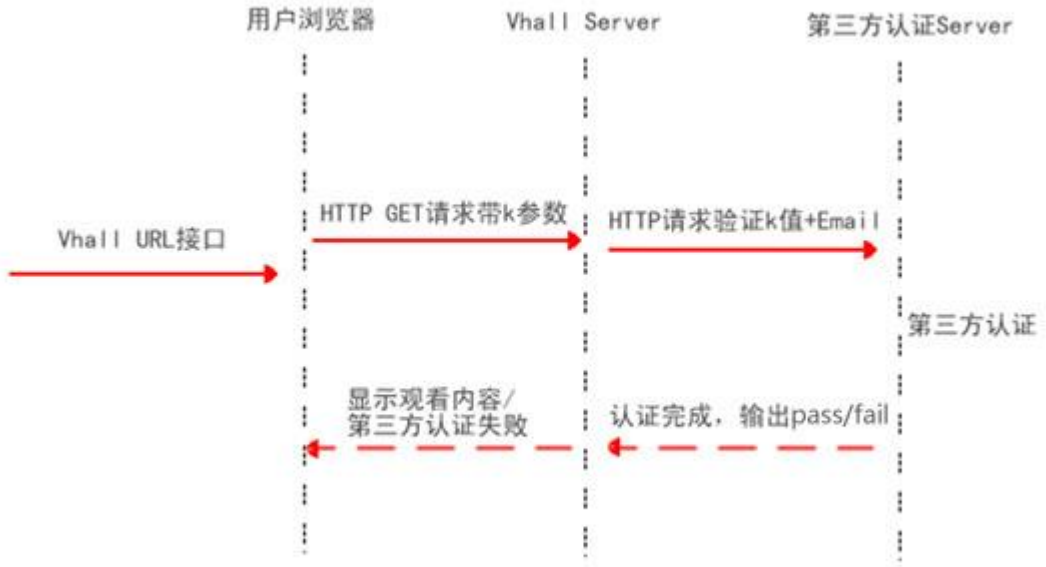
2)AccessToken:通过客户 Server 端接口获取到,此接口需调用 VHALL 接口 verify/access-token 获取。

七、 第三方 K 值认证

观看直播、观看回放的权限控制，支持使用客户的权限验证逻辑。

具体可参考：http://e.vhall.com/home/vhallapi/embed

1、 认证流程



2、 开启设置

1）全局设置： 针对所有的活动配置生效，如果针对单个活动再做配置，以单个活动配置为最终配置。通过接口调用设置 webinar/whole-auth-url 全局配置第三方K值验证URL

2) 针对某个活动的配置方式一：通过页面配置

http://e.vhall.com/webinar/auth/123456789 ，数字表示自己帐号下的活动id

3）针对某个活动的配置方式二：通过接口(webinar/create或webinar/update)设置

3、 K 值使用

1） 网页嵌入或SDK里的调用方法，请务必带上k参数，如果这个参数为空或者没有这个参数，则视为认证失败

●网页嵌入地址类似：

http://e.vhall.com/webinar/inituser/123456789?email=test@vhall.com&name=visitor&k=随机字符串

●SDK里的调用方法,需要传递3个参数name,email,pass

email:可选参数，如果不填写系统会随机生成邮箱地址。 由于email自身的唯一性，我们推荐使用email来作为唯一标识有效用户的字段。对于第三方自有用户数据的系统，也可以使用一些特征ID作为此标识，请以email的格式组织，比如在第三方系统中，用户ID为123456，可在其后添加一个@domain.com,组成123456@domain.com形式的email地址。

name: 可选参数，如果不填写系统会随机生成。此字段表示用户昵称、姓名或其他有意义的字符串。可以为中文，但必须为UTF-8，且经过URL编码(urlencode)。

k： 可选参数，此字段为了提供给第三方可以根据自己的权限系统，验证客户是否可访问直播地址。

ZReqEngine.watch(id , APP_KEY , APP_SECRET_KEY , name , email , password , new ReqCallback())

```
@param id                    //  活动 ID 必传
@param APP_KEY           //  app_key
@param APP_SECRET_KEY    //  app_secret_key
@param name              //  必传
@param email             //  必传
@param password          //  活动如果有 K 值需要传
@callback ReqCallback        //  传入回调 获取返回的参数
```

观看直播 (仅HLS可用)

ZReqEngine.watch(id , APP_KEY , APP_SECRET_KEY , name , email , password , new ReqCallback())

```
@param id                    //  活动 ID 必传
@param APP_KEY               //  app_key
@param APP_SECRET_KEY        //  app_secret_key
@param name                  //  必传，参会人员姓名，用于统计
@param email             //  必传，参会人员邮箱，用于统计，并做为用户的唯一标识
@param password               //  可选，活动如果有 K 值需要传
@callback ReqCallback        //  传入回调 获取返回的参数
```

2） Vhall系统收到用户的接口访问请求后，会向第三方认证URL(auth_url)发送HTTP POST请求，同时将email和k值作为POST数据提交 给第三方认证。由第三方系统验证k值的合法性。如果认证通过，第三方认证URL(auth_url)返回字符串pass,否则的返回fail

注：需要确保您的回调地址支持 multipart/form-data 方式接收 post 数据。

3） Vhall 系统根据第三方认证URL返回值判断认证是否成功。只有收到pass，才能认定为验证成功，否则一律跳转到指定的认证失败 URL，或者提示'非法访问'

4）参数特征

URL请求很容易被探测截获，这就要求第三方系统生成的K值必须有以下特征：

- 唯一性：每次调用接口必须产生不同的K值
- 时效性：设定一个时间范围，超时的K值即失效。
- 如果包含有第三方系统内部信息，必须加密和混淆过。

5）建议的K值实现

第三方系统可以考虑K值元素包括：用户ID、Vhall直播ID、时间戳（1970-01-01至今的秒数）元素组合后加密后，使用Base64或者hex 匹配成URL可识别编码。K值在第三方系统中持久化或放在Cache中

回调验证时，根据时间戳判断是否在设定时间内有效

验证结束，若认证通过，则从DB或Cache中移除K值

DB或Cache建议有时效性控制，自动失效或定期清理过期数据