VOIP_SDK 控件化集成指南

Android 版

Version 1.2

宁波菊风系统软件有限公司

文档版本	更新人	更新时间
1.0	Upon，Jeff	2017-05-03
1.1	Mofei	2017-07-24
1.2	Mofei	2017-12-08

1. 配置环境

1.1. 新建项目

◆ 在 Android Studio 下面新建项目空的项目

1.2. 添加类库

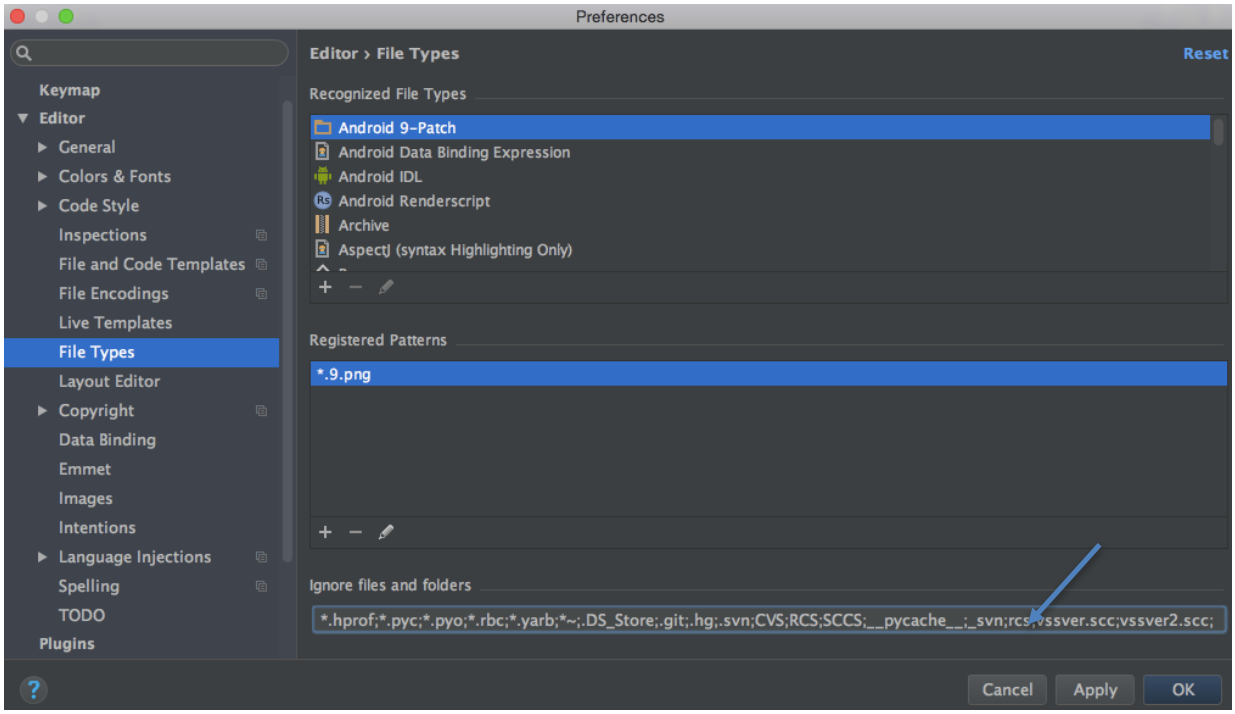
◆ 将所需集成的 aar 文件存放在 libs 文件夹下
◆ 打开 Module 的 build.gradle，添加以下配置

android 里添加

```
repositories {
    flatDir {
        dirs 'libs'
    }
}
```

dependencies 内添加

```
compile(name: 'voiplibs', ext: 'aar') //SDK 库
compile(name: 'rcslogin', ext: 'aar') //登陆模块
compile(name: 'rcscall', ext: 'aar') //通话模块
compile 'com.zhy:okhttputils:2.6.2'
```



注意：一般 Android Studio 默认设置会把 rcs 及一些相关的关键字忽略，导致库引入后不能被找到，这时候就要设置一下，如下图.
打开 Android Studio 的 Preference，搜索 file types，然后将下面的忽略选项做修改，把 rcs 相关全部去掉。

1.3. 添加权限

- AndroidManiFest 添加权限

```
<uses-feature android:name="android.hardware.camera" />
<uses-feature android:name="android.hardware.camera.autofocus" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.BROADCAST_STICKY" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS"
/>
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission
android:name="android.permission.PROCESS_OUTGOING_CALLS" />
<uses-permission android:name="android.permission.BATTERY_STATS" />
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.WRITE_CONTACTS" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.CALL_PHONE" />
<uses-permission
android:name= “android.permission.RECEIVE_BOOT_COMPLETED" />
<uses-permission android:name="android.permission.BLUETOOTH" />
```

1.4. 自定义通话界面

- AndroidManiFest 添加呼叫 Activity,

```
<activity android:name=“APP 包名.自己自定义的 Activity"
// “com.juphoon.rcs.call.module.JusCallActivity“ 可以使用默认的
   android:excludeFromRecents="true"
   android:exported="true"
   android:launchMode="singleTop"
   android:screenOrientation="portrait" />
```

2. 初始化

2.1. 获取 APPKEY

打开网站 http://download.juphoon.com/, 注册帐号并登录.点击创建应用,填写应用信息,勾选需要支持的语音,视频,多方通话的能力,获取 APPKEY

2.2. 清单文件

```
<meta-data
        android:name="com.juphoon.voipapi.APP_KEY"
        android:value="APPKEY" />
```

name:不可修改
value:获取的 APPKEY

2.3. 在线 license 初始化

JusLoginDelegate.initSDKWithOnlineLicense(Context context,
JusLoginDelegate.JusLicenseListener listener);
context: 上下文环境
listener: 监听在线下载 license 结果

2.4. JusLicenseListener 的下载结果回调

public void didDownloadLicense(int result);
result: JusLoginDelegate.RESULT_SUCCESS 下载成功,其他结果则下载失败

2.5. 离线 license 初始化

JusLoginDelegate.initSDKWithOfflineLicense(Context context);
context: 上下文环境
return: JusLoginDelegate.RESULT_SUCCES 初始化成功,其他结果则失败

2.6. 判断 license 是否存在

JusLoginDelegate.existsLicenseInFiles(Context context);
return: true 表示存在在 data 目录下
JusLoginDelegate.existsLicenseInAsset(Context context);
return: true 表示存在在 assets 文件夹下

2.7. 能力查询

主叫发起语音,视频,多方通话时会进行能力查询,成功则发起,失败则发起失败;
该能力需要在注册应用时勾选上.

示例:在 MainActivity里面初始化 SDK 和 APP 的根目录。

```
 private void init() {
      int initSDKResult = JusLoginDelegate.loadLibrary(getApplicationContext()); //加载 SDK 的 SO 库，并做些全局环境准备、
      if (initSDKResult == JusLoginDelegate.RESULT_SUCCESS) {
         if (JusLoginDelegate.existsLicenseInFiles(getBaseContext())) {/*是否已下载 license文件至data/data/包名/files文件夹 */
             initSDKResult=JusLoginDelegate.initSDKWithOfflineLicense(getBaseContext());
         } else if (JusLoginDelegate.existsLicenseInAsset(getBaseContext())) {//assets文件夹中是否存在 license文件(离线 license方案)
             String licensePath = getBaseContext().getFilesDir().getAbsolutePath() + "/license.sign";
            RcsCommonUtils.saveAssetFile(getBaseContext(), "license.sign", licensePath);
            initSDKResult = JusLoginDelegate.initSDKWithOfflineLicense(getBaseContext());
         }
       }
       if (initSDKResult != JusLoginDelegate.RESULT_SUCCESS) {
            String resultTitle = titleWithSDKResult(initSDKResult);
            showAlertDialog("", resultTitle);
            return;
      } else if (!JusLoginDelegate.existsLicenseInFiles(getBaseContext())) {
            showLicenseProgressDialg("", "下载 license 中...");
            JusLoginDelegate.initSDKWithOnlineLicense(getBaseContext(), new
            JusLoginDelegate.JusLicenseListener() {
                  @Override
                  public void didDownloadLicense(int i) {
                       //初始化操作
                  }
            });
       } else {
            //初始化操作
       }
}
```

2.5. 创建项目相关目录

初始化成功后，SDK 会在“/sdcard/应用名”路径下创建文件夹，文件夹内包含如下子文件夹：

1. profiles：所有账号文件夹（每个文件夹内有 provision-v1.xml 为账号详细信息）
2. log：SDK 日志文件夹（crash 为 app 奔溃日志、mme 前缀为媒体日志、mtc 前缀为 SDK 打印日志、若打开登录日志开关，则会创建 login 文件夹存放登录日志）

3. 控件化开发一级接口

1.1. RcsLogin

➢ 监听注册状态

```
RcsLoginManager.addListener(new RcsLoginListener() {
@Override
public void onRegisterStateChanged(int state, int statCode,String statMsg) {
   //statCode 为错误码，statMsg 为错误码中文提示
   switch (state) {
      case RcsLoginManager.MTC_REG_STATE_REGED:
         //登录成功
         break;
      case RcsLoginManager.MTC_REG_STATE_IDLE:

         break;
      case RcsLoginManager.MTC_REG_STATE_REGING:
         //登录中
         break;
      case RcsLoginManager.MTC_REG_STATE_UNREGING:
         //注销中
         break;
      }
      setTitle(loginState);
}
});
```

➢ 发起注册

JusLoginDelegate.login(String username, String password, String authName, String serverIP, String serverRealm, **int** tptType, **int** port, **int** srvType)

username: 用户名
password: 密码
authname: 鉴权名
serverIP: 服务器地址
serverRealm：服务器域名
tptType：传输类型
port：端口
srvType：注册类型（普通 VoIP 用户：EN_MTC_REG_SRV_VOIP，
标准 RCS 用户：EN_MTC_REG_SRV_JOYN_HF，
CMCC RCS 用户: EN_MTC_REG_SRV_CMCC_RCS）

1.2. RcsCall

➢ 初始化通话模块

JusCallDelegate.init(context)

➢ 一对一通话

JusCallDelegate.call(number,isVideo)
number：对方号码
isVideo：是否为视频通话

➢ 多方通话

JusCalDelegate.mutiCall(numbers)
numbers：号码数组

如有额外的需求，如添加通话记录等，则需要 JusCallDelegate.setJusCallListener 来设置监听整个通话过程，详见源码注释。

4. 控件化开发二级接口

1.1. RcsLogin

- 监听注册状态

```
RcsLoginManager.addListener(new RcsLoginListener() {
@Override
public void onRegisterStateChanged(int state, int statCode,String statMsg) {
   //statCode 为错误码，statMsg 为错误码中文提示
   switch (state) {
      case RcsLoginManager.MTC_REG_STATE_REGED:
         //登录成功
         break;
      case RcsLoginManager.MTC_REG_STATE_IDLE:

         break;
      case RcsLoginManager.MTC_REG_STATE_REGING:
         //登录中
         break;
      case RcsLoginManager.MTC_REG_STATE_UNREGING:
         //注销中
```

```
            break;
        }
        setTitle(loginState);
}
});
```

- 发起注册

JusLoginDelegate.login(String username, String password, String authName, String serverIP, String serverRealm, **int** tptType, **int** port, **int** srvType)
username: 用户名
password: 密码
authname: 鉴权名
serverIP: 服务器地址
serverRealm：服务器域名
tptType：传输类型
port：端口
srvType：注册类型（普通 VoIP 用户：EN_MTC_REG_SRV_VOIP，
标准 RCS 用户：EN_MTC_REG_SRV_JOYN_HF，
CMCC RCS 用户：EN_MTC_REG_SRV_CMCC_RCS）

1.2. RcsCall

1.1.1. 一对一音频通话

1.1.1.1. 主叫方

调用 JusCallDelegate.call(String number, boolean isVideo)
number：对方号码
isVideo：false 为音频呼叫.

在调用之后会跳转至 JusCallActivity 并在 Intent 中携带通话信息，详细参照 JusCallActivity 的 handleIntent()处理方法。

➢ **作为主叫时 RcsCallSessionListener 的回调流程**

```
public void callSessionProgressing(RcsCallSession session) {
  //响铃中等待对方接听，UI 提示用户正在振铃，根据 session 判断是否为视频通话来决定是否要显示视频画面
}

public void callSessionStarted(RcsCallSession session){
  //对方接听了通话，这时 UI 应将所有可操作的业务按钮显示，并提示用户通话开始，计时器开始计时，如果是视频通话，则要将对方的视频画面进行显示
}
```

```
public void callSessionTerminated(RcsCallSession session, CallReasonInfo
reasonInfo){
   //通话结束，这时应将 RcsCallSession 释放，并将 JusCallActivity finish
}
public void callSessionStartFailed(RcsCallSession session, CallReasonInfo
reasonInfo){
//表示语音呼叫失败，具体原因可以看回调参数 reasonInfo，然后根据 stateCode 来判
断失败原因以及下一步的 UI 变化（比如直接结束通话或者给用户提示）
}
```

➢ **作为主叫时 JusCallDeleage.JusCallListener 的回调流程**

```
public void jusCallOutgoing(RcsCallSession callSession) {
   // 可添加一对一呼出的历史记录
}

public void jusCallTalking(RcsCallSession callSession) {
   // 可添加一对一已接通的历史记录
}

public void jusCallTermed(RcsCallSession callSession) {
   // 可添加未接通或通话结束的历史记录
   // 从 IncomingDone 直接 Termed 表示未接听
   // 从 IncomingDone 到 Talking 再到 Termed 表示正常通话结束
}
```

1.1.1.2. 被叫方

收到来电后将会根据 JusCallDelegate.jusCallActivityClass()判断是否设置了自定义的 CallActivity，如果返回为 null，则使用默认的 JusCallActivity。

收到来电会唤起 JusCallActivity 通话页面并在 Intent 中携带通话信息，详细参照 JusCallActivity 的 handleIntent()处理方法。

```
int callId = intent.getIntExtra(JusCallDelegate.EXTRA_CALL_ID,
RcsCallDefines.INVALIDID);
RcsCallSession rcsCallSession =
RcsCallManager.getInstance().getCallSession(callId);
```

使用 rcsCallSession.getCallMember 获取的对方信息更新来电界面。

选择接听 rcsCallSession.accept(type)，type 为通话类型（音频、视频）
拒绝接听 rcsCallSession.reject(reason)，reason 为挂断原因

➢ **作为被叫时 RcsCallSessionListener 的回调流程**

```
public void callSessionStarted(RcsCallSession session){
      //这时 UI 应将所有可操作的业务按钮显示，并提示用户通话开始，计时器
      开始计时，通过判断 session 是否为视频通话来选择界面做何处理
}
```

```
public void callSessionStartFailed(RcsCallSession session, CallReasonInfo
reasonInfo){
//通话建立失败，可根据 reasonInfo 的 stateCode 提醒用户失败原因，并在 UI
上显示。
}

public void callSessionTerminated(RcsCallSession session, CallReasonInfo
reasonInfo){
    //拒接或者挂断之后收到该回调
    //通话结束，这时应将 RcsCallSession 释放，并将 JusCallActivity finish
}
```

➢ **作为被叫时 JusCallDelegate.JusCallListener 的回调流程**

```
public boolean jusCallIncomingReceived(RcsCallSession callSession) {
  //返回 true 表示接受该一对一来电邀请，返回 false 表示屏蔽该来电邀请
（如黑名单）
  return true;
}

public void jusCallIncomingDone(RcsCallSession callSession) {
  // 可添加一对一来电的历史记录
}

public void jusCallTalking(RcsCallSession callSession) {
  // 可添加一对一已接通的历史记录
}

public void jusCallTermed(RcsCallSession callSession) {
  // 可添加未接通或通话结束的历史记录
  // 从 IncomingDone 直接 Termed 表示未接听
  // 从 IncomingDone 到 Talking 再到 Termed 表示正常通话结束
}
```

1.1.2. 一对一视频通话

1.1.1.1. 主叫方

调用 JusCallDelegate.call(String number, boolean isVideo)
  number：对方号码
    isVideo：true 为视频呼叫.

在调用之后会跳转至 JusCallActivity 并在 Intent 中携带通话信息，详细参照 JusCallActivity 的 handleIntent()处理方法。

➢ 创建可渲染视图

显示预览或者远端的图像都要创建一个可渲染的视图，通过 RcsCallManager.getInstance().createSurfaceView(getBaseContext()); 即可创建传入参数大小的视图，然后根据需要将其放置显示。

➢ 作为主叫时 RcsCallSessionListener 的回调流程

```
public void callSessionProgressing(RcsCallSession session) {
    //这时候就应该更新 UI 成振铃状态，显示本地预览的图像，详细参考
    JusCallActivity 的 startPreviewView()方法
}

public void callSessionStartFailed(RcsCallSession session, CallReasonInfo
reasonInfo){
    //会话建立失败，提醒用户会话建立失败，释放掉 session 以及预览图像，
    详细参考 JusCallActivity 的 stopVideo 方法，最后销毁 activity。
}

public void callSessionStarted(RcsCallSession session) {
    //会话建立成功，更新界面，设置远端图像，详细参考 JusCallActivity 的
startVideo 方法
}

public void callSessionTerminated(RcsCallSession session, RcsCallReasonInfo
reasonInfo) {
  //会话结束，清除本地和远程图像，详细参考 JusCallActivity 的 stopVideo
方法，并将 JusCallActivity finish
}
```

➢ 收到远端图像

```
public void onVideoCallRenderStarted(RcsCallSession session, SurfaceView
surfaceView, int iSource, String iRender, int iWidth, int iHeight) {
    //这时可将本地的预览图像缩小，将远端图像充满屏幕，详细可以参考
    JusCallActicity 的 shrinkPreview()方法。
}
```

➢ 翻转摄像头

调用 callSession.getVideoCallProvider().switchCamera()接口。
具体参考 JusCallActivity 的 onSwitch()方法。

➢ 作为主叫时 JusCallDelegate.JusCallListener 的回调流程

同一对一语音。

1.1.1.2. 被叫方

收到来电后将会根据 JusCallDelegate.jusCallActivityClass()判断是否设置了自定义的 CallActivity，如果返回为 null，则使用默认的 JusCallActivity。

收到来电会唤起 JusCallActivity 通话页面并在 Intent 中携带通话信息，详细参照 JusCallActivity 的 handleIntent()处理方法。

int callId = intent.getIntExtra(JusCallDelegate.EXTRA_CALL_ID,
RcsCallDefines.INVALIDID);
RcsCallSession rcsCallSession =
RcsCallManager.getInstance().getCallSession(callId);
使用 rcsCallSession.getCallMember 获取的对方信息更新来电界面。

选择接听 rcsCallSession.accept(type)，type 为通话类型（音频、视频）
拒绝接听 rcsCallSession.reject(reason)，reason 为挂断原因

➢ **作为被叫时 RcsCallSessionListener 的回调流程**

除没有 callSessionProgressing 回调外，其余流程同主叫方

1.1.3. 一对一音视频呼叫切换

1.1.3.1. 主叫方

➢ **音频切为视频**

RcsCallSession.update(RcsCallDefines.CALL_TYPE_ONE_VIDEO);
同时应将自己的视频预览图开启。

➢ **视频切为音频**

RcsCallSession.update(RcsCallDefines.CALL_TYPE_ONE_AUDIO);
直接更新为音频通话界面

➢ **音视频切换回调**

```
public void callSessionUpdated(RcsCallSession session){
  //若对方对你的音频转视频邀请做出响应，则会收到该回调
  //可以根据此时 session.isVideo()来更新 UI 为视频通话界面或是音频通话界面
}
```

1.1.3.2. 被叫方

```
public void callSessionUpdateReceived(RcsCallSession callSession) {
//只在主叫音频切换到视频的时候，询问被叫是否同意切换
//拒绝 session.responseUpdate(RcsCallDefines.CALL_TYPE_ONE_VOICE)
```

```
//接受 session.responseUpdate(RcsCallDefines.CALL_TYPE_ONE_VIDEO)
}
```

之后会收到 callSessionUpdated(session)的回调，可以根据此时 session.isVideo 来更新 UI 为视频通话界面或是音频通话界面

1.1.4. 呼叫保持

1.1.1.1. 发起保持

RcsCallSession.hold()

➢ **相关回调**

```
public void callSessionHoldOk(RcsCallSession callSession) {
    //保持成功
    //此时可更新保持通话按钮的状态和停止定时器，来提示用户目前是保持状态。
}

public void callSessionHoldFailed(RcsCallSession callSession ,RcsCallReasonInfo reasonInfo) {
    //保持失败，UI 侧提示用户
}
```

1.1.1.2. 被保持

```
public void callSessionUnHoldOk(RcsCallSession callSession){
    //收到被保持请求
    //此时可更新保持通话按钮的状态和停止定时器，来提示用户当前为保持状态
}
```

1.1.1.3. 解除保持

RcsCallSession.unHold();

➢ **相关回调**

```
public void callSessionUnHoldOk(RcsCallSession callSession) {
    //解除保持成功
    //此时可更新保持通话按钮的状态和打开定时器，来提示当前为保持状态
}

public void callSessionUnHoldFailed(RcsCallSession callSession,RcsCallReasonInfo reasonInfo){
    //解除保持失败，提示用户
}
```

1.1.1.4. 被保持者

```
public void callSessionUnHoldReceived(RcsCallSession callSession){
    //收到解除被保持请求
    //此时可更新保持通话按钮的状态和打开定时器，来提示用户当前为保持状态
}
```

1.1.5. 呼叫等待

```
public void callSessionWaiting(RcsCallSession anotherSession){
    //在已有通话的情况下收到新来电会收到
    //根据 session 获取通话的详细信息，提示用户是否要接受。
    //如果接受，就将前一路通话保持 curSession.hold();
    //接受：anotherSession.accept(参数为音频还是视频);
    //拒绝：anotherSession.Reject(参数为音频还是视频);
    //后续回调流程与普通来电相同
}
```

1.1.6. 录音

1.1.1.1. 开始录音

RcsCallSession.recordStart()

➢ **触发回调**

```
public void callSessionRecordStartOK(RcsCallSession callSession) {
    //录音开始成功
}
public void callSessionRecordStartFailed(RcsCallSession callSession,
RcsCallReasonInfo reasonInfo){
    //录音开始失败
}
```

1.1.1.2. 停止录音

RcsCallSession.recordStop()

➢ **触发回调**

```
public void callSessionRecordStopOK(RcsCallSession callSession) {
    //录音停止成功
}

public void callSessionRecordStopFailed(RcsCallSession callSession,
RcsCallReasonInfo reasonInfo){
    //录音停止失败
}
```

1.1.7. 呼叫后转

➢ **发起呼叫转移**

callSession.transf(peerNumber);//参数为要转接的号码

➢ **触发回调**

```
public void callSessionUpdated(RcsCallSession callSession){
    //转移成功，此时更新当前的通话信息
}

public void callSessionStartFailed(RcsCallSession callSession, RcsCallReasonInfo
reasonInfo){
    //转移失败，提醒用户
}
```

1.1.8. DTMF

callSession.sendDtmf(char c)
参数二直接传空就可以
支持 0~9 以及# *的字符 dtmf 发送

1.1.9. 多方语音通话

1.1.1.1. 主叫方

调用 JusCallDelegate.multiCall(String[] peerNumbers)
peerNumbers：成员号码数组

在调用之后会跳转至 JusCallActivity 并在 Intent 中携带通话信息，详细参照 JusCallActivity 的 handleIntent()处理方法。

➢ **作为主叫时 RcsCallSessionListener 的回调流程**

```
public void callSessionProgressing(RcsCallSession session) {
    //更新 UI 为振铃状态，UI 界面显示所邀请的成员，成员状态为邀请中
}
public void callSessionStarted(RcsCallSession session){
  //多方通话建立，这时 UI 应将所有可操作的业务按钮显示，并提示用户通话开始，
计时器开始计时
}

public void callSessionStartFailed(RcsCallSession session, CallReasonInfo
reasonInfo){
    //表示多方语音失败，具体原因可以看回调参数 reasonInfo，然后根据 stateCode
来判断失败原因以及下一步的 UI 变化（比如直接结束通话或者给用户提示）
```

```
}

public void callSessionTerminated(RcsCallSession session, CallReasonInfo
reasonInfo){
    //通话结束，这时应将 RcsCallSession 释放，并将 JusCallActivity finish
}
```

➢ **作为主叫时 JusCallDelegate.JusCallListener 的回调流程**

```
public void jusConfOutgoing(RcsCallSession callSession) {
  // 可添加多方呼出的历史记录
}

public void jusConfConned(RcsCallSession callSession) {
  // 可添加多方已接通的历史记录
}

public void jusConfDisced(RcsCallSession callSession) {
  // 可添加未接通或通话结束的历史记录
  // 从 IncomingDone 直接 Termed 表示未接听
  // 从 IncomingDone 到 Talking 再到 Termed 表示正常通话结束
}
```

1.1.1.2. 被叫方

收到来电后将会根据 JusCallDelegate.jusCallActivityClass()判断是否设置了自定义的 CallActivity，如果返回为 null，则使用默认的 JusCallActivity。

收到来电会唤起 JusCallActivity 通话页面并在 Intent 中携带通话信息，详细参照 JusCallActivity 的 handleIntent()处理方法。

选择接听 rcsCallSession.accept(type)，type 为 RcsCallDefines.CALL_TYPE_LST_VOICE
拒绝接听 rcsCallSession.reject(reason)，reason 为挂断原因

➢ **作为被叫时 RcsCallSessionListener 的回调流程**

```
public void callSessionStarted(RcsCallSession session){
    //这时 UI 应将所有可操作的业务按钮显示，并提示用户通话开始，计时器
    开始计时，
}

public void callSessionStartFailed(RcsCallSession session, CallReasonInfo
reasonInfo){
//通话建立失败，可根据 reasonInfo 的 stateCode 提醒用户失败原因，并在 UI
上显示。
}

public void callSessionTerminated(RcsCallSession session, CallReasonInfo
reasonInfo){
```

```
    //拒接或者挂断之后收到该回调
    //通话结束，这时应将 RcsCallSession 释放，并将 JusCallActivity finish
}
```

➢ **作为被叫时 JusCallDelegate.JusCallListener 的回调流程**

```
public boolean jusConfIncomingReceived(RcsCallSession callSession) {
  //返回 true 表示接受该多方语音来电邀请，返回 false 表示屏蔽该来电邀请（如黑名单）
  return true;
}

public void jusConfIncomingDone(RcsCallSession callSession) {
  // 可添加多方语音来电的历史记录
}

public void jusConfTalking(RcsCallSession callSession) {
  // 可添加多方语音已接通的历史记录
}

public void jusCallTermed(RcsCallSession callSession) {
  // 可添加未接通或通话结束的历史记录
  // 从 IncomingDone 直接 Termed 表示未接听
  // 从 IncomingDone 到 Talking 再到 Termed 表示正常通话结束
}
```

1.1.1.3. 多方通话成员管理

1.1.1.1.1. 添加成员

```
callSession.inviteParticipants(String[] participants)
```

➢ **RcsCallSessionListener 的相关回调**

```
public void callSessionInviteParticipantsRequestDelivered(RcsCallSession callSession){
    //邀请发送成功，UI 侧提示用户
}
public void callSessionInviteParticipantsRequestFailed(RcsCallSession session, CallReasonInfo reasonInfo){
    //邀请发送失败，UI 侧提示用户
}
```

➢ **JusCallListener 的相关回调**

```
public void jusConfIvtAcpt(RcsCallSession callSession, RcsCallMember callMember)
{
  //多方邀请成员成功
}
```

1.1.1.1.2. 踢出成员

callSession.removeParticipants(String[] participants)

➢ **RcsCallSessionListener 的相关回调**

```
public void callSessionRemoveParticipantsRequestDelivered(RcsCallSession
callSession){
    //提出成员请求发送成功，UI 侧提示用户
}
public void callSessionRemoveParticipantsRequestFailed(RcsCallSession session,
CallReasonInfo reasonInfo){
    //提出成员请求发送失败，UI 侧提示用户
}
```

➢ **JusCallListener 的相关回调**

```
public void jusConfKickAcpt(RcsCallSession callSession, RcsCallMember
callMember) {
  //提出成员成功
}
```

1.1.1.1.3. 成员信息更新

➢ **RcsCallSessionListener 的相关回调**

```
public void callSessionConferenceStateUpdated(RcsCallSession callSession){
    //可从 session 中获得 RcsCallMembers 对象，进行 UI 的更新成员信息
}
```

➢ **JusCallListener 的相关回调**

```
public void jusConfUpdt(RcsCallSession callSession, RcsCallMember callMember) {
   //某个成员更新
}
```

1.1.1.1.4. 参与者

多方通话参与者一般不具备控制通话成员的权利，是否可以收到成员更新的通知也由服务器决定。若平台支持参与者成员更新通知，则参与者可在群成员状态改变时收到callSessionConferenceStateUpdated(RcsCallSession callSession)回调。开发者可以在此回调中进行参与者通话界面的更新。

1.1.10. 一对一通话转多方通话

1.1.1.1. 两路通话合并为一路通话

当前已有一路通话时可以通过挂起当前通话发起新一路的通话，或是挂起当前通话接听另一个通话来实现两路通话。此时可以通过调用任一通话的 callSession.merge()来将两路通话合并为多方通话。

➢ **RcsCallSessionListener 的主席回调**

```
public void callSessionMergeStarted(RcsCallSession session, RcsCallSession
mergeSession) {
    //开始合并两路通话
}
public void callSessionMergeComplete(RcsCallSession session) {
    //合并成功
}
public void callSessionMergeFailed(RcsCallSession session, CallReasonInfo
reasonInfo) {
    //合并失败
}
```

➢ **RcsCallSessionListener 的被叫回调**

```
public void callSessionConferenceExtendReceived(RcsCallSession callSession){
    //作为被叫被扩展为多方通话
}
```

➢ **JusCallListener 的主席方回调**

```
public void jusConfOutgoing(RcsCallSession callSession)
{
  //多方通话开始发起
}
public void jusConfConned(RcsCallSession callSession)
{
  //多方通话创建成功
}
public void jusConfDisced(RcsCallSession callSession)
{
  //多方通话结束
}
```

➢ **JusCallDelegate 的成员方回调**

```
public void jusCallToConf(RcsCallSession callSession)
{
    //一对一通话转为多方通话
}
```

1.1.1.2. 一路通话扩展为多方通话

当前已有一路通话，可以邀请多人直接扩展为多方通话。
[callSession extendToConference:array];

➢ **RcsCallSessionListener 的主席回调流程**

```
public void callSessionConferenceExtendStarted(RcsCallSession callSession)
{
    //扩展开始
}
```

```
public void callSessionConferenceExtendComplete(RcsCallSession callSession)
{
    //扩展成功
}
public void callSessionConferenceExtendFailed(RcsCallSession callSession, CallReasonInfo
reasonInfo
{
    //扩展失败
}
```

➢ **RcsCallDelegate 的被叫回调**

```
public void callSessionConferenceExtendReceived(RcsCallSession session)
{
    //作为被叫被扩展为多方通话
}
```

➢ **JusCallListener 的主席方回调**

同两路通话合并为多方通话

➢ **JusCallListener 的成员方回调**

同两路通话合并为多方通话

4.2.11. 音视频加解密

4.2.11.1 音频加解密

➢ **设置是否对音频加解密**

RtpPacketDelegate.setRtpAudioEnable(boolean enable);
enable : true 为对音频加解密

➢ **获取是否对音频加解密**

RtpPacketDelegate.getRtpAudioEnable();
return : true 对音频进行加解密

4.2.11.2 视频加解密

➢ **设置视频加解密**

RtpPacketDelegate.setRtpVideoEnbale(boolean enable);
enable : true 为对视频加解密

➢ **获取是否对视频加解密**

RtpPacketDelegate.getRtpVideoEnable();
return : true 为对视频加解密

4.2.11.3 监听音视频数据回调

```
RtpPacketDelegate.setListener(new RtpPacketDelegate.RtpPacketListener() {
        //发送数据的回调
        @Override
        public void onSendRtpPacket(int id, String rmtAddr, byte[] pData) {
            // TODO 加密发送数据
        }
        //接收数据的回调
        @Override
        public void onRecvRtpPacket(int id, boolean video, byte[] pData) {
            // TODO 收到数据进行解密
        }
    });
```

4.2.11.4 发送加密后数据

```
RtpPacketDelegate.sendEncrypedPacket(int id, String rmtAddr, byte[] encrypedData)
```

id: 透传,使用回调里的 id
rmtAddr: 透传,使用回调里的 rmtAddr
encrypedData:加密后数据

4.2.11.5 对收到数据解密后调用

```
RtpPacketDelegate.recvDecrypedPacket(int id, boolean video, byte[] decrypedData);
```

id: 透传,使用回调里的 id
video: 透传,使用回调里的 video
decrypedData:解密后数据

5. DM 接口

1.1. 初始化

RcsCpManager.init()
接口说明:
注册 CP 状态的回调

1.2. 进行对应的参数设置

RcsCpManager.setAutoConfig
（context,account,dmAddress,time,port）
参数说明
1. Context 上下文
2. 账号名(不知道情况下可以填空字符串，wifi 下必须要号码)
3. dm 地址
4. 超时时间
5. 端口

1.3. 发起自动配置

RcsCpManager.tryCp();

1.4. 监听的回调状态

RCS_CP_STATE_IDLE = 0;
//自动配置失败，可将当前打开的账号给移除
//RcsLoginManager.removeAccount(mCpAccount);
// mCpAccount 为账号

RCS_CP_STATE_CPING = 1;
//表示正在 cp

RCS_CP_STATE_AUTH_IND = 2;
RCS_CP_STATE_RECV_MSG = 3;
//表示 cp 过程中收到协议条款
//通过 RcsCpManager.getCpRecvMsgTitle()获取协议标题
//通过 RcsCpManager.getCpRecvMsgContent()获取协议内容

//如果接受 调用 RcsCpManager.acceptCp()
//拒绝的话 调用 RcsCpManager.rejectCp();

RCS_CP_STATE_OK = 4;
//表示 cp 成功，调用 RcsLoginManager.login(mCpAccount)登录

RCS_CP_STATE_FAILED = 5;
//表示 cp 不成功。

RCS_CP_STATE_OTP = 6;
//表示验证码正在发送中，需要在 UI 侧把获取验证码的按钮置灰并且开个定时器倒计时
//获取到验证码之后调用 RcsCpManager.promPtOTP(otp)来验证，成功后终端可收到 DM 服务器下发的账号配置文件。（参数 otp 是验证码）

1.5. 获取验证码（示例代码）

```
RcsCmccAuthorize.getInstance().getSmsCode(account, new TokenListener() {
    @Override
    public void onGetTokenComplete(JSONObject jsonObject) {
        dissmissProgressDialogIfNeed();
        int resultCode =
jsonObject.optInt(SsoSdkConstants.VALUES_KEY_RESULT_CODE);
        if (resultCode != AuthnConstants.CLIENT_CODE_SUCCESS) {
            String resultString =
jsonObject.optString(SsoSdkConstants.VALUES_KEY_RESULT_STRING);
            showAlertDialog("获取验证码失败", resultString);
            return;
        }
        Toast.makeText(getBaseContext(), "获取验证码成功",
Toast.LENGTH_SHORT).show();
    }
});
```