ALIBABA CLOUD

阿里云

智能联络中心
开发指南

文档版本：20220328

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

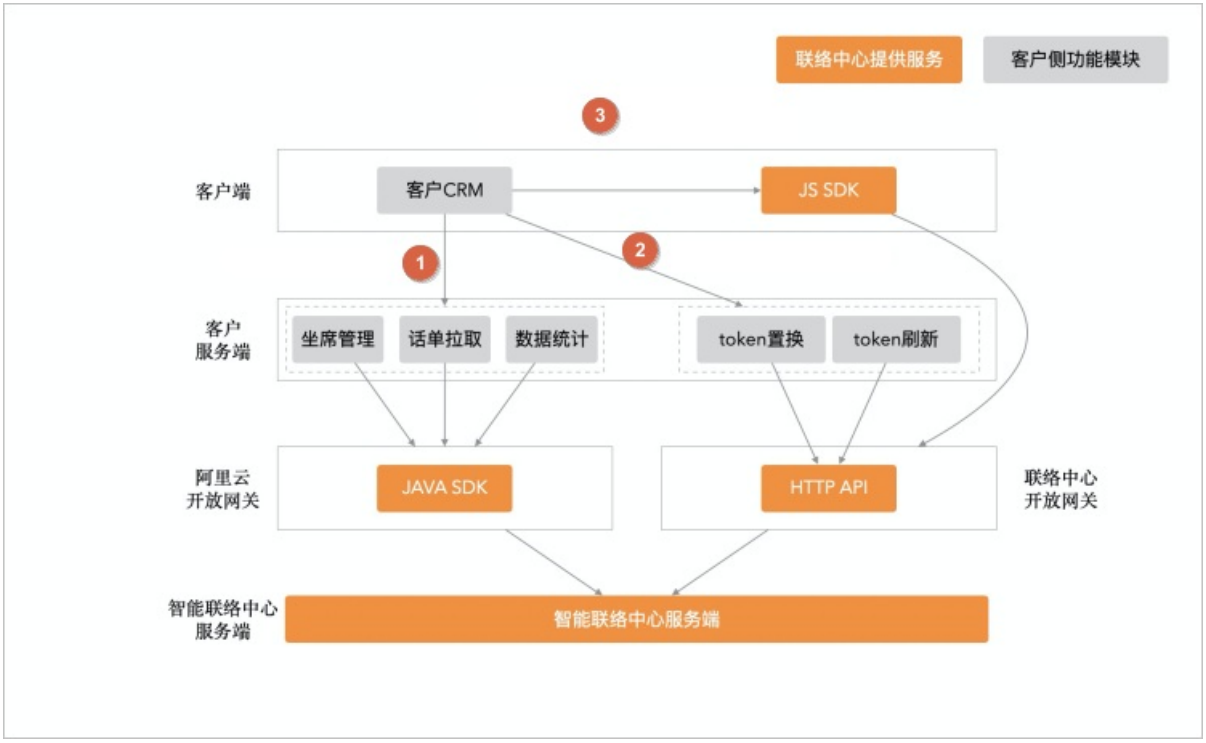
目录

1.客户CRM集成呼叫能力

1.1. 集成简介

智能联络中心JS-SDK，帮助企业快速、简单的把呼叫中心功能集成到企业的业务系统中，实现在CRM、ERP、OA等各个业务场景使用电话功能，进行呼叫中心的外呼和电话接听。当前，智能联络中心提供PC的SDK，支持Chrome内核浏览器。

坐席集成架构



接入流程信息说明：

序号	描述
①	运营管理，更多信息，请参见Java SDK。
②	获取token，更多信息，请参见三方账号授权。
③	SDK初始化，更多信息，请参见联络中心SDK前端接入。

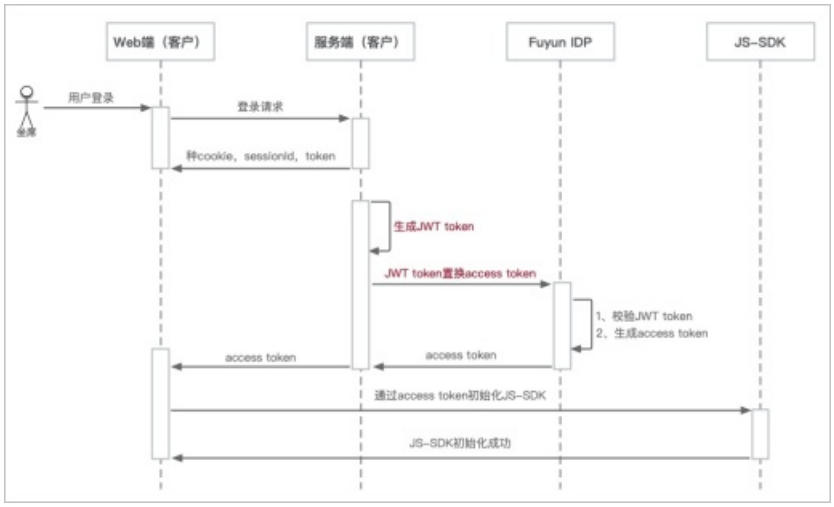
1.2. 三方账号授权

本文为您介绍了三方账号授权的操作流程。

免登流程

1. 第三方系统的用户在己方账号系统登录成功，获取用户登录状态。
2. 第三方系统为用户根据约定规则生成JWT Token。

3. 第三方系统以JWT Token请求免登接口，获取Fuyun门户系统access token。
4. 第三方系统以上述access token请求前端JS-SDK。



> 说明
> - Fuyun：智能联络中心产品开放平台。
> - IDP：Identity provider（身份提供者）。

三方账号免登开发列表

1. 在第三方账号系统登录成功后，生成JWT token（JWT生成规则）。
2. 调用Fuyun IDP免登接口获取Fuyun access token（免登接口）。

JWT token生成规则

payload包含字段

字段	描述	示例
iss	签发者网站域名	"iss":"http://signin.rhino****.com"
exp	过期时间戳	
user_name	JWT token颁发给的用户，此user_name用来映射唯一坐席	
jti	随机uuid	"b774ef13-a5bc-****-8346-042d879efb1a"

JWT token加密

- JWT token的加密方式采用RS256方式加签，public key需要在注册IDP Client的时候提供，private key请您妥善保管。
- private key与public key生成过程可参考下文使用OpenSSL生成密钥对，或者采用三方账号自己的方式生成。

三方账号免登Fuyun门户接口

> **说明** 此处获取的access_token可用于JS_SDK初始化授权。

1、token置换接口

请求URL

- URL：https://signin.rhinokeen.com/oauth/token_exchange。
- 请求类型：POST。

请求HEADER

字段	示例	描述
Authorization	"Authorization: Basic YWxpYmFiYS14aWFvZXI6YmNlMTllZDYtYTFhNC00NzA3LTgwZjAtYTM4OGY3MGUxNWQ3"	授权类型，接口使用http basic authentication认证方式 Authorization= Basic Base64.encode(client_id:client_secret)

请求参数

字段	示例	描述
grant_type	"urn:ietf:params:oauth:grant-type:token-exchange"	授权类型
scope	"fuyun-dev"	访问范围
redirect_url	"http://****.com/callback"	回调地址
subject_token		第三方JWT token
subject_issuer	"http://****.com"	签发者网站域名

成功返回示例

```
{
    "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJodHRwOi8vZGV2ZWxvcGVyLnJoaW5va2Vlbi5jb20iLCJleHAiOjE1OTk3OTczNjMsInVzZXJfbmFtZSI6IumjkuWymiIsImp0aSI6ImI3NzRlZjEzLWE1YmMtNGM3Yi04MzQ2LTA0MmQ4NzllZmIxYSIsImNsaWVudF9pZCI6ImRpbmdkaW5nIiwic2NvcGUiOlsiZnV5dW4tZGV2Il19.jAEBjO9N0D6vjTA0tx8uiAh5dzDgSpo-LVHBmDIMnGVRSvJQWvEc3r6zrp2bRbK7pv83aouhSarHIhtUZ4O-utxP6vMAsGheQstNrXHk79J1XkK-UEXXJXznEyqJvgqjtRw8dTUA2PKSM-1CjRgdRDLvcD7kUGCTAFyYnHwrtdeDw6Vkq1IvkQ3jo4mlTa5pYjF_trXUdAil09IDYk4HNzKHYmuWaAKko42w7abv0GjF7LkNQWM4g6NY3otyge7kxkSHT8K4pOlD7O0zqZlRWfaUXjfGY6H3gdDeRU5N3NyyUXBt2I4VM7pZsW-qwR0XjacdasV5J8UObP37_WzEEw",
    "token_type": "bearer",
    "refresh_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX25hbWUiOiLpo5LlspoiLCJzY29wZSI6WyJmdXl1bi1kZXYiXSwiYXRpIjoiYjc3NGVmMTMtYTViYy00YzdiLTgzNDYtMDQyZDg3OWVmYjFhIiwiaXNzIjoiaHR0cDovL2RldmVsb3Blci5yaGlub2tlZW4uY29tIiwiZXhwIjoxNTk5ODQ0MTU5LCJqdGkiOiJjOTQ5N2ExMi1kZWE5LTRhYmYtOTgxYS0xYjQ1MTRkZWI2MDUiLCJjbGllbnRfaWQiOiJkaW5nZGluZyJ9.NlZ0ay2u0KcxUbgSnacoFw9YwVUJ6cOqwS_vhO92klZroedKzii3vw2rYzbDhTJQshaRgDdAtL2HVrI0bkuWB-T0IuLd_QEnpcDRujqjovxKKJN-uQack5GEd8AwBKYG67IiSGUztIn-RNXO9wGNPM18gQKACc8E34JRbgXnZL6sR6651_pJNT5LpItDe4juDHPgmLpNOHl5Um7wUiE0PpR21d9atJPy5Kt4LL4b4Tnbbtexhofg_vxWVqebNlh-2-nKJyryaKV9otlP0BEh21K-0MFVRotIuAVOwN7goObG6x5KEKFBI8lxI0s9An38TqQzJ1LvzIh77XG7wIDzFg",
    "expires_in": 3599,
    "scope": "fuyun-dev",
    "iss": "http://developer.rhinokeen.com",
    "jti": "b774ef13-a5bc-4c7b-8346-042d879efb1a"
}
```


错误返回示例


```
{
    "error": "invalid_grant",
    "error_description": "Invalid refresh token: 1631958c-df3a-4acd-ac1f-829bcbb6caf01"
}
```


2、token刷新接口

请求信息

- URL：https://signin.rhinokeen.com/oauth/token。
- 请求类型：GET

请求HEADER

字段	示例	描述
Authorization	"Authorization: Basic YWxpYmFiYS14aWFvZXI6YmNlMTllZDYtYTFhNC00NzA3LTgwZjAtYTM4OGY3MGUxNWQ3"	授权类型，接口使用http basic authentication认证方式。 Authorization= Basic Base64.encode(client_id:client_secret)

请求参数

字段	示例	描述
grant_type	"refresh_token"	授权类型

字段	示例	描述
refresh_token	token_exchange获取的refresh_token	refresh_token

成功返回示例


```
{
    "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJodHRwOi8vZGV2ZWxvcGVyLnJoaW5va2Vlbi5jb20iLCJleHAiOjE1OTk3OTczNjMsInVzZXJfbmFtZSI6IumjkuWymiIsImp0aSI6ImI3NzRlZjEzLWE1YmMtNGM3Yi04MzQ2LTA0MmQ4NzllZmIxYSIsImNsaWVudF9pZCI6ImRpbmdkaW5nIiwic2NvcGUiOlsiZnV5dW4tZGV2Il19.jAEBjO9N0D6vjTA0tx8uiAh5dzDgSpo-LVHBmDIMnGVRSvJQWvEc3r6zrp2bRbK7pv83aouhSarHIhtUZ4O-utxP6vMAsGheQstNrXHk79J1XkK-UEXXJXznEyqJvgqjtRw8dTUA2PKSM-1CjRgdRDLvcD7kUGCTAFyYnHwrtdeDw6Vkq1IvkQ3jo4mlTa5pYjF_trXUdAil09IDYk4HNzKHYmuWaAKko42w7abv0GjF7LkNQWM4g6NY3otyge7kxkSHT8K4pOlD7O0zqZlRWfaUXjfGY6H3gdDeRU5N3NyyUXBt2I4VM7pZsW-qwR0XjacdasV5J8UObP37_WzEEw",
    "token_type": "bearer",
    "refresh_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX25hbWUiOiLpo5LlspoiLCJzY29wZSI6WyJmdXl1bi1kZXYiXSwiYXRpIjoiYjc3NGVmMTMtYTViYy00YzdiLTgzNDYtMDQyZDg3OWVmYjFhIiwiaXNzIjoiaHR0cDovL2RldmVsb3Blci5yaGlub2tlZW4uY29tIiwiZXhwIjoxNTk5ODQ0MTU5LCJqdGkiOiJjOTQ5N2ExMi1kZWE5LTRhYmYtOTgxYS0xYjQ1MTRkZWI2MDUiLCJjbGllbnRfaWQiOiJkaW5nZGluZyJ9.NlZ0ay2u0KcxUbgSnacoFw9YwVUJ6cOqwS_vhO92klZroedKzii3vw2rYzbDhTJQshaRgDdAtL2HVrI0bkuWB-T0IuLd_QEnpcDRujqjovxKKJN-uQack5GEd8AwBKYG67IiSGUztIn-RNXO9wGNPM18gQKACc8E34JRbgXnZL6sR6651_pJNT5LpItDe4juDHPgmLpNOHl5Um7wUiE0PpR21d9atJPy5Kt4LL4b4Tnbbtexhofg_vxWVqebNlh-2-nKJyryaKV9otlP0BEh21K-0MFVRotIuAVOwN7goObG6x5KEKFBI8lxI0s9An38TqQzJ1LvzIh77XG7wIDzFg",
    "expires_in": 3599,
    "scope": "fuyun-dev",
    "iss": "http://developer.rhinokeen.com",
    "jti": "b774ef13-a5bc-4c7b-8346-042d879efb1a"
}
```


错误返回示例

```
{
    "error": "invalid_grant",
    "error_description": "Invalid refresh token: 1631958c-df3a-4acd-ac1f-829bcbb6caf01"
}
```

使用OpenSSL生成密钥对

生成RSA密钥

首先进入OpenSSL工具，输入以下命令：

```
OpenSSL> genrsa -out app_private_key.pem   2048  #生成私钥
OpenSSL> pkcs8 -topk8 -inform PEM -in app_private_key.pem -outform PEM -nocrypt -out app_private_key_pkcs8.pem #Java开发者需要将私钥转换成PKCS8格式
OpenSSL> rsa -in app_private_key.pem -pubout -out app_public_key.pem #生成公钥
OpenSSL> exit #退出OpenSSL程序
```

经过以上步骤，开发者可以在当前文件夹中（OpenSSL运行文件夹），看到 app_private_key.pem（开发者RSA私钥，非 Java 语言适用）、app_private_key_pkcs8.pem（pkcs8格式开发者RSA私钥，Java语言适用）和app_public_key.pem（开发者RSA公钥）3个文件。开发者将私钥保留，将公钥提交配置到开放平台，用于验证签名。以下为私钥文件和公钥文件示例。

> **说明** 对于使用Java的开发者，需将生成的pkcs8格式的私钥去除头尾、换行和空格，作为私钥填入代码中，对于.NET和PHP的开发者来说，无需进行pkcs8命令行操作。

标准的私钥文件示例（PHP、.NET使用）

```
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQC+L0rfjLl3neHleNMOsYTW8r0QXZ5RVb2p/vvY3fJNNugvJ7lo4+fdBz+LN4mDxTz4MTOhi5e2yeA
qx+v3nKpNmPzC5LmDjhHZURhwbqFtIpZD51mOfno2c3MDwlrsVi6mTypbNu4uaQzw/TOpwufSLWF7k6p2pLoVmmqJzQ
iD0QIDAQABAoGAakB1risquv9D4zX7hCv9MTFwGyKSfpJOYhkIjwKAik7wrNeeqFEbisqv35FpjGq3Q1oJpGkem4pxa
LVEyZOHONefZ9MGVChT/MNH5b0FJYWl392RZy8KCdq376Vt4gKVlABvaV1DkapL+nLh7LMo/bENudARsxD55IGObMU1
9lkCQQDwHmzWPMHfc3kdY6AqiLrOss+MVIAhQqZOHhDe0aW2gZtwiWeYK1wB/fRxJ5esk1sScOWgzvCN/oGJLhU3kip
HAkEAysNoSdG2oWADxlIt4W9kUiiiqNgimHGMHPwp4JMxupHMTm7D9XtGUIiDijZxunHv3kvktNfWj3Yji0661zHVJw
JBAM8TDf077F4NsVc9AXVs8N0sq3xzqwQD/HPFzfq6hdR8tVY5yRMb4X7+SX4EDPORKKsgnYcur5lk8MUi7r072iUCQ
QC8xQvUne+fcdpRyrR4StJlQvucogwjTKMbYRBDygXkIlTJOIorgudFlrKP/HwJDoY4uQNl8gQJb/1LdrKwIe7FAkBl
0TNtfodGrDXBHwBgtN/t3pyi+sz7OpJdUklKE7zMSBuLd1E3O4JMzvWP9wEE7JDb+brjgK4/cxxUHUTkk592
-----END RSA PRIVATE KEY-----
```

pkcs8处理后的私钥文件示例（Java使用）

```
MIICeAIBADANBgkqhkiG9w0BAQEFAASCAmIwggJeAgEAAoGBAN0yqPkLXlnhM+2H/57aHsYHaHXazr9pFQun907TMvm
bR04wHChVsKVgGUF1hC0FN9hfeYT5v2SXg1WJSg2tSgk7F29SpsF0I36oSLCIszxdu7ClO7c22mxEVuCjmYpJdqb6Xw
eAZzv4Is661jXP4PdrCTHRdVTU5zR9xUByiLSVAgMBAAECgYEAhznORRonHylm9oKaygEsqQGkYdBXbnsOS6busLi6x
A+iovEUdbAVIrTCG9t854z2HAgaISoRUKyztJoOtJfI1wJaQU+XL+U3JIh4jmNx/k5UzJijfvfpT7Cv3ueMtqyAGBJr
kLvXjiS7O5ylaCGuB0Qz711bWGkRrVoosPM3N6ECQQD8hVQUgnHEVHZYtvFqfcoq2g/onPbSqyjdrRu35a7PvgDAZx6
9Mr/XggGNTgT3jJn7+2XmiGkHM1fd1Ob/3uAdAkEA4D7aE3ZgXG/PQqlm3VbE/+4MvNl8xhjqOkByBOY2ZFfWKhlRzi
LEPSSAh16xEJ79WgY9iti+guLRAMravGrs2QJBAOmKWYeaWKNNxiIoF7/4VDgrcpkcSf3uRB44UjFSn8kLnWBUPo6WV
+x1FQBdjqRviZ4NFGIP+KqrJnFHzNgJhVUCQFzCAukMDV4PLfeQJSmna8PFz2UKva8fvTutTryyEYu+PauaX5laDjyQ
bc4RIEMU0Q29CRX3BA8WDYg7YPGRdTkCQQCG+pjU2FB17ZLuKRlKEdtXNV6zQFTmFc1TKhlsDTtCkWs/xwkoCfZKstu
V3Uc5J4BNJDkQOGm38pDRPcUDUh2/
```

公钥文件示例

```
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDQWiDVZ7XYxa4CQsZoB3n7bfxLDkeGKjyQPt2FUtm4TWX9OYrd523
iw6UUqnQ+Evfw88JgRnhyXadp+vnPKP7unormYQAfsM/CxzrfMoVdtwSiGtIJB4pfyRXjA+KL8nIa2hdQy5nLfgPVGZ
N4WidfUY/QpkddCVXnZ4bAUaQjXQIDAQAB
-----END PUBLIC KEY-----
```

附录

JWT token生成代码demo

```
// pom依赖
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-jwt</artifactId>
  <version>1.1.1.RELEASE</version>
</dependency>
/**
 * 生成jwtToken
 * @return
 */
public static String generateToken(String userName, String iss, long expireTime, String pri
vateKeyStr) {
    try {
        privateKeyStr = privateKeyStr.replaceAll("\\s+", "");
        byte[] decodedPrivateKey = Base64.getDecoder().decode(privateKeyStr.getBytes());
        PKCS8EncodedKeySpec spec = new PKCS8EncodedKeySpec(decodedPrivateKey);
        KeyFactory keyFactory = KeyFactory.getInstance("RSA");
        PrivateKey privateKey = keyFactory.generatePrivate(spec);
        RsaSigner signer = new RsaSigner((RSAPrivateKey) privateKey);
        // generate token
        Map<String, Object> payloadMap = new HashMap<>(4);
        payloadMap.put("iss", iss);
        payloadMap.put("jti", UUID.randomUUID().toString());
        payloadMap.put("user_name", userName);
        payloadMap.put("exp", expireTime);
        return JwtHelper.encode(new JSONObject(payloadMap).toJSONString(), signer).getEncod
ed();
    } catch (InvalidKeySpecException | NoSuchAlgorithmException e) {
        e.printStackTrace();
    }
    return null;
}
```

1.3. 热线SDK接入（新版）

本文为您详细介绍呼叫中心SDK前端接入流程。

在线体验

如果您已经有SaaS工作台的账号，您可以在线体验，并且完全可以按照这个Demo的方式进行接入。

接入热线SDK

所有使用 `${version}` 的地方，需要替换为真实的版本号。详细记录信息，请参见热线SDK更新记录。

- **使用非定制UI接入**

- 如果您不需要定制自己的UI，并且您是React体系的项目接入，只需添加以下CDN的资源：

```
<link rel="stylesheet" href="//at.alicdn.com/t/font_1263869_rz6l63j0yrp.css"/>
<link rel="stylesheet" href="//g.alicdn.com/code/lib/antd/4.x.x/antd.min.css"/>
<link rel="stylesheet" href="//g.alicdn.com/hotline-client/hotline-client-sdk/${version
}/hotline-client-ui/index.css"/>
<script src="//g.alicdn.com/code/lib/antd/4.x.x/antd.min.js"></script>
<script src="//g.alicdn.com/hotline-client/hotline-client-sdk/${version}/hotline-client
-ui/index.js"></script>
```

> **注意** 请确保您的项目中已经有React、ReactDOM、**antd的CDN资源**，其中antd的版本建议使用4.x系列。

- 因为构建出的是标准的UMD资源，所以现在可以按照以下方式接入：

```
window.HotlineClientUi.renderClient(
    domContainer, //  比如 document.querySelector('#root')，可以替换为任何你想插入的DOM节点
。
    props,
);
```

- 如果您使用了webpack，还可以使用以下方式接入：

```
// webpack.config.js
module.exports = {
  //...
  externals: {
    HotlineClient: 'HotlineClientUi'
  }
};
// in your app or component
import { HotlineClientUI } from 'HotlineClient';
window.HotlineClientUi.renderClient(
    domContainer, //  比如 document.querySelector('#root')，可以替换为任何你想插入的DOM节点
。
    props,
)
```

上述所有入参的props的内容，请参见HotlineClientUIProps。

- **使用定制UI接入**

  如果您需要定制自己的UI风格，或者需要定制其他UI框架下的组件（比如VUE），可以按照下面的方式来实现：

  i. 引入以下资源：

  ```
  <script src="//g.alicdn.com/hotline-client/hotline-client-sdk/${version}/hotline-clie
  nt-fuyun/index.js"></script>
  ```

  ii. 在自己的代码中创建一个HotlineClient的实例：

  ```
  const { FuYunHotlineClient } = window.HotlineClientFuyun;
  const client = new FuYunHotlineClient(config);
  ```

  iii. 如果您是React体系的项目但是需要定制UI，我们也提供了hooks供您使用：

```
// 需要在cdn中添加 <script src="//g.alicdn.com/hotline-client/hotline-client-sdk/${vers
ion}/hotline-client-ui/index.js"></script>
const { useClient } = window.HotlineClientUi;
const {
  client,           // HotlineClient  实例
  enableState,      // EnableState 使能状态
  agentContext,     // AgentContext
  callContext,      // CallContext
  checkIn,          // 上班签入，可以直接调用
  handleAnswerCall, // 接电话
  handleDial,       // 拨电话
} = useClient(props);
```

其中props的内容请参考HotlineClientUIProps介绍，EnableState内容请参考EnableState。

- **不需要UI接入**

  如果您不需要UI，直接使用API的形式调用，我们也提供了一个扁平化API的版本，只需要引入下面这个CDN地址：

  ```
  <script src="//g.alicdn.com/hotline-client/hotline-client-sdk/${version}/hotline-client-a
  pi/index.js"></script>
  ```

  使用方式：

  ```
  const getHotlineClientSDK = window.HotlineClientApi.default;
  // config: HotlineClientInitConfig
  const SDK = getHotlineClientSDK(config);
  console.log(SDK.client); // 获取client。
  console.log(SDK.agent); // 获取agent。
  onsole.log(SDK.call); // 获取call。
  console.log(SDK.enableState); //获取使能状态。
  // 上班签入。
  SDK.agentCheckin();
  //  下班签出。
  SDK.agentCheckout();
  // 切空闲。
  SDK.agentCheckReady();
  // 切小休。
  SDK.agentCheckRestful();
  //外呼。
  SDK.dial(calleePhoneNumber: string, callerPhoneNumber?: string);
  // 接电话
  SDK.answer();
  // 电话保持。
  SDK.hold();
  // 电话取回。
  SDK.retrieve();
  // 电话转交。
  SDK.deflect(identifier: DeflectionIdentifier);
  // 电话挂断。
  SDK.hangUp();
  // 释放资源。
  SDK.dispose();
  ```

因为状态机是维护在后端的，除了由于没有client、agent或者call导致的报错外（比如没有通话直接调挂断），其余的动作都需要用户根据enableState当前的状态判断是否可以发起，比如发起外呼：

当前如果外呼是否可以发起，API本身不会拦截，应该由状态进行拦截。

SDK接入常见问题

- **怎么订阅事件？**

  热线中基本上每一个动作（比如状态切换、电话状态变更等）都会对应一些事件，为了保证所有后端的事件都完整的传递给上层应用，我们在HotlineClient的实例上提供了完整的事件监听功能，您可以按照这样的方式从client上发起监听：

  ```
  client.on('eventName', (data: EventData) => {})
  ```

  其中eventName可以是HotlineSystemEvents和HotlineCustomEvents中任意的一个key，正如命名所指，HotlineSystemEvents推送的都是后端直接给到的事件信息，而HotlineCustomEvents是SDK系统中自己触发的事件。

  目前所有的EventData都是HotlineSocketEventData。

- **使用过程中会不会有环境问题？**

  会的，目前我们只支持Chrome浏览器64版本以上的环境，我们提供了完整的环境检测工具，包括网络检测、麦克风检测和远端音频播放流程检测。

- **以UI形式接入环境检测工具**

  首先，一定需要以带UI的形式使用，然后加入下面的2个CDN：

  ```
  <link rel="stylesheet" href="//g.alicdn.com/hotline-client/hotline-client-sdk/1.0.1/hotline-client-check-tools/index.css"/>
  <script src="//g.alicdn.com/hotline-client/hotline-client-sdk/1.0.1/hotline-client-check-tools/index.js"></script>
  ```

  现在你只需要将openCheckTool设置为true即可。

- **是否可以用npm的形式接入？**

  考虑到一些依赖需要单独处理的问题，建议使用CDN的形式接入。

术语表

术语详细说明，请参见下表：

术语	说明
Client	即HotlineClient的实例，可以通过create的方式获取，如果直接以UI形式接入，可以通过onClientOnline回调的方式获取。
Agent	即AgentContext的实例，client调用login之后可以拿到，如果以UI形式接入，可以通过onAgentOnline回调的方式获取。

Call	即CallContext的实例，agent调用dial（外呼）或者振铃的时候，可以从几个特殊事件中可以获取到。
	• 系统外呼事件
	外呼事件触发的时候会有callContext：
	`client.on('SystemCallDialOut', (dialOutCallContext) => {console.log(dialOutCallContext)})`
	• 系统振铃事件
	系统振铃事件触发会有callContext：
	`client.on('SystemCallRinging',(ringingCallContext) => {console.log(ringingCallContext) });`
	• 断线重连（用户主动刷新浏览器）事件
	系统断线重连事件触发会有CallContext：
	`// status 当前断线重连的状态。`
	`client.on('AgentCallReconnect',(status, callContext) => {console.log(status, callContext) });`

EnableState	状态使能类表示在当前的坐席状态下是否允许一些操作的使用，比如在一通话务中，是否可以挂断，是否可以转接等等完全由这个“使能类”来定义，类型定义，请参见：HotlineClientEnableState。

类型定义

- **Client相关**

- HotlineClient

```
interface HotlineClient {
  /**
   * 获取此热线客户端的上下文对象。
   */
  readonly context: HotlineClientContext;
  /**
   * 发起登录。
   *
   * {@return 一个坐席上下文对象}。
   */
  login(): Promise<AgentContext>;
  /**
   * 更新token。
   */
  updateToken(token: string): void;
}
```

- FuYunHotlineClient

```
class FuYunHotlineClient implements HotlineClient {
    /**
   * ...
   */
}
```

- HotlineClientContext

```
/**
 * 表示热线 client 的上下文对象
 */
export interface HotlineClientContext {
  /**
   * 获取此 client 上的配置信息。
   */
  readonly config: HotlineClientInitConfig;
  /**
   * 获取此 client 上的热线服务。
   */
  readonly hotlineService: HotlineService;
}
```

- HotlineClientInitConfig

```
/**
 * 提供 {@link HotlineClient} 初始化参数的描述。
 */
export interface HotlineClientInitConfig {
  instanceId: string; // 实例 id
  token: string; // token
}
```

- HotlineClientUIProps

```
/**
 * HotlineClientUIProps 定义。
 */
export type HotlineClientUIProps = {
  /**
   * 初始化 Client 需要的参数。
   */
  config?: HotlineClientInitConfig;
  /**
   * 是否支持拖拽。默认为 false。
   */
  draggable?: boolean;
  /**
   * 拖拽相关的配置。具体可以参考 https://www.n****.com/package/react-draggable
   * 的 DraggableProps
   */
  draggableProps?: Partial<DraggableProps>;
  /**
   * 是否打开检测工具。默认是关闭的。
   */
  openCheckTool?: boolean;
  /**
   * 电话号码是否加密，(脱敏)，默认是 false。
   */
  isPhoneNumberEncrypted?: boolean;
  onClientOnline?: (client: HotlineClient) => void;
  onAgentOnline?: (agent: AgentContext) => void;
  onCallContextChange?: (callContext: CallContext) => void;
  onLoginError?: (error: Error) => void;
};
```

- HotlineClientEnableState

```
export interface HotlineClientEnableState {
  /**
   * 状态。
   */
  status: AgentStatus;
  /**
   * 可签出。
   */
  checkoutEnable: boolean;
  /**
   * 可签入。
   */
  checkinEnable: boolean;
  /**
   * 可空闲。（继续工作）
   */
  readyEnable: boolean;
  /**
   * 可小休。
   */
  breakEnable: boolean;
  /**
   * 可培训。
   */
  trainEnable: boolean;
  /**
   * 可接电话。
   */
  callAnswerEnable: boolean;
  /**
   * 可挂电话。
   */
  callHangupEnable: boolean;
  /**
   * 可拨号。
   */
  callDialEnable: boolean;
  /**
   * 电话可保持。
   */
  callHoldEnable: boolean;
  /**
   * 电话可取回。
   *
   * 保持 + 转接。
   */
  callRetrieveEnable: boolean;
  /**
   * 电话可转接。
   */
  callDeflectEnable: boolean;
}
```

- **Agent相关**

  AgentContext

```
/**
 * 表示坐席的接口定义。
 */
export interface AgentContext extends Disposable {
  /**
   * 坐席发起签入。
   */
  checkIn(): Promise<unknown>;
  /**
   * 坐席发起签出。
   */
  checkOut(): Promise<unknown>;
  /**
   * 坐席发起小休。
   */
  doRest(): Promise<unknown>;
  /**
   * 坐席切到在线。
   */
  checkReady(): Promise<unknown>;
  /**
   * 坐席发起拨号。
   * @param calleePhoneNumber 被叫号码。
   * @param callerPhoneNumber 主叫号码。
   */
  dial(calleePhoneNumber: string, callerPhoneNumber?: string): Promise<CallContext>;
  /**
   * 坐席接电话。
   * @param params 接电话需要的身份信息。
   */
  answer(params?: CallIdentity): Promise<unknow>;
}
```

- **Call相关**
  - CallContext

```
/**
 * 表示通话上下文的结构体。
 */
export interface CallContext extends EventEmitter<HotlineConnectionEvents>, Disposable
{
  /**
   * 获取此通话的标识信息。
   */
  readonly identity: CallIdentity;
  /**
   * 获取此通话的方向。
   */
  readonly callDirection: CallDirection;
  /**
   * 获取此通话的呼叫方身份。
   */
  readonly caller: CallerIdentity;
  /**
   * 获取此通话的被呼叫方身份。
   */
  readonly callee: CallerIdentity;
  /**
   * 挂断此通话。
   */
  hangUp(): Promise<unknown>;
  /**
   * 转接此通话。
   * @param identifier 转接需要的标识符。
   *
   * 根据 DeflectionIdentifier 的 deflectionType 来确认是否需要返回一个可取回的通话。
   */
  deflect(identifier: DeflectionIdentifier): void;
  /**
   * 保持此通话。
   */
  hold(): Promise<unknown>;
  /**
   * 表示取回此通话。
   */
  retrieve(): Promise<unknown>;
}
```

- CallIdentity

```
/**
 * 表示通话身份的结构体。
 */
export interface CallIdentity {
  /**
   * 获取此通话的 acId。
   */
  readonly acId: string;
  /**
   * 获取此通话的连接 Id。
   */
  readonly connId: string;
  /**
   * 获取此通话被保持时的连接 Id。
   */
  readonly holdConnId?: string;
  /**
   * 获取此通话的 jobId。
   */
  readonly jobId: string;
}
```

- CallDirection

```
/**
 * 表示通话方向的枚举，如呼入或呼出。
 */
export enum CallDirection {
  /**
   * 表示呼入。
   */
  Incoming,
  /**
   * 表示呼出。
   */
  Outgoing,
}
```

- **Event事件相关**

- HotlineSystemEvents

```
/**
 * 后端推送的事件。
 */
export interface HotlineSystemEvents<EventData> {
  AgentCheckin: (data: EventData) => void; // 坐席签入
  AgentReady: (data: EventData) => void; // 坐席进入空闲(在线)状态
  AgentCallDialOut: (data: EventData) => void; // 坐席正在外呼事件
  AgentJoinChannel: (data: EventData) => void; // RTC 建立，坐席加入聊天
  AgentLeaveChannel: (data: EventData) => void; // RTC 断开，坐席离开聊天
  AgentCheckout: (data: EventData) => void; // 坐席签出事件
  AgentBreak: (data: EventData) => void; // 坐席小休事件
  AgentAcw: (data: EventData) => void; // 坐席进入话后处理事件
  AgentRinging: (data: EventData) => void; // 坐席振铃
  // AgentCallRelease: (data: EventData) => void; // 通话结束,商业化场景暂时没有
  AgentCallAnswerRequest: (data: EventData) => void; // FIXME: 待确认
  AgentCallInboundEstablish: (data: EventData) => void; // 入呼通道建立事件
  AgentCallOutBoundEstablish: (data: EventData) => void; // 外呼通道建立事件
  AgentCallConferenceWaitAnswer: (data: EventData) => void; // 双步转等待第三方接听事件
  AgentCallHeld: (data: EventData) => void; // 通话保持事件
}
```

- HotlineCustomEvents

```
export interface HotlineCustomEvents<EventData> {
AgentStatusChange: (data: EventData) => void;                    // 坐席状态变更事件
EnableStateChange: (data: HotlineClientEnableState) => void;// 使能类变更事件。
CallVoiceText: (data: EventData) => void;                        // 通话语音文本事件
AgentReconnect: (status: AgentStatus) => void;                   // 坐席断线重连消息。
AgentCallReconnect: (status: AgentStatus, callContext: CallContext) => void; // 通话断线
重连消息。
BeforeCallHold: () => void;                                      // 电话保持动作被触发前触发。

AfterCallHold: () => void;                                       // 电话保持成功后触发。
BeforeCallDeflect: () => void;                                   // 电话转交动作被触发前触发。

BeforeCallHangup: () => void;                                    // 电话挂断动作被触发前触发。

AfterCallHangup: () => void;                                     // 电话挂断成功后触发。
BeforeCallAnswer: () => void;                                    // 电话接起动作被触发前触发。

AfterCallAnswer: () => void;                                     // 电话接起成功后触发。
BeforeCallDial: () => void;                                      // 电话拨号动作被触发前触发。

AfterCallDial: () => void;                                       // 电话拨号成功后触发。
BeforeCallRetrieve: () => void;                                  // 电话取回动作被触发前触发。

AfterCallRetrieve: () => void;                                   // 电话取回成功后触发。
UnHandleEvent: (data: EventData) => void;                        // 异常兜底。
TokenExpired: () => void;                                        // Token过期事件。
SystemCallRinging: (call: CallContext) => void;                  // 带话务上下文的系统振铃事件
。
SystemCallDialOut: (call: CallContext) => void;                  // 带话务上下文的系统外呼事件
。
}
```

- HotlineSocketEventData

```
/**
 * 热线 socket 消息体
 */
export interface HotlineSocketEventData {
  eventName: string;
  buId?: number;
  departmentId?: number;
  enumIconType?: string;
  enumSceneType?: string;
  gmtCreate?: number;
  head: HotlineSocketHeadData;
  isPersistent?: boolean;
  sceneType?: string;
  senderId?: number;
  senderType?: string;
  userId?: number;
  uuid?: string;
  content: string;
}
export interface HotlineSocketHeadData {
  agentBasicCode?: AgentBasicCode;
  agentBasicDesc?: string;
  agentCallCode?: AgentCallCode;
  agentCallDesc?: string;
  aid?: string;
  appName?: string;
  cmd?: string;
  departmentId?: string;
  mid?: string;
  name?: string;
  supportNewFunction?: string;
  time?: string;
  tk?: string;
  xspaceHotline?: string;
  jobId?: string;
  connId?: string;
  holdConnId?: string;
  acid?: string;
  dnis?: string;
  ani?: string;
}
```

- **其他类型说明**

  DeflectionIdentifier

```
export interface DeflectionIdentifier {
  isSingleTransfer?: boolean; // true 是单步转，false 是双步转，默认是 true
  isTransferSkillGroup?: boolean; // 是否转接给技能组。
  isTransferPhone?: boolean; // 是否转接给电话号。
  caller?: string; // 转接到电话的主叫号码。
  callee?: string; // 转接到电话的被叫号码。
  skillGroupId?: string; // 转接的技能组 id
}
```

1.4. 热线SDK接入（旧版）

本文为您详细介绍呼叫中心SDK前端接入流程。

> **注意** 当前版本仅做维护，后续不会继续迭代。如有需求，请参见热线SDK接入（新版）。

引用SDK

1. **SDK cdn assets(umd module)**

   https://g.alicdn.com/xspace/phone/0.5.1/sdk.js

2. **SDK Preview Sandbox**

   https://g.alicdn.com/xspace/phone/0.5.1/index.html#/sample/use-sdk

依赖

1. SIM-API：https://g.alicdn.com/crm/sipml-api/0.0.8/SIPml-api.js。
2. HTTP：https://g.alicdn.com/xspace/phone/0.5.1/http.js。

启用渲染拨号盘会自动加载以上依赖。加载后即可获取SDK实例内容：

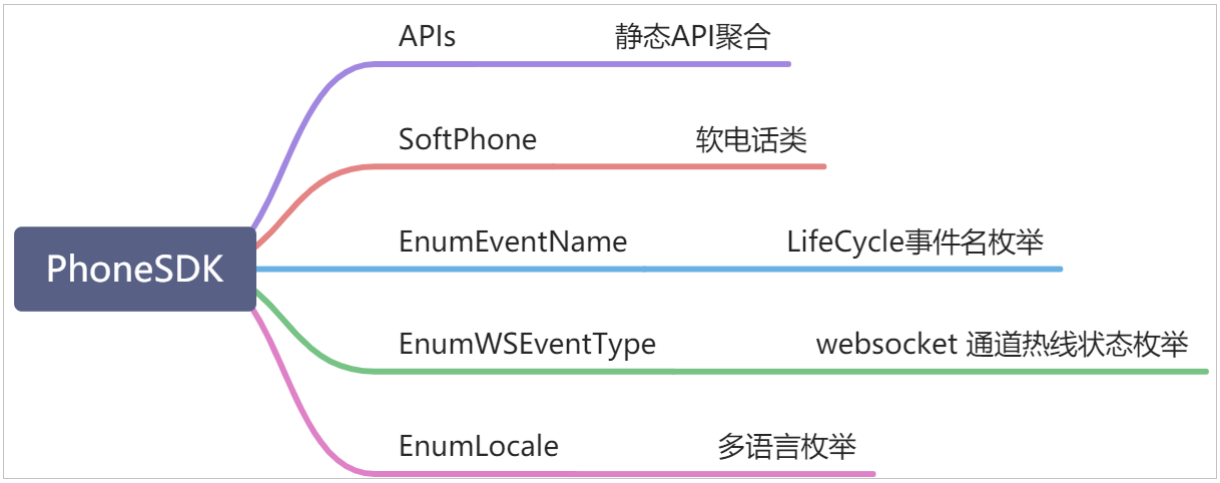
```
const { SoftPhoneSDK } = window
```

前提条件

初始化前需要完成以下准备工作：

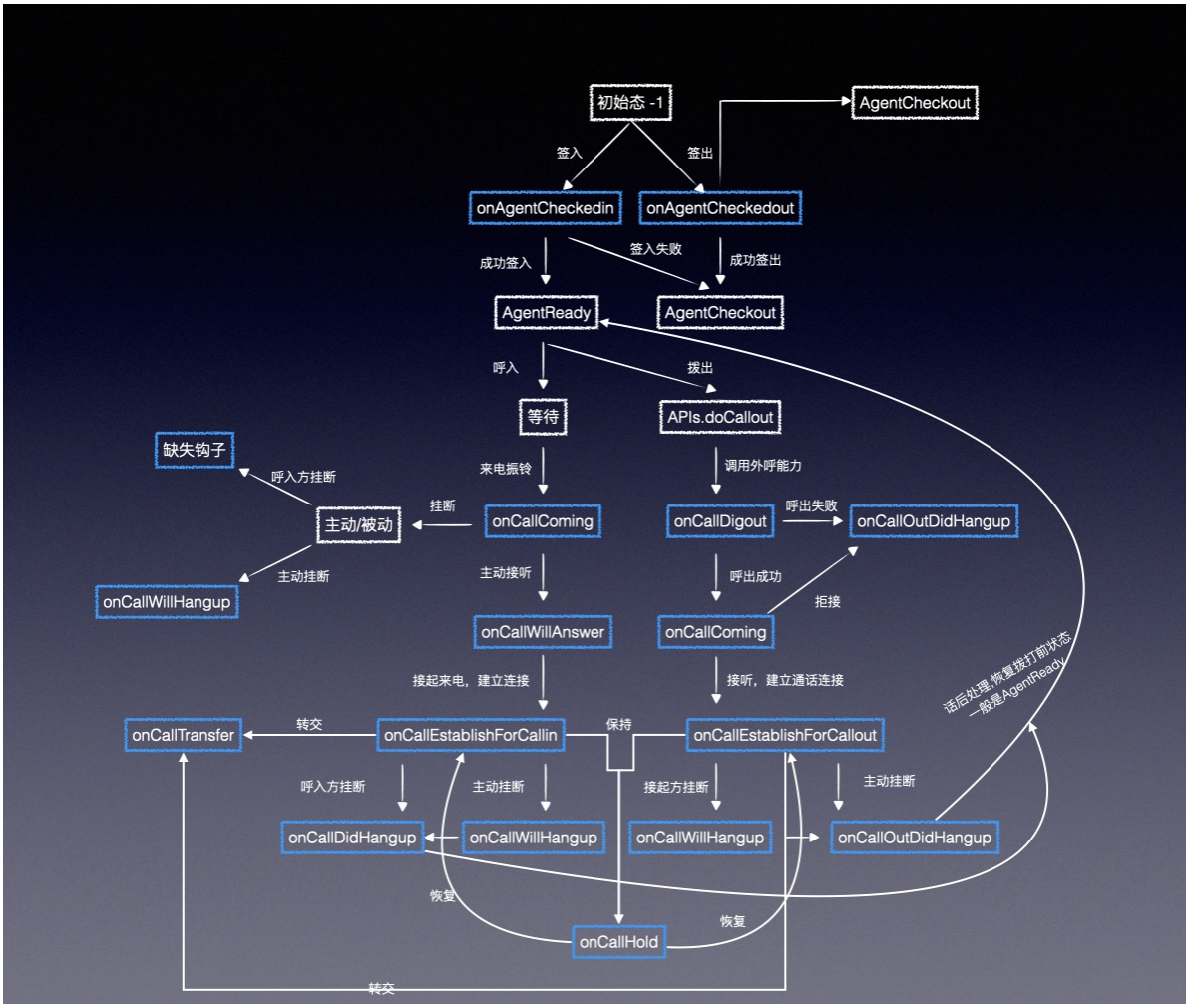
1. 必须使用Chrome浏览器，版本号为58以上。原因是云呼叫中心的通话是通过webRTC技术实现的，目前Chrome浏览器对于webRTC技术的支持是最好的。为了保证您的通话质量及安全性，所以我们做出了这样的要求。
2. 软电话SDK所嵌入的自有业务系统必须使用HTTPS协议。原因是Chrome在47版本之后，禁止HTTP协议获取系统麦克风权限，会造成无法正常通话。
3. 如果您是在 `iframe` 标签内使用软电话SDK，那么需要为 `iframe` 标签增加 `allow="microphone"` 属性，来允许 `iframe` 标签获取系统麦克风权限。

接入手册



状态流转

- 蓝色为流转状态触发的事件hook，请参见事件回调枚举。
- 白色为坐席状态，请参见坐席状态枚举。
- 加载SDK初始坐席状态为-1表示未注册，需要申请InstanceId BearerToken进行签入坐席操作。每个坐席原则意义上按1人/位，可能存在设置同一坐席公用。具体流程如下：



1. **软电话类**

```
const { SoftPhone } = window.SoftPhoneSDK;
const softPhone = SoftPhone.getInstance(); // softphone follow Singleton Pattern
```

通过getInstance方法生成唯一实例。

- **实例API**

API	type	description
setConfig	(config: Partial<SDKConfig> = {})	注入配置，如下所示
register	(eventName: EnumEventName, cb: (eventData: { type: EnumEventName, preConfig: Partial<SDKConfig>, data: any)} => void) => Canceler	注册LifeCycle事件返回其注销方法。
getUIComponent	() => React.ElementType	获取拨号条组件实例，配合APIs.renderComponent 渲染

使用getUIComponent获取的组件必须使用APIs.renderComponent去挂载，这是因为SDK内部集成了React&ReactDOM，使用外部版本挂载会导致版本不通或者双副本的问题。

- **SDKConfig Options**

option	type	defaultvalue	description
InstanceId	string	""	实例ID，可在阿里云控制台使用主账号登录实例列表获取https://ccc.console.aliyun.com/AccInstance。
BearerToken	string	""	bearerToken，通过三方账号授权置换token中的access_token。
env	"online" | "preOnline"	'online'	配置环境。
apiHost	"scsp.aliyuncs.com" | "aiccs.aliyuncs.com" | "api.rhinokeen.com"	'scsp.aliyuncs.com'	API服务器。
Version	string	'2020-07-02'	POP API版本号。 ■ scsp.aliyun.com对应版本为 `'2020-07-02'` ■ aiccs.aliyun.com对应版本为 `'2019-10-15'` ■ api.rhinekeen.com无需版本
locale	EnumLocale	EnumLocale['zh-CN']	国际化。

isPlugin	boolean	true	拨号条插件是否支持拖拽， 默认支持。
needDesensitize	boolean	false	是否对入呼或外呼号码脱敏展示，默认不脱敏。
autoChangeOnLineTime	number	1	自定义话后处理时间，单位为s（秒），为0代表不需要自动切状态，默认为1s自动结束话后处理。
canChangeStatusByHand	boolean	false	是否能够手动切换状态的能力，默认为false，不展示操作栏&不支持手动更新。
enableVoiceToText	Array<'callin' | 'callout'>	[]	启用语音转文本，该能力需要BU配置支持。
enableServiceSummary	boolean	false	启用服务摘要。
disableUI	boolean	false	是否隐藏UI，默认为false，不隐藏。
cdnPath	string	//g.alicdn.com	内置资源默认引用CDN域。

- **Demo**

```
const config =  {
  InstanceId: 'ccc_xp_pre-cn-78v1gnp97002',    // 实例id，可在阿里云控制台实例列表获取
  BearerToken: '',                             // bearerToken，通过【三方账号授权】置换to
ken中的access_token
  env: 'online',                               // online || preOnline,
  apiHost: 'api.rhinokeen.com',                // scsp.aliyuncs.com || aiccs.aliyuncs.
com || api.rhinokeen.com
  Version: '',                                 // Pop Api 版本号，默认为'2020-07-02'. sc
sp.aliyun.com对应版本为'2020-07-02'; aiccs.aliyun.com对应版本为'2019-10-15'; api.rhineke
en.com无需版本
  locale: EnumLocale['zh-CN'],
  isPlugin: true,                              // 插件是否支持拖拽，默认支持
  needDesensitize: false,                      // 是否对入呼/外呼号码脱敏展示,默认不脱敏
  autoChangeOnLineTime: 1,                     // 自定义话后处理时间，单位为s，为0代表不需
要自动切状态，默认为1s自动结束话后处理
  canChangeStatusByHand: false,                // 是否能够手动切换状态
  disableUI: false                             // 是否隐藏UI，默认为false，不隐藏
};
softPhone.setConfig(config);
softPhone.register(EnumEventName.actionError, () => {
  softPhone.setConfig({
    BearerToken: 'newBearerToken'              // token失效后，此处进行事件注册，设置新的t
oken
  });
});
const SoftPhoneUI: React.ElementType<any> = softPhone.getUIComponent();
// if use React Framework, you can render component directly
const ExampleApp: React.FC = () => {
  return (
    <SoftPhoneUI />
  );
}
// if not use React Framework, you can render softphone to specified ElementNode
APIs.renderComponent(SoftPhoneUI, document.querySelector('#phone-container'));
```

2. **LifeCycle事件注册**

   旨在各调度过程中触发的钩子，采用订阅派发的模式进行管理。具体流转状态，请参见流转状态流程图。

   - **事件回调枚举**

```
export enum EnumEventName {
  actionError = 'actionError', // 出错
  actionSuccess = 'actionSuccess', // 拨号盘流程回调
  onAgentCheckedin = 'onAgentCheckedin', // 上班签入成功回调
  onAgentCheckedout = 'onAgentCheckedout', // 下班签出成功回调
  onCallComing = 'onCallComing', // 新来电振铃回调
  onCallDialOut = 'onCallDialOut', // 外拨请求中回调
  onCallWillAnswer = 'onCallWillAnswer', // 呼入电话执行接听动作回调
  onCallEstablishForCallin = 'onCallEstablishForCallin', // 呼入通话建立完成回调（等同于完成接听）：
  onCallEstablishForCallout = 'onCallEstablishForCallout', // 呼出通话建立完成回调
  onCallWillHangup = 'onCallWillHangup', // 执行挂断动作回调
  onCallDidHangup = 'onCallDidHangup', // 呼入电话挂断完成回调
  onCallOutDidHangup = 'onCallOutDidHangup', // 呼出电话挂断完成回调
  onCallTransferOut = 'onCallTransferOut', // 转接回调 0.4.16版本新增
  onCallHold = 'onCallHold', // 通话保持回调 0.4.16版本新增
  onCallRecovery = 'onCallRecovery', // 通话恢复回调 0.4.16版本新增
  'actionError.tokenExpired' = 'actionError.tokenExpired',  // token失效回调
}
```

- **注册监听Demo**

```
// register action error callback
const unRegister = softPhone.register(EnumEventName['actionError.tokenExpired'], (eventData, prevConfig, data) => {
  // here! your can update BearerToken by softPhone.setConfig({BearerToken: 'xxx'});
    const { type, data } = eventData;
  console.log('[SoftPhone actionError.tokenExpired]',type, data);
});
unRegister() // unRegister current actionError.tokenExpired event
```

eventData字段说明：

```
acid: string // 通话id channelId
agentBasicCode: "AgentBusyForCall" // 代理状态Code
agentBasicDesc: "坐席通话忙" // 代理状态描述
agentCallCode: "AgentCallDialOut" // 拨号状态Code
agentCallDesc: "外拨中" // 拨号状态描述
aid: string // 在线id
ani: string
appName: string
cmd: string
connId: string // 话务Id
departmentId: string
dnis: string
eventTime: string // 事件时间
holdConnId: ""
jobId: string  // 小二职位id
mid: string // memberId?
name: string // 会员名称
status: AgentBarStatus // 坐席状态AgentBarStatus
subSessionId: string // 子会话id
supportNewFunction: string
time: string
```

3. **静态APIs列表**

```
const APIs = {
  renderComponent, // 渲染组件到指定节点
  doCallOut, // 执行外呼
  startWork,  // 执行开始上班，已废弃，可由agentCheckin替代
  agentCheckin, // 执行拨号盘签入
  agentCheckout, // 执行拨号盘签出
  getAgentBarStatus, // 获取当前拨号盘状态
  callAnswer,    // 来电接听
  callHangup,            // 电话挂断
  takeWebSocketNotify, // 获取 websocket 通道实例
}
```

- **坐席签入Demo**

  登录操作，状态流转到AgentBarStatus.AgentCheckout。

```
const { APIs } = window.SoftPhoneSDK;
APIs.agentCheckin(); // 是否成功可通过注册onAgentCheckedin事件来监听
```

- **坐席签出Demo**

  登出操作，状态流转到 AgentBarStatus.AgentCheckin。

```
const { APIs } = window.SoftPhoneSDK;
APIs.agentCheckout(); // 是否成功可通过注册onAgentCheckedout事件来监听
```

- **发起呼叫Demo**

  调用外呼能力，外呼成功后执行AgentBarStatus.onCallDialOut，通过返回的eventData.data执行挂断操作。

```
const { APIs } = window.SoftPhoneSDK;
APIs.doCallOut({
  number, // 要呼叫的号码
  onSuccess: this.onSuccess, // 外呼成功接通后的回调，参数为通话信息对象
  param: {
    number, // 要呼叫的号码
    memberId, // 必填，匿名或未知用户可传-1
    memberName, // 必填，用户名称，未知可传 '匿名会员'
    calloutNumber, // 选填，主叫号码，如未填则默认为配置的第一个
    fromSource: 'other_system_out', // 必填，hotlinebs_out || ticket_out || other_syst
em_out （热线||工单||其他系统）
    taskId: this.props.common.taskId, // 如需查动作记录/服务记录时，channelId || caseId
必传其中之一
    channelId: this.props.channelId, // 会话id 可选
    caseId: this.props.common.caseId, // 工单id
  }
})
```

- **挂断电话Demo**

```
APIs.callHangup(connId, jobId, acid, aid);
```

  应用场景：呼入不接起挂断、呼入接起主动挂断、拨出未接/接起主动挂断。

```
// 拨入挂断
softPhone.register(EnumEventName.AgentRinging, (eventData, prevConfig, data) => {
    const { data: { connId, jobId, acid, aid } } = eventData;
  APIs.callHangup(connId, jobId, acid, aid);
});
// 拨出挂断
softPhone.register(EnumEventName.AgentCallDialOut, (eventData, prevConfig, data) => {
    const { data: { connId, jobId, acid, aid } } = eventData;
  APIs.callHangup(connId, jobId, acid, aid);
});
```

- **接听来电Demo**

```
softPhone.register(EnumEventName.AgentRinging, (eventData, prevConfig, data) => {
  // 参数可从 AgentRinging 的事件回调参数里取
    const { data: { connId, jobId, acid, aid } } = eventData;
  APIs.callAnswer(connId, jobId, acid, aid);
});
```

- **获取坐席状态Demo**

回坐席状态，对应值对应的状态如下，初始为-1未注册。

```
const { APIs } = window.SoftPhoneSDK;
const status = await APIs.getAgentBarStatus();
enum AgentBarStatus {
  AgentCheckout = 1, // 签出
  AgentCheckin = 2, // 签入
  AgentReady = 3, // 空闲
  AgentBreak = 4, // 小休
  AgentCallRelease = 5, // 通话结束（挂断的时候）
  AgentAcw = 6, // 话后处理
  AgentRinging = 7, // 振铃
  AgentCallAnswerRequest = 8, // 接听请求中
  AgentCallDialOut = 9, // 拨号请求中
  AgentCallInboundEstablish = 10, // 呼入通话
  AgentCallOutBoundEstablish = 11, // 呼出通话
  AgentCallInternalBoundEstablish = 12, // 内部通话b打给c
  AgentCallConsultEstablish = 13, // 被动求助通话建立
  AgentCallHeld = 14, // 通话保持
  AgentCallConferenceWaitAnswer = 16, // 发起三方
  AgentCallThirdConsult = 17, // 三方通话中
  AgentThirdRetrieveRequest = 18, // 三方取回中
  AgentCallConference = 19, // 会议  三方通话状态
}
```

- **获取webSocket通道**

当某些场景下我们需要定制能力时（e.g.自定义语音转文本能力），可以直接获取websocket通道，该对象提供了两个方法和一个websocket实例。

```
type Subscribe = (eventName: string, callback: (msgData: any) => void) => void;
type PhoneNotify = {
  listenerContainer: {
    [eventName: string]: Array<(msgObj: any) => void>;
  };
  on: Subscribe;
  off: Subscribe;
  socket: ReconnectingWebSocket;
};
const { APIs } = window.SoftPhoneSDK;
const notify: PhoneNotify = APIs.takeWebSocketNotify();
function listenerHotlineMsg(msgData) {
  // 当通道返回的 data.sceneType === 'hotline-message' 时执行
}
notify.on('hotline-message', listenerHotlineMsg); // 执行订阅
notify.off('hotline-message', listenerHotlineMsg); // 注销订阅
```

当前（0.4.16）已定的 sceneType有：

- `hotline-message` 热线消息，表示通话状态和信息。
- `voice-to-text` 语音转文本，表示推送实时语音同声传译后的文本内容。

4. **多语言**
   - 已支持语言枚举：

```
export enum EnumLocale {
  en = 'en',
  'zh-CN' = 'zh-CN',
}
```

   - 默认语言：

     `zh_CN`

   - 设置语言Demo：

```
const { SoftPhone, EnumLocale } = window.SoftPhoneSDK;
const softPhone = SoftPhone.getInstance();
softPhone.setConfig({
    locale: EnumLocale.en, // default = 'zh_CN'
});
```

5. **热线状态**

   当我们想对热线中状态流转有更高的定制需求时，e.g.

   - 在RTC通道（打开或关闭）时打点。
   - 在通话恢复时同步文本语音信息等。

   可以先获取到websocket实例，然后根据 `msgData.head.agentCallCode` 参照 `EnumWSEventType` 枚举热线状态进行定制逻辑处理。具体枚举：

```
export enum EnumWSEventType {
  // 上班状态
  AgentCheckin = 'AgentCheckin', // 签入
  AgentCheckout = 'AgentCheckout', // 签出
  AgentReady = 'AgentReady', // 空闲
  AgentBreak = 'AgentBreak', // 小休
  AgentAcw = 'AgentAcw', // 话后处理
  // 通话事件监听
  AgentRinging = 'AgentRinging', // 来电
  AgentCallInboundEstablish = 'AgentCallInboundEstablish', // 呼入电话建立
  AgentCallOutBoundEstablish = 'AgentCallOutBoundEstablish', // 呼出通话建立
  AgentCallDialOut = 'AgentCallDialOut', // 拨号呼出
  AgentCallAnswerRequest = 'AgentCallAnswerRequest', // 呼入通话执行接听
  AgentCallRelease = 'AgentCallRelease', // 坐席释放
  AgentCallHeld = 'AgentCallHeld', // 通话保持
  // 双步转
  AgentCallConferenceWaitAnswer = 'AgentCallConferenceWaitAnswer', // 三方求助等待应答
  AgentCallConference = 'AgentCallConference', // 会议中
  AgentCallThirdConsult = 'AgentCallThirdConsult', // 三方求助通话中
  AgentThirdRetrieveRequest = 'AgentThirdRetrieveRequest', // 三方取回请求中
  AgentCallConsultThirdRetrieveReq = 'AgentCallConsultThirdRetrieveReq', // 三方求助三方
取回请求中
  AgentCallConsultEstablish = 'AgentCallConsultEstablish', // 三方求助电话建立
  AgentCallInternalBoundEstablish = 'AgentCallInternalBoundEstablish', // 内部通话中
  // RTC
  AgentJoinChannel = 'AgentJoinChannel', // 创建 RTC 连接
  AgentLeaveChannel = 'AgentLeaveChannel' // 销毁 RTC 连接
}
```

e.g.

```
const { APIs, EnumWSEventType } = window.SoftPhone;
const notify: PhoneNotify = APIs.takeWebSocketNotify;
function listenerHotlineMsg(msgData) {
  // 当通道返回的 data.sceneType === 'hotline-message' 时执行
  if(_get(msgData, 'head.agentCallCode') === EnumWSEventType.AgentLeaveChannel) {
      // dosomething
  }
}
takeWebSocketNotify.on('hotline-message', listenerHotlineMsg); // 执行订阅
```

完整版Demo

```
export function APPControll(props: RouterCommonProps) {
  const [hash, refresh] = useHash();
  const { state: { config } } = useContext(Context); // 配置托管到context
  const standWrap = useRef();
  useRegisterHooks(props.history); // 注册钩子
  useInitalIDPToken(hash); // fetch BearerToken assign to `context state` when hash changed
  useDeepEffect(() => {
    instance.setConfig(config);
    // 检测到过期 regenerator token
    const unRegister = instance.register(EnumEventName.actionError, refresh);
    // 渲染拨号条
```

```
    APIs.renderComponent(instance.getUIComponent(), standWrap.current);
    // 执行签入操作
    APIs.agentCheckin();
    return () => {
      APIs.agentCheckout();
      unRegister();
    };
  }, [config]);
  return (
        <div>
        <App />
      <div ref={standWrap} className="statusbar" />
    </div>
  )
}
// e.g. use ReactHooks Synchronization status to Context
function useRegisterHooks(history: RouterCommonProps['history']) {
  const { state, dispatch } = useContext(Context);
  useEffect(() => {
    softPhone.register(EnumEventName.onAgentCheckedin, e => {
      safeConsole('成功登录', e);
      dispatch({
        type: EnumEventName.onAgentCheckedin,
        payload: _get(e, 'data'),
      });
    });
    softPhone.register(EnumEventName.onAgentCheckedout, e => {
      safeConsole('成功登出', e);
      dispatch({
        type: EnumEventName.onAgentCheckedout,
        payload: _get(e, 'data'),
      });
    });
    softPhone.register(EnumEventName.onCallDialOut, e => {
      safeConsole('外拨请求中', e);
      // ...
    });
    softPhone.register(EnumEventName.onCallComing, e => {
      safeConsole('新来电振铃', e);
      // navigation to callpanel page...
      // 暂不支持呼入方主动挂断的事件，绕一下轮询去侦测是否放开坐席
      ;(async function() {
        while (true) {
          await sleep(1000);
          const status = await APIs.getAgentBarStatus();
          if (status === agmentBarStatus.AgentReady) {
            // 处理呼入方主动挂断逻辑...
            return false;
          }
        }
      })();
    });
    softPhone.register(EnumEventName.onCallWillAnswer, e => {
      safeConsole('呼入电话执行接听动作', e);
```

```
    });
    softPhone.register(EnumEventName.onCallEstablishForCallin, e => {
      safeConsole('已接听来电', e);
    });
    softPhone.register(EnumEventName.onCallEstablishForCallout, e => {
      safeConsole('呼出通话建立完成', e);
    });
    softPhone.register(EnumEventName.onCallWillHangup, e => {
      safeConsole('挂断回调', e);
      // navigation to pre page before callin
    });
    softPhone.register(EnumEventName.onCallDidHangup, e => {
      safeConsole('呼入电话挂断完成', e);
    });
    softPhone.register(EnumEventName.onCallOutDidHangup, e => {
      safeConsole('呼出电话挂断完成', e);
    });
    softPhone.register(EnumEventName.onCallHold, (e) => {
      safeConsole('通话保持', e);
    });
    softPhone.register(EnumEventName.onCallRecovery, () => {
      safeConsole('通话恢复');
    });
    softPhone.register(EnumEventName.onCallTransferOut, () => {
      safeConsole('转交');
    });
    softPhone.register(EnumEventName['actionError.tokenExpired'], e => {
      safeConsole('token失效', e);
      // refresh logic...
    });
    return () => /* hanlder cancel events... */
  }, []);
}
```

版发布记录

0.5.0

语音转文本提供呼入呼出可选自开。

0.4.17

- 优化代码逻辑，移除非必要打印信息（sipml除外）
- 新增抛出三个钩子（保持、恢复、转交）

0.4.16

- 新增语音转文本插件。
- 新增服务摘要插件。
- 新增对外暴露websocket以支持定制功能。
- 在线Demo更新。

0.4.15

新增本地化归属

0.4.12

- 新增对外的接听和挂断API。
- 新增隐藏UI的配置参数。

0.4.10

- 归属地外化多tab不显示问题修复
- 增加初始化配置参数-autoChangeOnLineTime用来配置话后处理时间（默认为0，不触发）
- 增加初始化配置参数-canChangeStatusByHand用来配置是否隐藏手动操作状态栏的能力（默认为false，不需要隐藏）

0.4.9

- 号码列表支持“号码组外呼”
- 浮层层级提升至9999
- 拨打电话，上下班等操作提供“同步api”
- 增加“获取拨号盘状态”的前端api
- fuyun api token 过期事件修复
- 入呼、外呼归属地外化

0.4.7

- 拨号盘内部事件通信优化，弃用 `msg-bus`
- iconfont更新

0.4.6

Demo页增加 `cdnPath` 的可配置化

0.4.5

- 增加对后端接口外呼的支持
- 支持在已加载SIPml-api的环境下依旧能够正常运行
- 修改默认外呼来源为 `other_system_out`

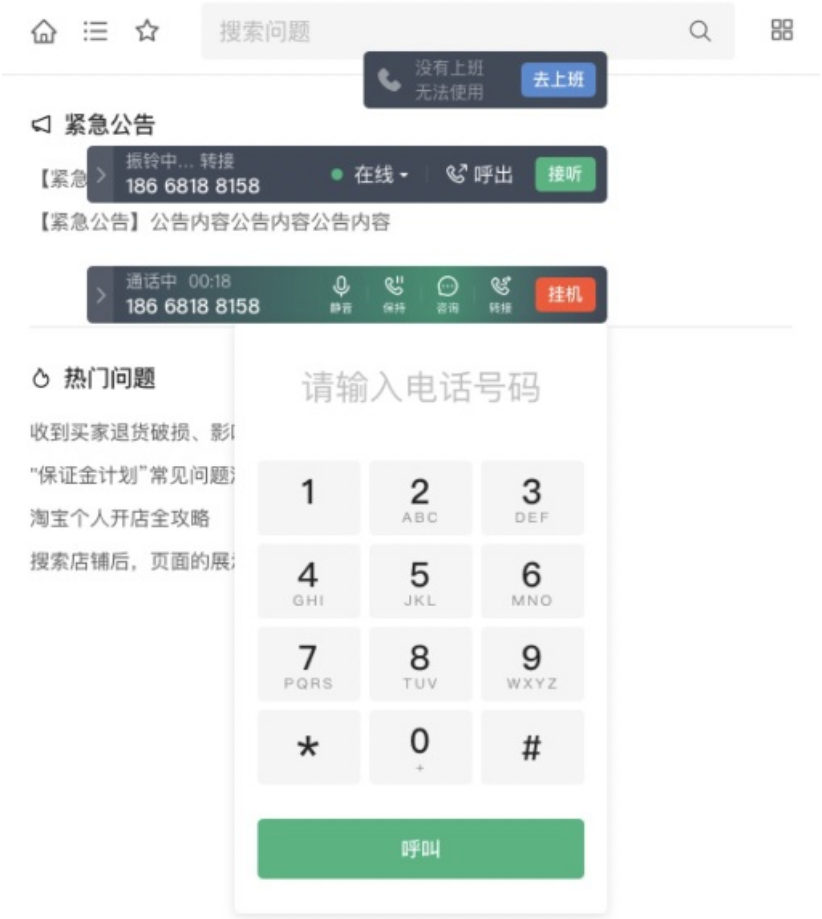
0.4.4

- 增加双步转UI调整
- 测试环境变量修改

0.4.3

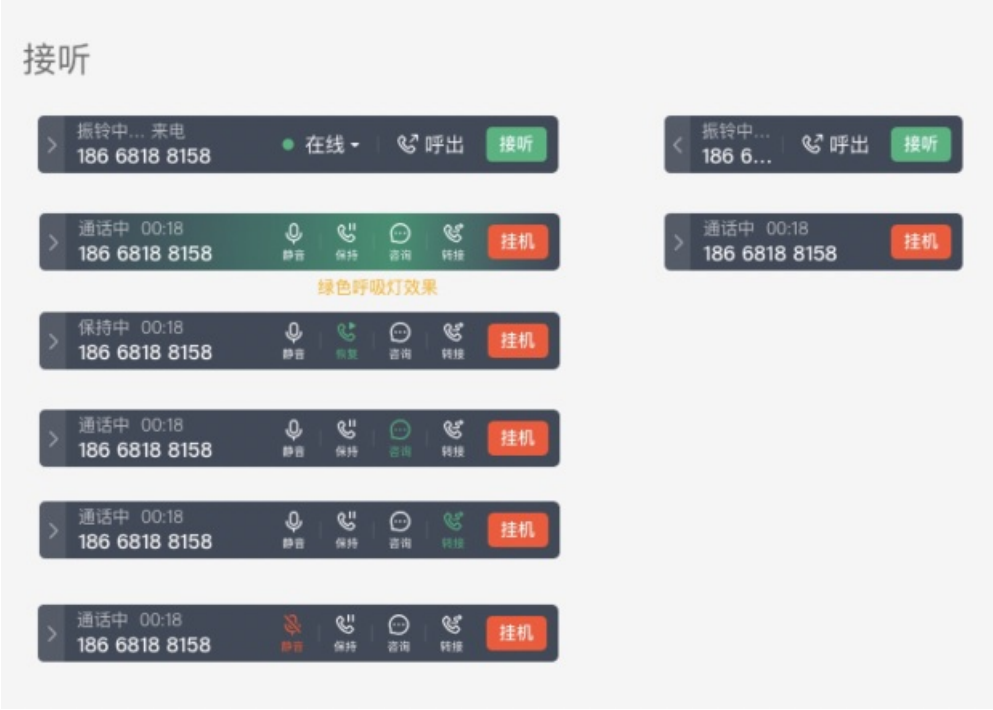
- 交互体验优化
- 增加系统外呼透露
- 增加是否脱敏展现号码的配置项
- 增加是否可拖动的配置项
- Fuyun和Pop空入参优化
- 转交逻辑优化，加入咨询和会议逻辑（暂时隐藏）
- 部分已知bug修复

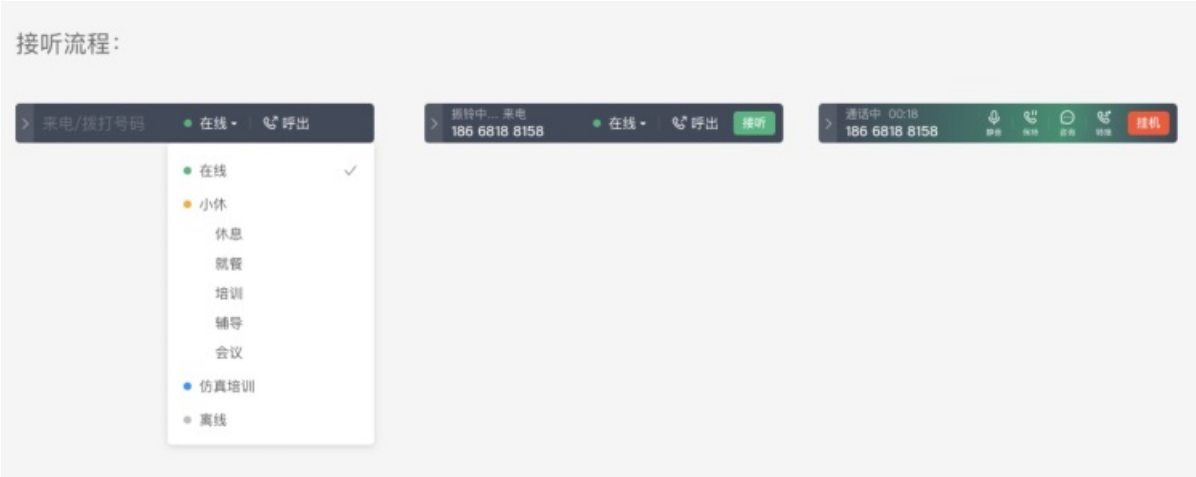
附：运行正常时，图例


搜索问题
没有上班
无法使用
去上班
紧急公告
振铃中... 转接
186 6818 8158
在线
呼出
接听
【紧急公告】公告内容公告内容公告内容
通话中 00:18
186 6818 8158
静音
保持
咨询
转接
挂机
热门问题
收到买家退货破损、影
"保证金计划"常见问题
淘宝个人开店全攻略
搜索店铺后，页面的展
请输入电话号码
1
2 ABC
3 DEF
4 GHI
5 JKL
6 MNO
7 PQRS
8 TUV
9 WXYZ
*
0 +
#
呼叫


未接通
来电/拨打号码
在线
呼出
振铃中... 转接
186 6818 8158
在线
接听
振铃中... 转接
186 6818 8158
在线
呼出
接听
振铃中... 转接
186 6818 8158
在线
呼出
接听


仅外呼
来电/拨打号码
在线
接听
请输入电话号码
1
2 ABC
3 DEF
4 GHI
5 JKL
6 MNO
7 PQRS
8 TUV
9 WXYZ
*
0 +
#
呼出号码：18618618158
呼叫


接听
振铃中... 来电
186 6818 8158
在线
呼出
接听
振铃中...
186 6...
呼出
接听
通话中 00:18
186 6818 8158
挂机
通话中 00:18
186 6818 8158
挂机
绿色呼吸灯效果
保持中 00:18
186 6818 8158
挂机
通话中 00:18
186 6818 8158
挂机
通话中 00:18
186 6818 8158
挂机
通话中 00:18
186 6818 8158
挂机



1.5. 热线SDK更新记录

版本号 1.0.5

日期 2021-06-30

更新说明:

1. ws连接失败新增渐进式重试机制。
2. ws连接失败时使用新的签名。

版本号 1.0.4

日期 2021-05-25

更新说明:

1. 修复默认UI上选择了外呼号码后，通过 API 调用不生效的问题。
2. 替换拖拽组件，HotlineClientUIProps增加draggaleProps，支持自定义拖拽相关的内容，比如可以通过bounds来控制可拖拽的范围，支持所有的react-draggable props定义，可以按需设置。

版本号 1.0.3

日期 2021-05-18

更新说明:

增加websocket被动断开后的重连机制，默认重试5次，5次失败后触发事件 `UnHandleEvent` ，在默认的UI中会自动提示。

版本号 1.0.2

日期 2021-04-22

更新说明：

紧急bug修复，传入onAgentOnline的时候死循环的问题。

版本号 1.0.1

日期 2021-04-22

1. **更新1说明：**
   i. UI组件优化Token过期的提示。
   ii. HotlineClient增加更新Token 的方法。

   使用Demo：

```
client.on('TokenExpired', () => {
    client.updateToken(newToken);
});
```

2. **更新2说明：**

   UI组件增加onLoginError回调函数。

   使用Demo：

```
const onLoginError = (err: Error) => {};
window.HotlineClientUi.renderClient(document.querySelector('#root'), {
  config,
  onLoginError,
});
```

3. **更新3说明：**

   client.removeAllListeners()可以移除所有的监听器。

版本号 1.0.0

日期 2021-04-13

更新说明：首个rc版本发布

2.移动热线集成能力

2.1. 服务端开发指南

本文为您介绍服务端的操作步骤及说明。

1. 调用 CreateAgent创建移动坐席账号。
   - 请求参数

参数名称	参数类型	是否必选	示例值	描述
InstanceId	String	是	agent_12345	AICCS实例ID。可在智能联络中心控制台上获取。
AccountName	String	是	1501234****	账号名称，实例内唯一（可使用坐席手机号、邮箱）。
DisplayName	String	是	测试账号	账号显示名称。
SkillGroupLists	List<Long>	否	123456，145678	坐席所属的技能组列表。

   - 返回数据

参数名称	出参类型	示例值	描述
RequestId	String	"563075BA-4E7A-4546-A8B4-14326822F39C"	请求ID，用于跟踪错误原因。
Success	Boolean	true	接口调用是否成功。
Data	Long	54325	坐席ID。
Code	String	xxxx	错误编码。
Message	String	xxxx	错误描述。

   - 示例程序

```
// 阿里云账号获取ak，sk
DefaultProfile profile = DefaultProfile.getProfile("cn-shanghai", AK, SK);
IAcsClient client = new DefaultAcsClient(profile);
CreateAgentRequest request = new CreateAgentRequest();
request.setInstanceId("agent_88888");
request.setAccountName("150293453****");
request.setDisplayName("测试账号");
try {
    CreateAgentResponse response = client.getAcsResponse(request);
    System.out.println(JSON.toJSONString(response));
} catch (Exception e) {
    // TODO
}
```

2. 调用GetRtcToken获取token。
   - 请求参数

参数名称	参数类型	是否必选	示例值	描述
InstanceId	String	是	agent_12345	AICCS实例ID。可在智能联络中心控制台上获取。
AccountName	String	是	1501234****	账号名称，实例内唯一（可使用坐席手机号、邮箱）。

   - 返回数据

参数名称	出参类型	示例值	描述
RequestId	String	563075BA-4E7A-4546-A8B4-14326822F39C	请求ID，用于跟踪错误原因。
Success	Boolean	true	接口调用是否成功。
Data	Object	54325	坐席ID。

Data.Token	String	{“cleansession”:true,” clientId”:” GID_VOIP@@@ClientId_2000000000000009_100648480015”,” conferenceTopic”:” cs_alicom_voip_conference”,” host”:” mqtt-cn-4590mdhb901.mqtt.aliyuncs.com”,” meetingEventKeepAliveInterval”:0,” phoneTopic”:” alicom_voip_phone”,” port”:0,” reconnectTimeout”:2000,” registerTime”:0,” sdkClientPort”:8883,” serverId”:” GID_VOIP@@@MTEuMTMuMTM2LjExOA==”,” sgwServerTopic”:” alicom_voip_server_pre”,” tlsport”:443,” tokenData”:” abcdef”,” useTLS”:false}	token对象，整个JSON都需要反馈给客户端并且传给SDK。
Data.RtcId	String	2000000000000009	移动端坐席号。
Data.AccountName	String	1501234****	坐席账号。
Code	String	xxxx	错误编码。
Message	String	xxxx	错误描述。

- 示例程序

```
// 阿里云账号获取ak，sk
DefaultProfile profile = DefaultProfile.getProfile("cn-shanghai", AK, SK);
IAcsClient client = new DefaultAcsClient(profile);
CommonRequest request = new CommonRequest();
request.setSysDomain("aiccs.aliyuncs.com");
request.setSysVersion("2019-10-15");
request.setSysAction("GetRtcToken");
request.setSysMethod(MethodType.GET);
request.putQueryParameter("InstanceId", "agent_123234");
request.putQueryParameter("AccountName", "1501912****");
try {
    CommonResponse response = getClient().getCommonResponse(request);
    if (response != null && response.getHttpStatus() == 200 && StringUtils.isNotBlank
(response.getData())) {
        JSONObject data = JSONObject.parseObject(response.getData());
        if (data.getBoolean("Success")) {
            Object result = data.get("Data");
            System.out.println(result);
        }
    }
} catch (ServerException e) {
    log.error(e.getErrCode());
} catch (Exception e) {
    log.error("");
}
```

3. 调用MakeDoubleCall发起呼叫。
   - 请求参数

参数名称	参数类型	是否必选	示例值	描述
InstanceId	String	是	agent_12345	AICCS实例ID。可在智能联络中心控制台上获取。
AccountName	String	是	1501234****	账号名称，实例内唯一（可使用坐席手机号、邮箱）。
ServicerPhone	String	否	150****1234	坐席手机号（若填此参数，则呼叫是发起的是pstn的双呼）。
MemberPhone	String	是	150****1234	用户手机号。
OutboundCallNumber	String	是	0571****1234	外呼主叫号码。
BizData	String	否	{"bizData": "2323"}	扩展字段。

   - 返回数据

参数名称	出参类型	示例值	描述
RequestId	String	563075BA-4E7A-4546-A8B4-14326822F39C	请求ID，用于跟踪错误原因。
Success	Boolean	true	接口调用是否成功。
Data	JSONObject	{"Acid": 68255155365620598}	会话ID。
Code	String	xxxx	错误编码。
Message	String	xxxx	错误描述。

- 示例程序

```
// 阿里云账号获取ak，sk
DefaultProfile profile = DefaultProfile.getProfile("cn-shanghai", AK, SK);
IAcsClient client = new DefaultAcsClient(profile);
CommonRequest request = new CommonRequest();
request.setSysDomain("aiccs.aliyuncs.com");
request.setSysVersion("2019-10-15");
request.setSysAction("MakeDoubleCall");
request.setSysMethod(MethodType.POST);
request.putBodyParameter("InstanceId", INSTANCE_ID);
request.putBodyParameter("AccountName", "150****7032");
request.putBodyParameter("ServicerPhone", "");
request.putBodyParameter("MemberPhone", "10086");
request.putBodyParameter("OutboundCallNumber", "0571****5980");
try {
    CommonResponse response = getClient().getCommonResponse(request);
    if (response != null && response.getHttpStatus() == 200 && StringUtils.isNotBlank
(response.getData())) {
        JSONObject data = JSONObject.parseObject(response.getData());
        if (data.getBoolean("Success")) {
            Object result = data.get("Data");
            System.out.println(result);
        }
    }
} catch (ServerException e) {
    log.error(e.getErrCode());
} catch (Exception e) {
    log.error("");
}
```

2.2. 客户端SDK开发指南

2.2.1. Android

2.2.1.1. Android SDK开发指南

您可通过接入Android SDK快速实现Android端上的音频通话功能（VoIP to PSTN）、及视频通话等功能，依托阿里云的研发资源，为您提供安全、可靠、弹性、低成本的实时通信服务。

前提条件

- 您已经完成注册阿里云账号，并完成企业实名认证。具体操作，请参见和。
- 获取阿里云访问密钥。具体操作，请参见获取AccessKey。
- 通过GetRtcToken获取token。用作为身份标识与服务进行交互，保证安全性。

> **说明** SDK在Token即将失效或需要更新Token时，会通过回调通知接入方，App需自行实现接口获取最新Token并传递给SDK。

SDK下载地址

- Android端SDK下载地址，请参见Android SDK。
- Crash组件下载地址，请参见Crash组件。
- 依赖包下载地址，请参见依赖包。

环境要求

- 设备要求：搭载Android系统的设备。
- 系统要求：语音版Android 5.0(API 21) 及其以上。
- 开发工具要求：Android Studio。

SDK集成

- **导入SDK**

- 方法一：手动导入（推荐）
  a. 请先下载融合通信SDK，解压后放至项目目录，然后根据项目打包环境，自行引入本地依赖。gradle配置参考：

```
implementation fileTree(dir: 'libs', include: ['*.jar', '*.aar'])
```

  b. 在项目目录下的build.gradle文件里面的allprojects里面加入Maven仓库。

```
allprojects {
repositories {
      ...
      maven { url "http://maven.aliyun.com/nexus/content/groups/public/" }
}
```

  c. 如若使用本地依赖，请确保项目中已引入以下依赖：

```
implementation 'com.alibaba:fastjson:1.2.48'
implementation 'org.eclipse.paho:org.eclipse.paho.client.mqttv3:1.1.1'
//如果启用crash组件，增加SDK稳定性请增加以下依赖
implementation 'com.ucweb.wpk:crashsdk:3.2.0.1@aar'
```

  d. 1.7.0版本SDK加入了crash组件，用来保障增加SDK稳定性。如果启用crash组件还需将demo工程lib目录下的 `crashshield-release.aar` 拷贝到自己的lib目录下并完成依赖 `implementation fileTree(dir: 'libs', include: ['*.jar', '*.aar'])`

> **注意** 如若接入方项目构建中因引入其他依赖包从而导致出现类重复或冲突，请联系业务经理。

- 方法二：gradle导入（暂未对外开放）。
  a. 在项目目录下的build.gradle文件里面的allprojects里面加入Maven仓库。

```
allprojects {
repositories {
      ...
      maven { url "http://mvnrepo.alibaba-inc.com/mvn/repository" }
}
```

  b. 配置依赖
     - gradle 3.4及以上版本

```
implementation('com.alicom.rtc:alicomRTC:latest.release@aar'){
     transitive true
 }
or
 api('com.alicom.rtc:alicomRTC:latest.release@aar'){
     transitive true
 }
```

     - gradle 3.4及以下版本

```
compile('com.alicom.rtc:alicomRTC:latest.release@aar'){
     transitive true
 }
```

- so多架构配置
  i. 修改build.gradle文件中artc版本

  由原来的 `implementation 'com.taobao.android:artc_engine:1.9.145.0@aar'` 改为 `implementation 'com.taobao.android:artc_engine:3.0.1.12@aar'` ，或手工导入最新的3.0.1.12版本的包，新版本不包含armeabi架构。

  ii. 修改build.gradle文件中ndk的配置

```
ndk {
            abiFilters  "armeabi-v7a", "arm64-v8a"
  }
```

> **说明** 若需要其他架构so库，请联系业务经理。

混淆配置

主线版本混淆配置：

```
-keep class * implements java.io.Serializable{*;}
-keep class org.artc.webrtc.**{*;}
-keep class org.eclipse.paho.client.mqttv3.**{*;}
-keep class com.taobao.artc.internal.**{*;}
-keep class com.taobao.artc.**{*;}
-keepclasseswithmembernames class * {
   native <methods>;
}
#crash组件
-keep class com.uc.crashsdk.** { *; }
-keep interface com.uc.crashsdk.** { *; }
```

权限配置

```
<uses-permission android:name="android.permission.INTERNET"/>
<!--视频版本必须-->
<uses-permission android:name="android.permission.CAMERA"/>
```

2.2.1.2. SDK使用说明

本文为您介绍了在Android开发时，SDK的使用说明及示例。

创建实例

```
//context建议传入Application，设置成单例对象
AlicomRTC alicomRTC = new AlicomRTC(context);
```

> **说明** AlicomRTC实例一旦需要重新创建新的实例请先destroy旧的实例再创建新的防止出现业务混乱。

参数描述：

参数	类型	说明
context	Context	传ApplicationContext对象

设置参数与回调（可选）

```
//添加一个监听服务状态的回调
alicomRTC.addListener(new ServiceListener() {
        //服务连接成功，当前处于可用状态
        @Override
        public void onServiceAvailable() {}
        //服务连接失败、或者连接断开了，正在销毁资源，当前处于不可操作
        @Override
        public void onServiceUnavailable(int errCode, String errMsg) {}
        //服务已销毁完成，当前处于未连接状态
        @Override
        public void onServiceIdle(){}
        //收到点对点音频来电
        @Override
        public void onReceivingAudioCall(Call call){}
    });
//可选，设置默认的呼叫超时时间
alicomRTC.setDefaultCallTimeout(60);
```

参数描述：

参数	类型	说明
serviceListener	ServiceListener	传监听服务状态的回调。

连接服务

```
/*
 *连接服务,有账号模式需调用initWithRtcId传入融合通信账号,
 *无账号模式则调用initWithCustomId传入自定义账号,
 *以及TokenUpdater的实现用于获取token
 *(Token的具体获取过程由接入方自己实现,
 *由接入方app从接入方服务端获取,
 *而服务端通过pop服务从云通信获取)。
 *SDK每过一段时间就会请求新的token,
 *接入方需要实现方法实时获取id对应最新token。
 *连接成功时会在第2步添加的ServiceListener中回调。
 */
TokenUpdater tokenUpdater =  new TokenUpdater() {
    @Override
    public void updateToken (TokenHandler tokenHandler){
        //get latest token string from network, then
        tokenHandler.setToken(Token.fromJsonString(latestTokenString));
    }
};
//调用此方法进行初始化连接
alicomRTC.initWithRtcId(RtcId, tokenUpdater);
```

参数描述：

参数	类型	说明
rtcId	String	调用AddRtcAccount接口创建获取rtcId。
tokenUpdater	TokenUpdater	更新token回调，获取 `tokenHandler` 来更新token，用 `Token.fromJsonString` 函数来将字符串格式的token转成Token对象。

> **注意** tokenUpdater回调函数的实现，更新的token设置一定要保证是token最新的有效token，尽量避免获取token超时的情况。

开始通话

当 `ServiceListener` 中的 `onServiceAvailable` 回调被唤起时，表示服务已经连接成功，此时可以开始发起各类通话。

- 发起voip2pstn呼叫

- **自有线路**

```
//customLineName为自有线路名称,targetShowNumber为被叫方显示的来电号码，需要从云通信购买可用号码
alicomRTC.builder().pstnCall(customLineName,targetShowNumber, targetPhoneNumber).listener(listener).build().start();
```

参数描述：

参数	类型	说明
customLineName	String	自有线路名称。
targetShowNumber	String	被叫号显，被叫方显示的来电号码。
targetPhoneNumber	String	被叫电话号码。

- **非自有线路**

```
//targetShowNumber为被叫方显示的来电号码，需要从云通信购买可用号码
alicomRTC.builder().pstnCall(customLineName,targetShowNumber, targetPhoneNumber).listener(listener).build().start();
```

参数描述：

参数	类型	说明
targetShowNumber	String	被叫号显，被叫方显示的来电号码。
targetPhoneNumber	String	被叫电话号码。

- 接听电话

```
1.//在ServiceListener中
public void onReceivingAudioCall(Call call) {
    call.start();
}
```

- 挂断/拒绝电话

```
call.stop();
```

- 销毁服务

```
alicomRTC.destroy();
```

- 本地静音，使自己静音，使对方无法听到自己的声音，别人不会知道，与Participant中的mute属性无关

```
call.muteLocalAudiocall();
```

- 使自己取消本地静音状态

```
call.unmuteLocalAudio();
```

- 打开或关闭扬声器

```
call.speakerOn(boolean on);
```

参数描述

参数	类型	说明
on	boolean	true表示打开扬声器，false表示关闭扬声器。

- 开启pstn电话服务端录音

```
pstnCallBuilder.serverRecordEnabled(true)
```

参数描述

参数	类型	说明
on	boolean	true表示开启服务端录音，false表示不开启服务端录音。

- 判断当前是否是本地静音状态

```
call.isLocalAudioMuted()
```

- 获取当前通话设置中扬声器是否打开

```
call.isSpeakerOn()
```

- 设置通话CallListener监听

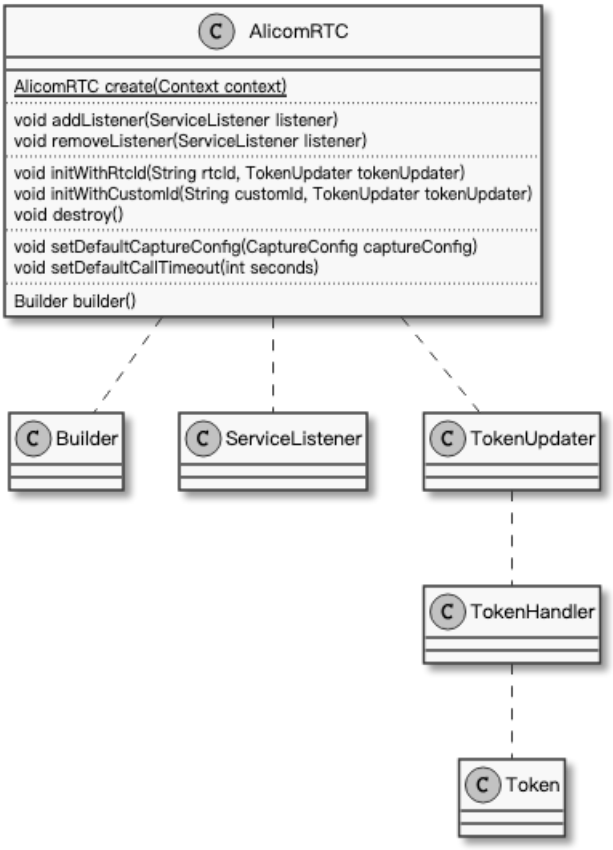
```
call.setCallListener(new AbstractAutoAnswerServiceListener() {
            //需要自动接听的号码列表
            @Override
            public List<String> getAutoAnswerCallerNumbers() {
                return numbers;
            }
            //接收到正常电话(非自动接听号码列表的电话)
            @Override
            public void onReceivingCall(Call call) {
            }
            //自动接听的电话
            @Override
            public void onAutoAnswered(Call call) {
            }
                        //服务连接成功，当前处于可用状态
            @Override
            public void onServiceAvailable() {
            }
                        //服务连接失败、或者连接断开了，正在销毁资源，当前处于不可操作
            @Override
            public void onServiceUnavailable(int errCode, String errMsg) {
            }
                        //服务已销毁完成，当前处于未连接状态
            @Override
            public void onServiceIdle() {
            }
        };
```

> **注意** 安卓9.0系统对App退后台的麦克风做了限制，为防止通话的时候程序退后台引起的通话被静音问题，请在App退后台情况下发送前台通知来防止通话被静音。

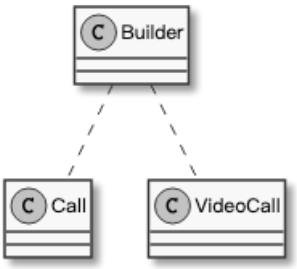
API概述

入口

- 类AlicomRTC为API入口，包含初始化服务、添加生命周期回调、销毁服务、设置全局默认属性、构建通话入口以及一些全局公共方法。
- 类TokenUpdater是接入方需要实现的获取最新token的方法，sdk在需要时会唤起该回调获取最新token，接入方在获取到最新token串后可以调用Token.fromJsonString()方法快捷的获取实例并调用TokenHandler.setToken()方法设置token。

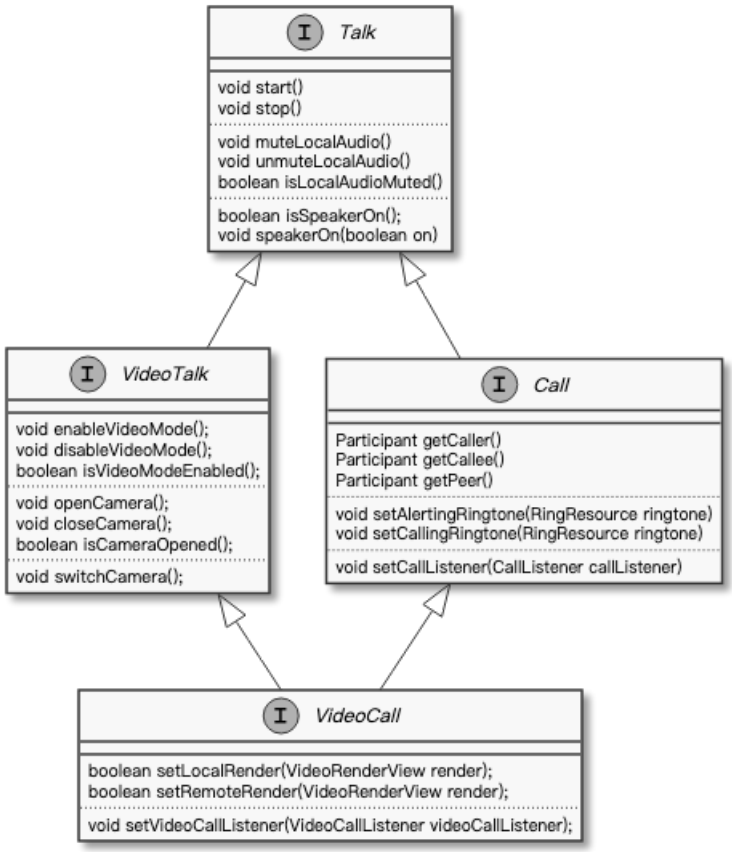


- 调用AlicomRTC.builder方法可以获得构建器用于创建各类通话，并可以设置部分参数。

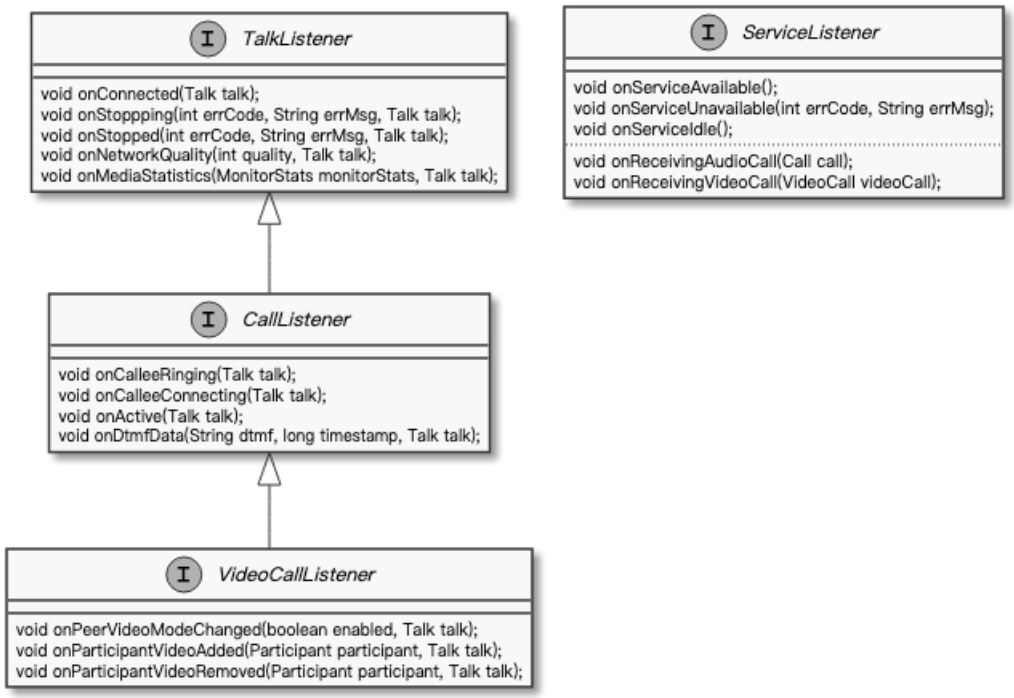


- 通信

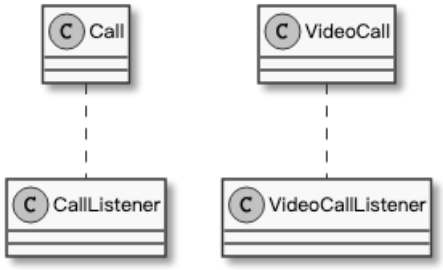
融合通信服务SDK将不同通话功能与类型分拆到了相应的类。比如VideoCall就表示视频点对点通话，包含了视频点对点通话中所有的操作方法，每一个该类的实例表示具体的一次通话，不可重复使用。而基类Talk与VideoTalk分别定义了音频与视频通话的公共方法，没有对应的具体实现



- 监听
  - 接口TalkListener中定义了所有公共回调，具体类型的监听回调接口都继承与该接口。



- 每一个具体的通话类型都对应一个事件监听回调，接入方可以在构建通话时设置监听，也可以调用相应set方法设置监听。



其他API，请参见API Reference。

错误码

错误码	含义
1000100	MQTT网络连接失败
1000101	消息发送失败
1000102/1000103/1000104	TOKEN无效/TOKEN获取失败/TOKEN上传失败（统归TOKEN更新失败）
1000105	MQTT连接断开
1000106	MQTT订阅topic失败
2000000	本地主动挂断
2000050	当前版本不支持视频
2000099	对端主动挂断
2000100	本端主叫时，不要拨打自己
2000101	本端主叫时，拨打的号码为空
2000102	连接媒体服务器超时
2000103	本端被叫时，本地超时未响应，于是本地主动拒绝
2000104	本端主叫时，被叫不在线
2000105	本端被叫时，接听发生错误
2000106	本端主叫时，被叫拒绝
2000107	本端被叫时，主叫取消
2000108	本端主叫时，被叫无响应
2000109	本端主叫时，被叫正在通话中

2000110	本端主叫时，被叫版本过低
2000111/2000112	roomId为空/被踢下线（多方通话的错误码）
2000113	当前服务处于不可用状态
2000116	本地主动销毁服务，以及导致的正在进行的通话中断
2000117	状态不对，当前有别的通话正在进行中，无法进行新的通话
2000118	被叫太多
2000120	自定义消息为空或长度超过1024
2000121	网络异常
2000122	对方接收自定义消息失败
2000130	没有录音权限
2000131	没有相机权限
2000132	主叫呼叫多个被叫时，对于被叫，该通呼叫已被其他设备接听
2000133	主叫呼叫多个被叫时，对于被叫，该通呼叫已被其他设备拒绝
2000134	本地主动销毁服务
2000135	没有麦克风设备
2000136	浏览器不允许使用麦克风
2000137	系统禁用麦克风或者麦克风被占用
3110000	当前账号在别处登录，被踢了，以及导致正在进行的通话中断
3110001	服务异常
3999999	服务器系统异常
3100000	被叫未注册/未登录
3100001	主叫通话异常
3100002	参数不正确
3100003	TOKEN验证失败
3100004	账号错误

3100005	业务停机
3100006	无效号显
3100007	无权操作
3100009	未订购对应产品
3100018/3100019	多方通话的错误码
3100020	被叫不合法
3100021	被叫连接媒体失败
3000000-3300000间别的错误码	服务端其他错误
3900000-4200000	媒体连接失败
8000000	未知错误

常见问题

1. 事件回调会在什么线程回调呢？

   SDK中所有的Listener回调都会在主线程回调到接入方。

2. 每次SDK回调需要更新Token时都要向服务端获取最新的Token吗？

   需要获取最新的Token。

3. 可以实例化多个AlicomRTC吗？

   建议您将AlicomRTC作为单例使用，否则多个实例间会造成Token互斥以及底层资源冲突。

2.2.2. iOS

2.2.2.1. iOS开发

您可通过接入iOS SDK快速实现iOS端上的音频通话等功能，依托阿里云的研发资源，为您提供安全、可靠、弹性、低成本的实时通信服务。

前提条件

- 客户端需要创建智能联络中心账号后才能接入。您可以使用CreateAgent创建移动坐席账号。
- 通过GetRtcToken获取token。作为身份标识与服务进行交互，保证安全性。

  > **说明** SDK在token即将失效或其他需求而需要更新token时，会通过回调通知接入方，App需自行实现接口获取最新token并传递给SDK。

- 设备及系统：
  - 支持所有iOS设备。
  - 支持 iOS 8.x 及以上系统。
  - 支持 armv64、armv7 架构，动态库。

- 开发工具建议使用Xcode 12及以上。

SDK下载地址

- iOS端SDK下载地址，请参见iOS SDK。
- Crash组件下载地址，请参见Crash组件包。

开发流程

1. 安装CocoaPods。

   使用CocoaPods，目前SDK只能收到引入，所依赖的第三方库需要使用CocoaPods，具体安装流程如下：

   i. 确保电脑已经正确安装Ruby环境（Mac电脑自带，需要确认下是否有安装）

   ii. 在Mac终端窗口中输入如下命令：

   ```
   sudo gem install cocoapods
   ```

2. 创建新工程。

   选择**file > new > project**创建新工程。

3. 导入SDK。

   - SDK 库名：AlicomRTCSDK.framework
     a. 下载AlicomRTCSDK.framework。
     b. 下载ArtcSDK.framework。
     c. 下载ArtcMediaEngine.framework。
     d. 在工程上右击选择**Add Files To > 工程名称**，将刚下载的三个SDK添加进来。
     e. 选择对应的target，选择Frameworks，Libraries，and Embedded Content将AlicomRTCSDK.framework和ARTCMediaEngine.framework的Embed设置为Embed & Sign。
   - 在工程target中Linked Frameworks and Libraries添加CoreTelephony.framework，libc++.tbd，libz.tbd系统库。
   - 在工程target中Build Settings的Other Linker Flags增加-ObjC配置。

4. OC兼容Swift。

   在已有的OC工程中新建一个Swift文件，命名为xxx.swift（可任意命名）。在提示框中，选择**Create Bridging Header**建立桥接文件，系统会建立**工程名-Bridging-Header.h**。

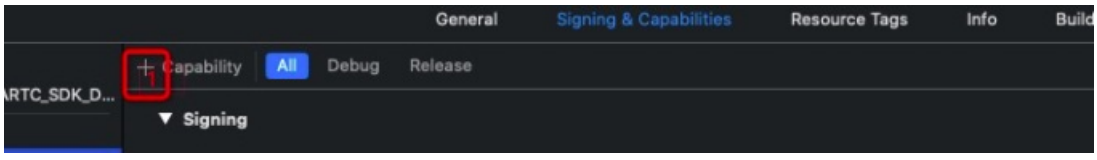
5. Swift兼容OC。

   创建一个oc文件，命名为xxx，继承的父类随便，可以是NSObject。在提示框中，选择**Create Bridging Header**，系统会建立**工程名-Bridging-Header.h**。

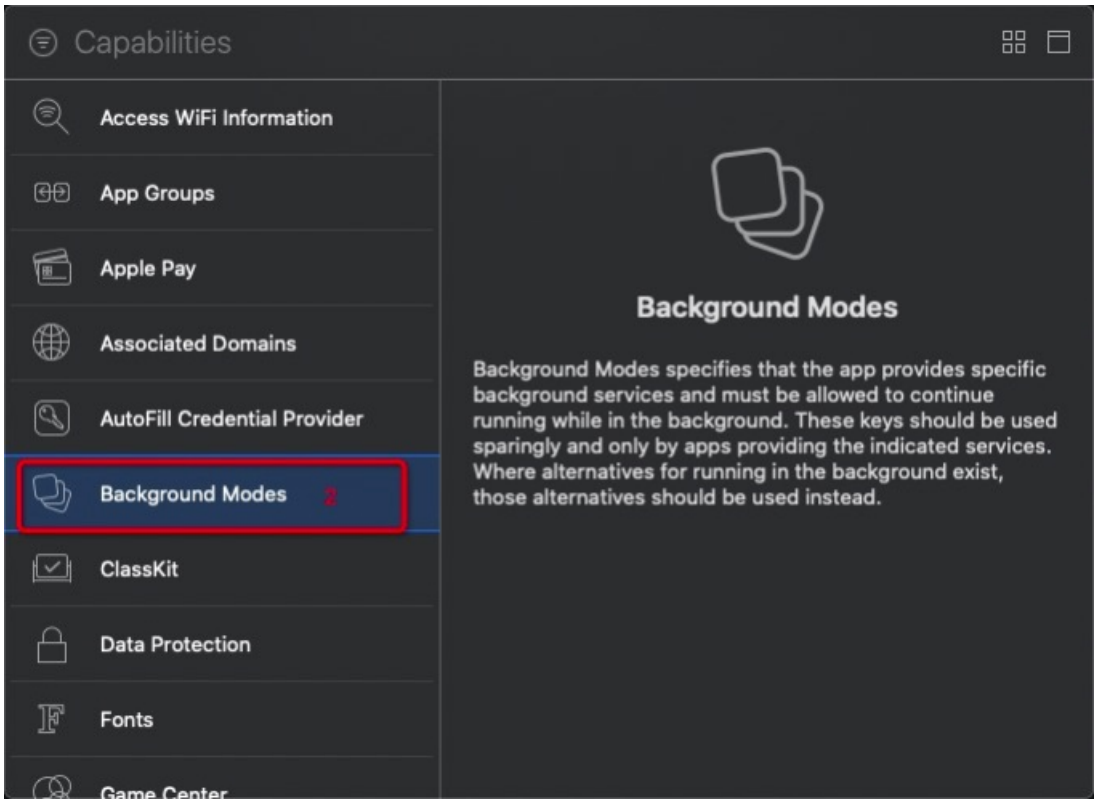
6. 开通后台权限。

   由于iOS系统的限制需要注册相应的后台服务才可以在应用退到后台之后继续使用VoIP服务，以Xcode 11.4.1为例具体开通流程如下：

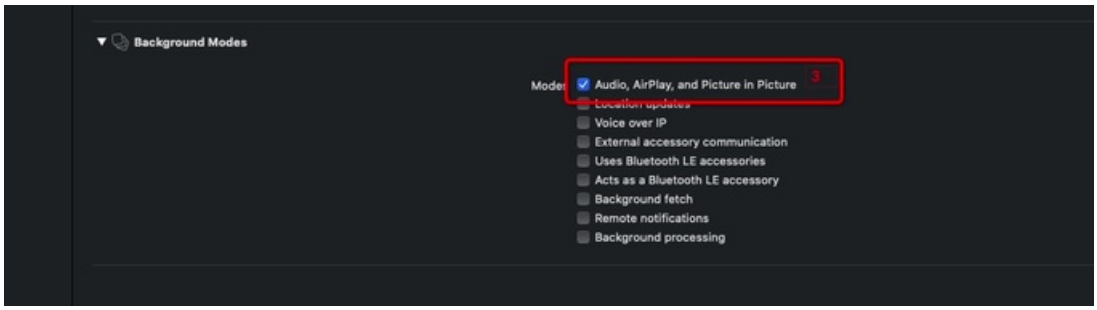
   i. 单击+号。

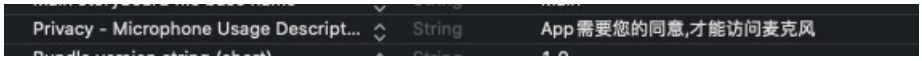
ii. 单击Background Modes。



iii. 选择Audio,AirPlay,and Picture in Picture。



7. 开通麦克风权限Privacy - Microphone Usage Description。



8. 关闭bitcode。

   设置Build Settings > Build Options > Enable Bitcode为NO。

9. 设置Strip Style。

   选择target > Build Settings > Deployment > Strip Style，将需要编译类型全部设置为Debugging Symbols。

2.2.2.2. SDK使用说明

本文为您介绍了在iOS开发时，SDK的使用说明及示例。

创建实例

1. 引入头文件。

```
#import <AlicomRTCSDK/AlicomRTCSDK.h>
```

2. 创建AlicomRTC实例。

```
[[AlicomRTC shareInstance] initWithRtcId:rtcid];
```

3. 设置自动接听的号码列表，如果不设置，默认内置号码为 `703358879` 。

```
/**
 需要自动接听的电话号码列表
 */
- (void)autoAnswerCallerNumbers:(NSArray *)numbers;
```

> **说明**
>
> - rtcid：账号ID由GetRtcToken接口获取，这里需要App端调用服务端接口进行获取app > appserver > aliServer。
> - 初始化之前不需要自己主动获取Token，Token的获取会在初始化完成之后，通过回调告知何时需要获取，只要实现updateToken給SDK设置Token即可。

4. 设置相关代理。

```
[ACToken shareInstance].delegate = self;
[AlicomRTC shareInstance].delegate = self;
```

5. ACTokenDelegate代理回调。

   通知用户更新token。

```
//刷新token使用
- (void)updateToken:(void(^)(ACToken *token))tokenHandler
```

> **说明** 用户在实现该接口的时候可以向服务端获取Token，通过服务端的GetRtcToken接口实现Token获取再返给客户端。

6. AlicomRTCServiceDelegate代理回调。

```
@required
/**
 初始化成功，服务可用
 */
-(void)onServiceAvailable;
/**
 初始化失败，服务不可用
 @param errCode 错误码
 @param errMsg 错误信息
 */
-(void)onServiceUnavailableWithErrorCode:(NSInteger)errCode errorMsg:(NSString *)errMsg
;
@optional
/**
 自定义消息接受回调
 @param sequence 消息序列号
 @param content 消息内容
 @param sourceRtcId 消息来源
 @param timestamp 时间戳
 */
-(void)onCustomMessageReceivedWithSequence:(long long)sequence sourceRtcId:(NSString *)
sourceRtcId content:(NSString *)content timestamp:(long long)timestamp;
/**
 自定义消息发送回调
 @param sequence 消息序列号
 @param isSuccess 发送消息是否成功
 @param errCode 失败的话，有错误码，否则为0
 @param errMsg 失败的话，错误信息。否则为nil
 */
-(void)onCustomMessageSendedWithSequence:(long long)sequence isSuccess:(BOOL)isSuccess
errorCode:(NSInteger)errCode errorMsg:(NSString *)errMsg;
```

> **说明** 其中 `onServiceAvailable` 和 `onServiceUnavailableWithErrorCode:errorMsg:` 是必须实现的接口。

7. 初始初化通话实例。

```
/**
 voip2pstn通话
 @param calleePhoneNumber 被叫电话号码
 @param calleeShowNumber 被叫显示来电号码
 @param customLine 自有线路名称，无则传入nil
 @param extendString 拓展字段（非必要参数）
 @return Call的对象（需判断返回是否为空值）
 */
- (Call *)createPstnCall:(NSString *)calleePhoneNumber andCalleeShowNumber:(NSString *)
calleeShowNumber customLine:(NSString *)customLine withExtendString:(NSString *)extendS
tring;
```

8. 在创建完成通话实例之后需要进行通话状态管理，调用方法如下：

```
-(void)start;
```

9. 停止呼叫。

```
-(void)stop;
```

10. （可选）通话状态管理，需要实现CallDelegate。

```
/**
 被叫来电，振铃彩铃
 @param call Call对象
 */
-(NSData *)ringtoneWithCallee:(Call*)call;
/**
 主叫呼叫中，振铃彩铃
 @param call Call对象
 */
-(NSData *)ringtoneWithCaller:(Call*)call;
/**
 音频电话来电，被叫振铃，并返回振铃数据
 @param call Call对象
 */
-(void)onReceivingAudioCall:(Call*)call;
/**
 视频电话来电，被叫振铃，并返回振铃数据
 @param call VideoCall对象
 */
-(void)onReceivingVideoCall:(Call*)call;
/**
 媒体连接成功了
 @param call Call对象
 */
-(void)onCallConnected:(Call*)call;
/**
 接通对方了，主叫振铃
 @param call Call对象
 */
-(void)onCallRinging:(Call*)call;
/**
 被叫接听了，主叫停止振铃
 @param call Call对象
 */
-(void)onCallAnswered:(Call*)call;
/**
 自动接听了
 @param call Call对象
 */
- (void)onAutoAnswered:(Call *)call;
/**
 通话结束
 @param call Call对象
 @param code code
 @param msg  msg
 */
-(void)onCallStopped:(Call *)call withCode:(NSInteger)code andMsg:(NSString *)msg;
/**
 音频输出模式变更
```

```
 @param call call对象
 @param mode 输出mode
 */
-(void)onPlayModeChanged:(Call*)call withMode:(Audio_Session_Mode)mode;
/**
 音频接收dtmf数据
 @param call call对象
 @param dtmf 数据内容,timestamp 时间戳
 */
-(void)onDtmfData:(Call*)call withDtmf:(NSString *)dtmf withTimestamp:(long long)timest
amp;
/**
 通话中的网络质量
 @param call Call对象，quality，网络质量，0-好，1-中，2-差
 */
-(void)onNetworkQuality:(Call*)call withQuality:(int)quality;
/**
 通话中的指标统计
 @param call Call对象，model StatisticModel 对象
 */
-(void)onMediaStatistics:(Call*)call withModel:(StatisticModel *)model;
/**
 音视频模式变更
 @param call call对象
 @param mode 输出mode，0-视频，1-音频
 */
-(void)onMediaModeChanged:(Call*)call withMode:(int )mode;
```

功能使用

- 本地静音。

```
-(void)mute;
```

- 取消本地静音。

```
-(void)unmute;
```

- 呼叫默认超时时间设置，默认45s，可设置时间区间30s~90s。

```
-(void)setCallTimeout:(int) time;
```

- 发送dtmf，仅在点对点通话中对端为PSTN且通话中才能发送，且只支持 `0-9、*、#` 输入。

```
-(BOOL)sendDtmfData:(NSString *)dtmf;
```

- 切换音频播放路由。

```
-(void)audioSpeakerIsOn:(BOOL)isOn; //isOn，音频播放路由：YES-扬声器播放，NO-听筒播放
```

> **说明** 调用该方法前需要确保通话已经建立。