融合通信Web接口文档

修订历史

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 版本 | 描述 | 作者 |
| 2022-01-01 | 1.0 | 发布版本 | Admin |

目录

[1. 开发指南 2](#_Toc93256301)

[1.1. 总体介绍 2](#_Toc93256302)

[1.2. 开发指导 2](#_Toc93256303)

[2. API参考 2](#_Toc93256304)

[2.1. 概览 2](#_Toc93256305)

[2.2. DISPRTC 2](#_Toc93256306)

[2.2.1. version 2](#_Toc93256307)

[2.2.2. createClient 2](#_Toc93256308)

[2.2.3. createRTCStream 2](#_Toc93256309)

[2.2.4. createRTSPStream 2](#_Toc93256310)

[2.2.5. destroy 2](#_Toc93256311)

[2.3. Browser 2](#_Toc93256312)

[2.3.1. checkRequirements 2](#_Toc93256313)

[2.3.2. getCameras 2](#_Toc93256314)

[2.3.3. getSpeakers 2](#_Toc93256315)

[2.3.4. getMicrophones 2](#_Toc93256316)

[2.4. Client 2](#_Toc93256317)

[2.4.1. login 2](#_Toc93256318)

[2.4.2. logout 2](#_Toc93256319)

[2.4.3. workStart 2](#_Toc93256320)

[2.4.4. workStop 2](#_Toc93256321)

[2.4.5. setUnattendMode 2](#_Toc93256322)

[2.4.6. setSuspendRing 2](#_Toc93256323)

[2.4.7. getCurrentOperatorInfo 2](#_Toc93256324)

[2.4.8. getOperatorStatusList 2](#_Toc93256325)

[2.4.9. GetOperatorLogList 2](#_Toc93256326)

[2.5. CallSessions 2](#_Toc93256327)

[2.5.1. makecall 2](#_Toc93256328)

[2.5.2. clearcall 2](#_Toc93256329)

[2.5.3. clearconnection 2](#_Toc93256330)

[2.6. ConferenceRoom 2](#_Toc93256331)

[2.7. VideoSessions 2](#_Toc93256332)

[2.8. SmsSessions 2](#_Toc93256333)

[2.9. FaxSessions 2](#_Toc93256334)

[2.10. Location 2](#_Toc93256335)

[2.11. DataStorage 2](#_Toc93256336)

[2.12. RTCStream 2](#_Toc93256337)

[2.13. RTSPStream 2](#_Toc93256338)

[2.14. Types 2](#_Toc93256339)

[2.14.1. DeviceInfo 2](#_Toc93256340)

[2.14.2. RTCStreamOptions 2](#_Toc93256341)

[2.14.3. RTSPStreamOptions 2](#_Toc93256342)

[3. API示例 2](#_Toc93256343)

[4. 常见问题（FAQ） 2](#_Toc93256344)

[5. 错误码 2](#_Toc93256345)

1. 开发指南
   1. 总体介绍
   2. 开发指导

请务必使用HTTPS协议或者localhost，否则SDK无法正常工作。

1. API参考
   1. 概览

Web SDK是通过HTML网页加载的JavaScript库实现在网页浏览器中调用API与服务器端连接，控制音视频通话等调度服务。

DISPRTC是Web SDK的全局对象，使用<script>标签引入SDK时产生，是Web SDK中所有可调用方法的入口，该对象含有该模块的所有成员。

请分别调用Client.on、RTCStream.on和RTSPStream.on方法添加Client, RTCStream和RTSPStream对象的监听器。

* 1. DISPRTC

DISPRTC是Web SDK的入口。

**Index**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Properties** | | |  |
|  | [version](#version) |  |  |  |
|  |  |  |  |  |
|  | **Methods** | | |  |
|  | [createClient](#createClient) | [createRTCStream](#createRTCStream) | [createRTSPStream](#createRTSPStream) |  |
|  | [destroy](#destroy) |  |  |  |
|  |  |  |  |  |

* + 1. version

|  |
| --- |
| version: *string* |

Web SDK的版本信息

* + 1. createClient

|  |
| --- |
| createClient(options:{server:*string*; token?:*string*; debug?:*boolean*}): [*Client*](#Client) |

该方法创建一个客户端对象，一个连接前调用一次即可。

**Parameters**

**options**:{**server**:*string*; **token?**:*string*; **debug?**:*boolean*}

**server**:*string*

服务器地址，比如https://localhost:80/

**token**:*string [可选]*

鉴权值。当页面刷新时，可传入已登录的token。

**debug**:*boolean [可选]*

是否开启debug模式。debug模式下浏览器会打印log日志。默认为false，即关闭状态。

**Returns** [*Client*](#Client)

返回[Client](#Client)对象

* + 1. createRTCStream

|  |
| --- |
| createRTCStream(options:[*RTCStreamOptions*](#Types_RTCStreamOptions)): [*RTCStream*](#RTCStream) |

该方法创建并返回一个实时音视频对象。

**Parameters**

**options**:[*RTCStreamOptions*](#Types_RTCStreamOptions)

配置参数。

**Returns** [*RTCStream*](#RTCStream)

返回RTCStream对象

* + 1. createRTSPStream

|  |
| --- |
| createRTSPStream(options:[*RTSPStreamOptions*](#Types_RTSPStreamOptions)): [*RTSPStream*](#RTSPStream) |

该方法创建并返回一个RTSP视频对象。

**Parameters**

**options**:[*RTSPStreamOptions*](#Types_RTSPStreamOptions)

配置参数。

**Returns** [*RTSPStream*](#RTSPStream)

返回RTSPStream对象

* + 1. destroy

|  |
| --- |
| destroy(client?:[*Client*](#Client)): *void* |

销毁Client对象。

**Parameters**

**client**: [*Client*](#Client) *[可选]*

指定要销毁的Client实例，默认销毁最初使用createClient创建的Client实例。

**Returns** *void*

* 1. Browser

Browser类是浏览器媒体设备的静态实现，你可以通过DISPRTC.Browser直接调用可用方法。

**Index**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Methods** | | |  |
|  | [checkRequirements](#checkRequirements) | [getCameras](#getCameras) | [getSpeakers](#getSpeakers) |  |
|  | [getMicrophones](#getMicrophones) | [getSupportedCodec](#getSupportedCodec) |  |  |
|  |  |  |  |  |

* + 1. checkRequirements

|  |
| --- |
| checkRequirements(): *boolean* |

检查Web SDK对正在使用的浏览器的适配情况。

请在创建Client对象([createClient](#createClient))之前调用该方法。SDK和浏览器的适配情况与浏览器类型和版本有关，不同的浏览器版本对HTML5和WebRTC的支持程度不同。

**Returns** *Boolean*

true：SDK与当前使用的浏览器适配

false：SDK与当前使用的浏览器不适配

* + 1. checkSupportWebsocket

|  |
| --- |
| checkSupportWebsocket(): *boolean* |

检查Web SDK对正在使用的浏览器对Websocket的适配情况。

请在创建Client对象([createClient](#createClient))之前调用该方法。SDK和浏览器的适配情况与浏览器类型和版本有关，不同的浏览器版本对HTML5和WebRTC的支持程度不同。

**Returns** *Boolean*

true：SDK与当前使用的浏览器适配

false：SDK与当前使用的浏览器不适配

* + 1. ~~getCameras~~

|  |
| --- |
| getCameras(): *Promise<[DeviceInfo](#Types_DeviceInfo)[]>* |

获取可用的视频输入设备。出于安全性考虑，该方法会触发浏览器的媒体设备权限申请，点击确定后才能获取到准确的设备信息。

**Returns** *Promise<[DeviceInfo](#Types_DeviceInfo)[]>*

返回DeviceInfo对象。

* + - 1. ~~getSpeakers~~

|  |
| --- |
| getSpeakers(): *Promise<[DeviceInfo](#Types_DeviceInfo)[]>* |

获取可用的音频输出设备。出于安全性考虑，该方法会触发浏览器的媒体设备权限申请，点击确定后才能获取到准确的设备信息。

**Returns** *Promise<[DeviceInfo](#Types_DeviceInfo)[]>*

返回DeviceInfo对象。

* + 1. ~~getMicrophones~~

|  |
| --- |
| getMicrophones(): *Promise<[DeviceInfo](#Types_DeviceInfo)[]>* |

获取可用的音频输入设备。出于安全性考虑，该方法会触发浏览器的媒体设备权限申请，点击确定后才能获取到准确的设备信息。

**Returns** *Promise<[DeviceInfo](#Types_DeviceInfo)[]>*

返回DeviceInfo对象。

* + 1. ~~getSupportedCodec~~

|  |
| --- |
| getSupportedCodec(): *Promise<{audio:[“OPUS”, “G711”], video:[“H264”, “VP8”]}>* |

检查当前浏览器同时支持的音视频编解码格式。

**Returns** *Promise<{audio:[“OPUS”, “G711”], video:[“H264”, “VP8”]}>*

返回当前浏览器支持的音视频编解码格式。

* 1. Client

Client类提供音视频通信的核心功能，你可以通过DISPRTC.[createClient](#createClient)创建一个Client对象。

**Index**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Methods** | | |  |
|  | [login](#login) | [getCurrentOperatorInfo](#getCurrentOperatorInfo) | [off](#off) |  |
|  | [logout](#logout) | [getOperatorStatusList](#getOperatorStatusList) | [offAll](#offAll) |  |
|  | [workStart](#workStart) | [getOperatorLogList](#getOperatorLogList) |  |  |
|  | [workStop](#workStop) | [getConnectionState](#getConnectionState) |  |  |
|  | [setUnattendMode](#setUnattendMode) | [on](#on) |  |  |
|  | [setSuspendRing](#setSuspendRing) | [once](#once) |  |  |
|  |  |  |  |  |
|  | **Events** | | |  |
|  | [onException](#onException) |  |  |  |
|  | [onConnectionStateChange](#onConnectionStateChange) |  |  |  |
|  |  |  |  |  |

* + 1. login

|  |
| --- |
| login(options:{username:*string*; password:*string*}): *Promise<void>* |

用户登录通信系统。

调用该方法登录成功后，会触发Client.on("[onConnectionStateChange](#onConnectionStateChange)") 回调。

**Parameters**

**options**:{**username**:*string*; **password**:*string*}

**username**:*string*

用户名称。

**password**:*string*

用户密码。

**Returns** *Promise<void>*

* + 1. logout

|  |
| --- |
| logout(): *Promise<void>* |

用户退出登录，退出后自动断开连接。

调用该方法成功后，会触发Client.on("[onConnectionStateChange](#onConnectionStateChange)") 回调。

**Returns** *Promise<void>*

* + 1. startWork

|  |
| --- |
| startWork(): *Promise<void>* |

该方法调用将用户的状态从“关班”，“无人值守”变为“开班”状态。

调用该方法成功后，会触发Client.on("[onConnectionStateChange](#onConnectionStateChange)") 回调。

**Returns** *Promise<void>*

* + 1. stopWork

|  |
| --- |
| startWork(): *Promise<void>* |

该方法调用将用户的状态从“开班”，“无人值守”变为“关班”状态。

调用该方法成功后，会触发Client.on("[onConnectionStateChange](#onConnectionStateChange)") 回调。

**Returns** *Promise<void>*

* + 1. setUnattend

|  |
| --- |
| setUnattendMode(calledDevice:*string,mode?:string*): *Promise<void>* |

该方法调用将用户的状态从“开班”变为“无人值守”状态。

调用该方法成功后，会触发Client.on("[onConnectionStateChange](#onConnectionStateChange)") 回调。

**Parameters**

**calledDevice**:*string*

值守号码。

**mode**:*string [可选]*

值守模式，可传入（open开启、close关闭）。缺省为open。

**Returns** *Promise<void>*

* + 1. suspendRing

|  |
| --- |
| suspendRing(): *Promise<void>* |

该方法调用设置当前新来电时，调度台不振铃，只有来电显示。挂机后，调度台恢复正常。

**Returns** *Promise<void>*

* + 1. getOperatorInfo

|  |
| --- |
| getOperatorInfo(): *Promise<[OperatorInfo](#Types_OperatorInfo)>* |

该方法调用获取当前Client的用户信息。

**Returns** *Promise<[OperatorInfo](#Types_OperatorInfo)>*

返回OperatorInfo对象。

* + 1. ~~getOperatorStatusList~~

|  |
| --- |
| getOperatorStatusList(options:{groupID:*string*; beginIndex?:*string;* count?:*string*}): *Promise<[ConnectionState](#Types_ConnectionState)[]>* |

该方法调用获取系统下的客户端状态列表。

**Parameters**

**options**:{**groupID**:*string*; **beginIndex**?:*string;* **count**?:*string*}

**groupID**:*string*

查询的组ID，为空默认查询所有。

**beginIndex**:*string [可选]*

分页起始行。

**count**:*string [可选]*

分页数量。

**Returns** *Promise<[ConnectionState](#Types_ConnectionState)[]>*

返回ConnectionState对象数组。

* + 1. ~~getOperatorLogList~~

|  |
| --- |
| getOperatorLogList(options:{keyWord:*string*;beginTime?:*string;*endTime?:*string;* beginIndex?:*string;*count?:*string*}): *Promise<[OperatorLog](#Types_Operatorlog)[]>* |

该方法调用获取客户端的操作日志。

**Parameters**

**options**:{**keyWord**:*string*;**beginTime**?:*string;***endTime**?:*string;***beginIndex**?:*string;* **count**?:*string*}

**keyWord**:*string*

查询条件，模糊匹配。

**beginTime**:*string [可选]*

起始时间。

**endTime**:*string [可选]*

截止时间。

**beginIndex**:*string [可选]*

分页起始行。

**count**:*string [可选]*

分页数量。

**Returns** *Promise<[OperatorLog](#Types_Operatorlog)[]>*

返回*OperatorLog*对象数组。

* + 1. ~~getConnectionState~~

|  |
| --- |
| getConnectionState(): *Promise<[ConnectionState](#Types_ConnectionState)>* |

该方法调用获取当前Client的客户端状态。

**Returns** *Promise<[ConnectionState](#Types_ConnectionState)>*

返回ConnectionState对象。

* + 1. on

|  |
| --- |
| on(event:*EventName*, listener:*Function*): *void* |

在该实例上添加listener函数到名为event的事件监听。

**Parameters**

**event**:*EventName*

监听的事件名称。

**listener**:*Function*

事件的回调函数。

**Returns** *void*

* + 1. once

|  |
| --- |
| once(event:*EventName*, listener:*Function*): *void* |

单次在该实例上添加listener函数到名为event的事件监听。

当监听后事件第一次触发时，该监听和回调函数就会被立刻移除，也就是只监听一次指定事件。

**Parameters**

**event**:*EventName*

监听的事件名称。

**listener**:*Function*

事件的回调函数。

**Returns** *void*

* + 1. off

|  |
| --- |
| off(event:*EventName*, listener:*Function*): *void* |

取消一个指定事件的监听。

**Parameters**

**event**:*EventName*

指定事件的名称。

**listener**:*Function*

监听事件时传入的回调函数。

**Returns** *void*

* + 1. offAll

|  |
| --- |
| offAll(): *void* |

取消所有的监听。

**Parameters**

**event**:*EventName [可选]*

指定事件的名称，如果没有指定事件，则取消所有事件的所有监听。

**Returns** *void*

* + 1. ~~onException~~

|  |
| --- |
| onException(code:*string*, msg*:string*): *void* |

SDK监测到异常事件回调。

异常事件见下表：

|  |  |  |
| --- | --- | --- |
| **事件码** | **提示信息** | **解决方法** |
|  |  |  |
|  |  |  |

**Parameters**

**code**:*string*

事件码，详细见上表。

**msg***:string*

提示信息。

**Returns** *void*

* + 1. ~~onConnectionStateChange~~

|  |
| --- |
| onConnectionStateChange(curState:[*ConnectionState*](#Types_ConnectionState), prevState*:*[*ConnectionState*](#Types_ConnectionState)): *void* |

SDK监测到系统下的客户端连接状态的事件回调。

**Parameters**

curState:[*ConnectionState*](#Types_ConnectionState)

当前的客户端状态。

prevState*:*[*ConnectionState*](#Types_ConnectionState)

原来的客户端状态。

**Returns** *void*

* 1. Client.CallSessions

Client.CallSessions类提供音视频的基本功能，如呼叫、挂断、应答、转接、广播、号码状态等。

**Index**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Methods** | | |  |
|  | [makecall](#callsessions_makecall) | [rollcall](#callsessions_rollcall) | [monitorcall](#callsessions_monitorcall) |  |
|  | [clearcall](#callsessions_clearcall) | [rollcallend](#callsessions_rollcallend) | [getcallqueuestatuslist](#callsessions_getcallqueuestatuslist) |  |
|  | [clearconnection](#callsessions_clearconnection) | [pollcall](#callsessions_pollcall) | [getcallconnstatuslist](#callsessions_getcallconnstatuslist) |  |
|  | [holdcall](#callsessions_holdcall) | [pollcallend](#callsessions_pollcallend) | [getcallconnstatus](#callsessions_getcallconnstatus) |  |
|  | [answercall](#callsessions_answercall) | [broadcastcall](#callsessions_broadcastcall) | [getcallrecordlist](#callsessions_getcallrecordlist) |  |
|  | [groupanswercall](#callsessions_groupanswercall) | [broadcastcallend](#callsessions_broadcastcallend) | [getmeetrecordlist](#callsessions_getmeetrecordlist) |  |
|  | [recordcall](#callsessions_recordcall) | [setTimingbroadcast](#callsessions_settimingbroadcast) | [getbroadcastrecordlist](#callsessions_getbroadcastrecordlist) |  |
|  | [joinmeetcall](#callsessions_joinmeetcall) | [getTimingbroadcast](#callsessions_gettimingbroadcast) |  |  |
|  | [advancecall](#callsessions_advancecall) | [singletransfercall](#callsessions_singletransfercall) |  |  |
|  | [groupcall](#callsessions_groupcall) | [consultcall](#callsessions_consultcall) |  |  |
|  | [groupcallend](#callsessions_groupcallend) | [forceinsertcall](#callsessions_forceinsertcall) |  |  |
|  | [selectcall](#callsessions_selectcall) | [forcereleasecall](#callsessions_forcereleasecall) |  |  |
|  | [selectcallend](#callsessions_selectcallend) | [forceclearcall](#callsessions_forceclearcall) |  |  |
|  |  |  |  |  |
|  | **Events** | | |  |
|  | [onCallConnStateChange](#callsessions_oncallconnstatechange) |  |  |  |
|  | [onCallQueueStateChange](#callsessions_oncallqueuestatechange) |  |  |  |
|  |  |  |  |  |

* + 1. makecall

|  |
| --- |
| makecall({calledDevice:*string*,callType?:string,duplexMode?:string, userID?:*string*})  *Promise<any>* |

呼叫远端号码。

**Parameters**

{**calledDevice**:*string*,

**callType?**:*string*,

**duplexMode?**:*string*

**userID?**:*string*

}

**calledDevice**:*string*

远端被叫号码。

**callType**:*string [可选] “audio”|“video”*

呼叫类型，*“audio”*：语音*；“video”*：视频。

**duplexMode**: *[可选] “full”|“half”*

呼叫模式，*“full”*全双工（默认）*；“half”*半双工呼叫。

**userID**:*string [可选]*

被叫用户id

**Returns** *Promise<any>*

* + 1. clearcall

|  |
| --- |
| clearcall(): *Promise<any>* |

挂断通话。

* + 1. clearconnection

|  |
| --- |
| clearconnection({calledDevice:*string*, userID?:*string*}): *Promise<any>* |

拆除通话。

**calledDevice**:*string*

*拆线号码。*

**userID**?:*string [可选]*

*用户ID*

错误：

{"msg":"called user call auth fail","code":321}

* + 1. holdcall

|  |
| --- |
| holdcall({calledDevice:*string*, userID?:*string*}): *Promise<any>* |

呼叫保持或解除保持。

* + 1. answercall

|  |
| --- |
| answercall(called: {calledDevice:*string*, userID?:*string*}): *Promise<any>* |

应答通话。

* + 1. groupanswercall
    2. recordcall
    3. joinmeetcall
    4. advancecall
    5. groupcall

|  |
| --- |
| groupcall({groupID:*string*, meetID:*string*, callMode?:*string*, meetMode?:*string*}): *Promise<any>* |

组呼。

groupID:*string*

*组ID*

meetID:*string*

*会议ID*

callMode:*string [可选] “serial”|“parallel”*

*呼叫方式，“serial”*：顺序呼叫*；“parallel”*：顺序呼叫。

meetMode:*string [可选] “audio”|“video”*

*会议类型，“audio”*：语音*；“video”*：视频。

* + 1. endGroupCall

|  |
| --- |
| endGroupCall({callSessionID:*string*}): *Promise<any>* |

结束组呼。

callSessionID:*string*

*组呼请求呼叫ID*

* + 1. selectCall

|  |
| --- |
| selectCall({called:[{calledDevice:*string,*userID*?:string*}],  meetID:string,  callMode?:string,  meetMode?:string}): *Promise<any>* |

选呼。

called:呼叫用户列表

calledDevice:string

呼叫号码。

userID:string *[可选]*

呼叫ID

meetID:*string*

*会议ID*

callMode:*string [可选] “serial”|“parallel”*

*呼叫方式，“serial”*：顺序呼叫*；“parallel”*：顺序呼叫。

meetMode:*string [可选] “audio”|“video”*

*会议类型，“audio”*：语音*；“video”*：视频。

* + 1. endSelectCall

|  |  |
| --- | --- |
| |  | | --- | | endSelectCall({callSessionID:*string*}): *Promise<any>* | |

结束选呼。

callSessionID:*string*

*请求呼叫ID*

* + 1. rollCall

|  |
| --- |
| rollCall({called:[{calledDevice:*string,*userID*?:string*}],  callMode?:string,  fileName?:string}): *Promise<any>* |

点名。

called:呼叫用户列表

calledDevice:string

呼叫号码。

userID:string *[可选]*

呼叫ID

callMode:*string [可选] “serial”|“parallel”*

*呼叫方式，“serial”*：顺序呼叫*；“parallel”*：顺序呼叫。

fileName?:*string*

*点名文件。*

* + 1. endRollCall

|  |  |
| --- | --- |
| |  | | --- | | endSelectCall({callSessionID:*string*}): *Promise<any>* | |

结束点名。

callSessionID:*string*

*请求呼叫ID*

* + 1. pollCall

|  |
| --- |
| rollCall({called:[{calledDevice:*string,*userID*?:string*}],  callMode?:string,  fileName?:string}): *Promise<any>* |

轮询。

called:呼叫用户列表

calledDevice:string

呼叫号码。

userID:string *[可选]*

呼叫ID

callType?:*string [可选] “audio”|“video”*

*呼叫类型，“audio”*：语音呼叫*；“video”*：视频呼叫。

* + 1. endPollCall

|  |  |
| --- | --- |
| |  | | --- | | endPollCall({callSessionID:*string*}): *Promise<any>* | |

结束轮询。

callSessionID:*string*

*请求呼叫ID*

* + 1. broadcastCall

|  |
| --- |
| rollCall({called:[{calledDevice:*string,*userID*?:string*}],  callMode?:string,  fileName:string,  callLoop?:int  callSessionID?:string}): *Promise<any>* |

广播呼叫。

called:呼叫用户列表

calledDevice:string

呼叫号码。

userID:string *[可选]*

呼叫ID

callMode?:*string [可选] “manual”|“file”|“tts”*

*广播方式，“manual”*：人工广播 *；“file”*：文件广播 *；“tts”*：文字广播。

callLoop?:*int [可选]*

*广播文件重复次数，缺省为0。广播方式为文件、文字广播可用。*

callSessionID?:string

广播返回呼叫id。

* + 1. endBroadcastCall

|  |  |
| --- | --- |
| |  | | --- | | endBroadcastCall({callSessionID:*string*}): *Promise<any>* | |

结束广播。

callSessionID:*string*

*广播请求呼叫ID*

* + 1. ~~setTimingbroadcast~~
    2. ~~getTimingbroadcast~~
    3. singleTransferCall

|  |
| --- |
| singleTransferCall({activeDevice:*string,*calledDevice:*string,*userID?:*string*}): *Promise<any>* |

呼叫转移。

activeDevice:string

被转移的用户号码

calledDevice:string

被转移的用户号码

userID:*string*

*用户id*

* + 1. consultCallTransfer

|  |
| --- |
| consultCallTransfer({heldDevice:*string,*heldUserID?:*string,*calledDevice:*string,*userID?:string}): *Promise<any>* |

咨询转接。

heldDevice:string

被转接的用户号码

heldUserID:string *[可选]*

被转接的用户ID

calledDevice:string

咨询的用户号码

userID:string *[可选]*

咨询的用户id

* + 1. forceInsertCall

|  |
| --- |
| forceInsertCall({calledDevice:*string,*userID?:string}): *Promise<any>* |

强插。

calledDevice:*string*

强插号码

userID:string *[可选]*

强插用户id

* + 1. forceReleaseCall

|  |
| --- |
| forceReleaseCall({calledDevice:*string,*userID?:string}): *Promise<any>* |

强拆。

calledDevice:*string*

强拆号码

userID:string *[可选]*

强拆用户id

* + 1. forceClearCall

|  |
| --- |
| forceClearCall({calledDevice:*string,*userID?:string}): *Promise<any>* |

强断。

calledDevice:*string*

强断号码

userID:string *[可选]*

强断用户id

* + 1. ~~monitorcall~~
    2. ~~Getcallqueuestatuslist~~
    3. listCallQueue

|  |
| --- |
| listCallQueue({beginIndex?:*string,*count?:string}): *Promise<any>* |

获取用户呼入队列列表。

beginIndex:*string*

呼入索引

count:string

呼入总数

* + 1. ~~getcallconnstatuslist~~
    2. getCallConnStatus

|  |
| --- |
| listCallQueue({localDevice:*string,*userID?:string}): *Promise<any>* |

获取号码详细状态信息。

localDevice:string

号码

userID:string

用户ID，可空

* + 1. ~~getcallrecordlist~~
    2. ~~getmeetrecordlist~~
    3. ~~getbroadcastrecordlist~~
    4. ~~onCallConnStateChange~~
    5. ~~onCallQueueStateChange~~
  1. Client.ConferenceRoom

Client.ConferenceRoom类提供音视频会议的基本功能，如创建、修改、删除、踢人、广播、录音等。

**Index**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Methods** | | |  |
|  | listMeet | kickMeet | startMeetVideoMix |  |
|  | createMeet | singleTalk | deleteMeet |  |
|  | editMeet | backMeet | lockMeet |  |
|  | getMeetDetail | banSpeak | meetBroadcast |  |
|  | listMeetMember | allowSpeak | getMeetingCalling |  |
|  | **Events** | | |  |
|  | [onCallConnStateChange](#callsessions_oncallconnstatechange) |  |  |  |
|  | [onCallQueueStateChange](#callsessions_oncallqueuestatechange) |  |  |  |
|  |  |  |  |  |

* + 1. listMeet

|  |
| --- |
| listMeet({data:Object}): *Promise<any>* |

获取会议列表。

data:Object

{createID:"", meetType:"", beginIndex:"", count:""}

createID:会议创建者ID

meetType: 会议类型

beginIndex: 起始索引

count：页数量

* + 1. createMeet

|  |
| --- |
| createMeet({data:Object}): *Promise<any>* |

创建会议。

data:Object

{"meetName":"","meetNum":"","meetMode":"video","isAllowSpeak":"yes","callinState":0,"callinNum":"","callinPwd":""}

meetName:string

会议名称

meetNum:string

会议号码

meetMode:string

会议模式

isAllowSpeak:string

是否允许发言

callinState:int

会议呼入开关

callinNum:string

会议呼入号码，callinState=1时必填

callinPwd:string

会议呼入密码，callinState=1时必填

* + 1. editMeet

|  |
| --- |
| editMeet({data:Object}): *Promise<any>* |

编辑会议。

data:Object

{"meetID": "","meetName":"","meetNum":"","isAllowSpeak":"yes","callinState":0,"callinNum":"","callinPwd":""}

meetID:string

会议id

meetName:string

会议名称

meetNum:string

会议号码

isAllowSpeak:string

是否允许发言

callinState:int

会议呼入开关

callinNum:string

会议呼入号码，callinState=1时必填

callinPwd:string

会议呼入密码，callinState=1时必填

* + 1. deleteMeet

|  |
| --- |
| deleteMeet({meetID:String}): *Promise<any>* |

删除会议。

meetID:String

会议id

* + 1. getMeetDetail

|  |
| --- |
| getMeetDetail({meetID:String}): *Promise<any>* |

获取会议详细信息。

meetID:String

会议id

* + 1. listMeetMember

|  |
| --- |
| listMeetMember(data:Object): *Promise<any>* |

获取会议成员列表。

data:Object

{"meetID":"","beginIndex":"0","count":"20"}

meetID:string

会议id

beginIndex:string

获取开始索引值

count:string

获取总条数

* + 1. kickMeet

|  |
| --- |
| kickMeet(data:Object): *Promise<any>* |

踢出成员。

data:Object

{"meetID":"","activeDevice":"","userID":""}

meetID:string

会议id

activeDevice:string

入会号码

userID:string

入会用户成员id

* + 1. singleTalk

|  |
| --- |
| singleTalk({meetID:string,callingDevice:string,activeDevice:string,userID?:string}): *Promise<any>* |

单独通话。

meetID:string

会议id

callingDevice:string

操作员入会手柄号

activeDevice:string

需要单独通话的入会号码

userID:string

入会用户成员id

* + 1. backMeet

|  |
| --- |
| backMeet({meetID:string,activeDevice:string,userID:string}): *Promise<any>* |

回到会场。

meetID:string

会议id

activeDevice:string

入会号码

userID:string

入会用户成员id

* + 1. banSpeak

|  |
| --- |
| banSpeak({meetID:string,activeDevice:string,userID:string}): *Promise<any>* |

禁言。

meetID:string

会议id

activeDevice:string

入会号码

userID:string

入会用户成员id

* + 1. allowSpeak

|  |
| --- |
| allowSpeak({meetID:string,activeDevice:string,userID:string}): *Promise<any>* |

发言。

meetID:string

会议id

activeDevice:string

入会号码

userID:string

入会用户成员id

* + 1. startMeetVideoMix

|  |
| --- |
| startMeetVideoMix({meetID:string,sourceInfo:Object,mixType:int,videoType:string}): *Promise<any>* |

会议混码(视频会议有效)。

meetID:string

会议id

sourceInfo:Object

混码内容，参数选一个填上即可

calledDevice:string

用户号码

userID:string

用户ID

videoID:string

视频监控ID

flowID:string

openvideo生成的flowID

mixType:int

混频类型，输出几分屏，目前支持：1，2，4，6，8，9，13，16

videoType:string

视频清晰度，默认720P

* + 1. lockMeet

|  |
| --- |
| allowSpeak({meetID:string,isLocked:string}): *Promise<any>* |

锁定会议。

meetID:string

会议id

isLocked:string

是否锁定会议，yes 锁定会议，no 解除锁定。

* + 1. meetBroadcast

|  |
| --- |
| meetBroadcast({meetID:string,isBroadcast:string,fileName:string}): *Promise<any>* |

会议广播。

meetID:string

会议id

isBroadcast:string

是否会议广播，yes 锁定会议，no 解除锁定。

fileName:string

文件名字

* + 1. getMeetingCalling

|  |
| --- |
| getMeetingCalling({meetID:string}): *Promise<any>* |

获取当前会议主叫号码。

meetID:string

会议id

* 1. Client.VideoSessions
     1. getVideoPhoneRtspUrl

|  |
| --- |
| getMeetingCalling({calledDevice:string,userID?:string}): *Promise<any>* |

获取视频通话图像。

calledDevice:string

userID?:string

* + 1. listVideoStatus

|  |
| --- |
| listVideoStatus(): *Promise<any>* |

获取视频实时状态。

* + 1. getRtspUrl

|  |
| --- |
| getRtspUrl(): *Promise<any>* |

获取视频实时状态。

* + 1. openVideo

|  |
| --- |
| openVideo({videoID:string}): *Promise<any>* |

获取视频通话图像。

videoID:string

监控、视频id

* + 1. closeVideo

|  |
| --- |
| openVideo({flowID:string}): *Promise<any>* |

关闭视频。

flowID:string

打开后的监控、视频flowid

* + 1. ptzControl
  1. Client.SmsSessions

短信状态：

unread：未读

read：已读

sending：发送中

send succ：已发送

send fail：发送失败

del：删除

* + 1. list

|  |
| --- |
| list(data:Object): *Promise<any>* |

获取短信列表。

data:Object

{"beginIndex":"0","count":"20"}

beginIndex:string

分页起始行

count:string

分页数量

* + 1. get

|  |
| --- |
| get(data:Object): *Promise<any>* |

获取短信详细信息。

data:Object

{"smsContacts":"","smsGroupID":"","beginIndex":"0","count":"20"}

smsContacts:string

短信号码，获取某个联系人的所有短信时必填

smsGroupID:string

群聊 ID，获取群聊的短信时必填

beginIndex:string

分页起始行

count:string

分页数量

* + 1. crontab

|  |
| --- |
| crontab(data:Object): *Promise<any>* |

获取定时短信列表。

data:Object

{"beginIndex":"0","count":"20"}

beginIndex:string

分页起始行

count:string

分页数量

* + 1. match

|  |
| --- |
| match(data:Object): *Promise<any>* |

根据关键字获取短信列表。

{"keyWord":"","smsDirect":"","smsStatus":"","smsRealFileName":"","beginTime":"","endTime":"","beginIndex":"0","count":"20"}

keyWord:string

搜索关键字，匹配号码、文本内容

smsDirect:string

发送方向

smsStatus:string

短信状态

smsRealFileName:string

文件名称

beginTime:string

开始时间

endTime:string

截止时间

beginIndex:string

分页起始行

count:string

分页数量

* + 1. send

|  |
| --- |
| send(data:Object): *Promise<any>* |

短信发送。

{"smsFormat":"sms","smsContent":"","smsType":"single","smsContacts":{},"smsGroupID":"","smsFileName":"","smsRealFileName":"","smsFileSize":"","sendTime":""}

smsFormat:string

短信类型，默认文本

可选值：

sms | [string]|文本| 默认值

photo | [string]|图片

video | [string]|视频

audio | [string]|语音

file | [string]|文件

smsContent:string

短信内容，支持中文，最大 1024 字节

smsType:string

发送类型

可选值：

single | [string]|单发| 默认值

group | [string]|群发

chat | [string]|群聊

mchat | [string]|会议

smsContacts:string

接收人的短信号码或座席 ID，支持群发多个，数组大小 1000。例 ["recv1","recv2"....]

smsGroupID:string

群聊ID

smsFileName:string

图片、视频上传到服务器的文件名（确保唯一性）

smsRealFileName:string

图片、视频显示文件名称

smsFileSize:string

文件大小，单位：字节

sendTime:string

定时发送时间，格式YYYYMMDDHHMMSS

* + 1. delete

|  |
| --- |
| delete(data:Object): *Promise<any>* |

短信删除。

{"smsID":"","smsContacts":"","smsGroupID":""}

smsID:string

短信任务ID，删除单条短信时填写

smsContacts:string

短信号码或座席 ID，删除某个联系人的所有短信时必填

smsGroupID:string

群聊ID，删除群聊组时填写

* + 1. read

|  |
| --- |
| read(data:Object): *Promise<any>* |

短信已读。

{"smsID":"","smsContacts":"","smsGroupID":""}

smsID:string

短信任务ID，删除单条短信时填写

smsContacts:string

短信号码或座席 ID，删除某个联系人的所有短信时必填

smsGroupID:string

群聊ID，删除群聊组时填写

* + 1. setGroup

|  |
| --- |
| setGroup(data:Object): *Promise<any>* |

群聊设置。

{"event":"smsGroupAdd","smsGroupName":"","smsGroupID":"","smsContacts":{}}

event:string

请求类型

可选值：

smsGroupAdd | [string]|群聊组新增

smsGroupMod | [string]|群聊组修改

smsGroupDel | [string]|群聊组删除

smsGroupContactAdd | [string]|群聊成员新增

smsGroupContactDel | [string]|群聊成员删除

smsGroupName:string

群聊名称

smsGroupID:string

群聊ID

smsContacts:string

群聊成员的短信号码或座席 ID，支持群发多个，数组大小 1000，新增删除群聊成员时必填

* + 1. getGroup

|  |
| --- |
| getGroup(data:Object): *Promise<any>* |

群聊查询。

{"smsGroupID":""}

smsGroupID:string

群聊ID

* 1. Client.FaxSessions
     1. list

|  |
| --- |
| list(data:Object): *Promise<any>* |

获取传真列表。

data:Object

{"beginIndex":"0","count":"20"}

beginIndex:string

分页起始行

count:string

分页数量

* + 1. get

|  |
| --- |
| get(data:Object): *Promise<any>* |

获取短信详细信息。

data:Object

{"faxContacts":"","faxGroupID":"","beginIndex":"0","count":"20"}

faxContacts:string

传真号码，获取某个联系人的所有传真时必填

faxGroupID:string

群聊 ID，获取群聊的短信时必填

beginIndex:string

分页起始行

count:string

分页数量

* + 1. crontab

|  |
| --- |
| crontab(data:Object): *Promise<any>* |

获取定时传真列表。

data:Object

{"beginIndex":"0","count":"20"}

beginIndex:string

分页起始行

count:string

分页数量

* + 1. match

|  |
| --- |
| match(data:Object): *Promise<any>* |

根据关键字获取传真列表。

{"keyWord":"","faxDirect":"","faxStatus":"","faxRealFileName":"","beginTime":"","endTime":"","beginIndex":"0","count":"20"}

keyWord:string

搜索关键字，匹配号码、文本内容

smsDirect:string

发送方向

smsStatus:string

传真状态

smsRealFileName:string

文件名称

beginTime:string

开始时间

endTime:string

截止时间

beginIndex:string

分页起始行

count:string

分页数量

* + 1. send

|  |
| --- |
| send(data:Object): *Promise<any>* |

短信发送。

{"smsFormat":"sms","smsContent":"","smsType":"single","smsContacts":{},"smsGroupID":"","smsFileName":"","smsRealFileName":"","smsFileSize":"","sendTime":""}

smsFormat:string

短信类型，默认文本

可选值：

sms | [string]|文本| 默认值

photo | [string]|图片

video | [string]|视频

audio | [string]|语音

file | [string]|文件

smsContent:string

短信内容，支持中文，最大 1024 字节

smsType:string

发送类型

可选值：

single | [string]|单发| 默认值

group | [string]|群发

chat | [string]|群聊

mchat | [string]|会议

smsContacts:string

接收人的短信号码或座席 ID，支持群发多个，数组大小 1000。例 ["recv1","recv2"....]

smsGroupID:string

群聊ID

smsFileName:string

图片、视频上传到服务器的文件名（确保唯一性）

smsRealFileName:string

图片、视频显示文件名称

smsFileSize:string

文件大小，单位：字节

sendTime:string

定时发送时间，格式YYYYMMDDHHMMSS

* + 1. delete

|  |
| --- |
| delete(data:Object): *Promise<any>* |

短信删除。

{"smsID":"","smsContacts":"","smsGroupID":""}

smsID:string

短信任务ID，删除单条短信时填写

smsContacts:string

短信号码或座席 ID，删除某个联系人的所有短信时必填

smsGroupID:string

群聊ID，删除群聊组时填写

* + 1. read

|  |
| --- |
| read(data:Object): *Promise<any>* |

短信已读。

{"smsID":"","smsContacts":"","smsGroupID":""}

smsID:string

短信任务ID，删除单条短信时填写

smsContacts:string

短信号码或座席 ID，删除某个联系人的所有短信时必填

smsGroupID:string

群聊ID，删除群聊组时填写

* 1. Client.Location
     1. subscribe

|  |
| --- |
| subscribe(data:Object): *Promise<any>* |

位置订阅。

{"mode":"subscribe","deviceCode":"","devices":[]}

mode:string

订阅或取消订阅

值可能：

subscribe

unsubscribe

deviceCode:string

订阅号码，为空代表订阅所有

devices:array

订阅多个号码

* + 1. last

|  |
| --- |
| last(data:Object): *Promise<any>* |

获取定位位置信息(最近一次的更新)。

{"deviceCode":"","devices":[],"beginIndex":"0","count":"100"}

deviceCode:string

定位号码

devices:array

定位号码，多个

beginIndex:string

分页起始行

count:string

分页数量

* + 1. history

|  |
| --- |
| history(data:Object): *Promise<any>* |

获取定位位置信息(最近一次的更新)。

{"deviceCode":"","start":"","end":"","beginIndex":"0","count":"100"}

deviceCode:string

定位号码

start:string

起始时间，格式yyyymmddhhmmss

end:string

截止时间，格式yyyymmddhhmmss

beginIndex:string

分页起始行

count:string

分页数量

* 1. Client.DataStorage

Client.DataStorage类提供基础数据的管理，如部门、用户、调度员、权限、监控等。

**Index**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Methods** | | |  |
|  | [groupSync](#data_groupsync) | [acdgroupSync](#data_acdgroupsync) | [softphoneSync](#data_softphonesync) |  |
|  | [operatorSync](#data_operatorsync) | [acdinfoSync](#data_acdinfosync) | [blacklistSync](#data_blacklistsync) |  |
|  | [userSync](#data_usersync) | [rightgroupSync](#data_rightgroupsync) | [dataSort](#data_dataSort) |  |
|  | [videogroupSync](#data_videogroupsync) | [rightmoduleSync](#data_rightmodulesync) | [getOperatorCount](#data_getoperatorcount) |  |
|  | [videoinfoSync](#data_videoinfosync) | [uimoduleSync](#data_uimodulesync) |  |  |
|  |  |  |  |  |
|  | **Events** | | |  |
|  | [onGroupSync](#data_ongroupsync) | [onVideogroupSync](#data_onvideogroupsync) |  |  |
|  | [onOperatorSync](#data_onoperatorsync) | [onVideoinfoSync](#data_onvideoinfosync) |  |  |
|  | [onUserSync](#data_onusersync) | [onRightgroupSync](#data_onrightgroupsync) |  |  |
|  |  |  |  |  |

* + 1. groupSync

|  |
| --- |
| groupSync(action:[*DataAction*](#Types_DataACtion), data:[*GroupInfo*](#Types_GroupInfo)*, filter:{}*): *Promise<[GroupInfo](#Types_GroupInfo)>* |

组操作。

action:string

数据操作类型

参考值：

add新增

update修改

delete删除

set设置

get查询

list查询列表

listid查询id列表

listsub查询子列表

count查询数量

data:Object

action为get、delete参数是：

{"groupID":""}

为listsub时参数是：

{"groupID":"0","hasChild":false,"beginIndex":0,"count":20}

为update时参数是：

{"groupID":"","fullName":"","groupName":"","parentID":"","groupType":"organ","keyNumber":0,"remark":"","icoType":"","pos\_x":"","pos\_y":"","province":"","city":"","district":"","bstrOperatorID":"","orderNO":""}

为add时参数是：

{"groupID":"","fullName":"","groupName":"","parentID":"","groupType":"organ","keyNumber":0,"remark":"","icoType":"","pos\_x":"","pos\_y":"","province":"","city":"","district":"","bstrOperatorID":""}

* + 1. operatorSync

|  |
| --- |
| operatorSync(action:[*DataAction*](#Types_DataACtion), data:[*GroupInfo*](#Types_GroupInfo)*, filter:{}*): *Promise<[GroupInfo](#Types_GroupInfo)>* |

操作员操作。

action:string

数据操作类型

参考值：

add新增

update修改

delete删除

set设置

get查询

list查询列表

listid查询id列表

listsub查询子列表

count查询数量

data:Object

action为get、delete参数是：

{"operatorID":""}

为listid时参数是：

{"groupID":"","keyWord":"","agentState":"","beginIndex":"0","count":"20"}

为listid时参数是：

{"operatorID":"0","keyWord":"","beginIndex":0,"count":20}

为update时参数是：

{"operatorID":"","userName":"","userPwd":"","realName":"","rightGroupID":"","groupID":"","videoID":[],"mainTel":"","viceTel":"","mainTelType":"phone","viceTelType":"phone","priorityLevel":"5","userType":"operator","mobile":"","faxTel":"","bcTel":"","pocTel":"","onDuty":false,"dutyTel":"","pos\_x":"","pos\_y":"","gpsBH":"","remark":"","softKey":"","icoType":"","orderNO":""}

为add时参数是：

{"operatorID":"","userName":"","userPwd":"","realName":"","rightGroupID":"","groupID":"","videoID":[],"mainTel":"","viceTel":"","mainTelType":"phone","viceTelType":"phone","priorityLevel":"5","userType":"operator","mobile":"","faxTel":"","bcTel":"","pocTel":"","onDuty":false,"dutyTel":"","pos\_x":"","pos\_y":"","gpsBH":"","remark":"","softKey":"","icoType":""}

为count时参数是：还要什么参数？

* + 1. ~~userSync~~
    2. videogroupSync

|  |
| --- |
| videogroupSync(action:[*DataAction*](#Types_DataACtion), data:[*GroupInfo*](#Types_GroupInfo)*, filter:{}*): *Promise<[GroupInfo](#Types_GroupInfo)>* |

操作员操作。

action:string

数据操作类型

参考值：

add新增

update修改

delete删除

set设置

get查询

list查询列表

listid查询id列表

listsub查询子列表

count查询数量

data:Object

action为get、delete参数是：

{"groupID":""}

为list、listid时参数是：

{"groupID":"0","beginIndex":0,"count":20}

为update时参数是：

{"groupID":"","groupName":"","parentID":"","remark":"","icoType":"","pos\_x":"","pos\_y":"","bstrOperatorID":"","orderNO":""}

为add时参数是：

{"groupID":"","groupName":"","parentID":"","remark":"","icoType":"","pos\_x":"","pos\_y":"","bstrOperatorID":"","orderNO":""}

* + 1. videoSync

|  |
| --- |
| videoSync(action:[*DataAction*](#Types_DataACtion), data:[*GroupInfo*](#Types_GroupInfo)*, filter:{}*): *Promise<[GroupInfo](#Types_GroupInfo)>* |

操作员操作。

action:string

数据操作类型

参考值：

add新增

update修改

delete删除

set设置

get查询

list查询列表

listid查询id列表

listsub查询子列表

count查询数量

data:Object

action为get、delete参数是：

{"videoID":""}

为list、listid时参数是：

{"groupID":"","keyWord":"","beginIndex":"0","count":"20"}

为update时参数是：

{"videoID":"","videoName":"","videoMode":0,"IPAddr":"","IPPort":"","userName":"","userPwd":"","puid":"","gate":0,"pos\_x":"","pos\_y":"","gpsBH":"","remark":"","icoType":"","orderNO":"","groupID":[{"groupID":"","orderNO":""}],"customGroupID":[{"groupID":"","orderNO":""}]}

为add时参数是：

{"videoID":"","videoName":"","videoMode":0,"IPAddr":"","IPPort":"","userName":"","userPwd":"","puid":"","gate":0,"pos\_x":"","pos\_y":"","gpsBH":"","remark":"","icoType":"","groupID":[{"groupID":"","orderNO":""}],"customGroupID":[{"groupID":"","orderNO":""}]}

* + 1. ~~acdgroupSync~~
    2. ~~acdinfoSync~~
    3. ~~rightgroupSync~~
    4. ~~rightmoduleSync~~
    5. ~~uimoduleSync~~
    6. ~~softphoneSync~~
    7. blacklistSync

|  |
| --- |
| blacklistSync (action:[*DataAction*](#Types_DataACtion), data:[*GroupInfo*](#Types_GroupInfo)*, filter:{}*): *Promise<[GroupInfo](#Types_GroupInfo)>* |

操作员操作。

action:string

数据操作类型

参考值：

add新增

update修改

delete删除

set设置

get查询

list查询列表

listid查询id列表

listsub查询子列表

count查询数量

data:Object

为delete时参数是：

{"blacklistID":""}

为list、listid时参数是：

{"keyWord":"","beginIndex":"0","count":"20"}

为update时参数是：

{"blacklistID":"","blacklistTel":"","description":"","remark":"","beginTime":"","endTime":""}

为add时参数是：

{"blacklistTel":"","description":"","remark":"","beginTime":"","endTime":""}

* + 1. ~~dataSort~~
    2. ~~getOperatorCount~~
    3. ~~onGroupSync~~
    4. ~~onOperatorSync~~
    5. ~~onUserSync~~
    6. ~~onVideogroupSync~~
    7. ~~onVideoinfoSync~~
    8. ~~onRightgroupSync~~
    9. emloyeeSync

|  |
| --- |
| emloyeeSync(action:[*DataAction*](#Types_DataACtion), data:[*GroupInfo*](#Types_GroupInfo)*, filter:{}*): *Promise<[GroupInfo](#Types_GroupInfo)>* |

操作员操作。

action:string

数据操作类型

参考值：

add新增

update修改

delete删除

set设置

get查询

list查询列表

listid查询id列表

listsub查询子列表

count查询数量

data:Object

为delete时参数是：

{"dutyID":""}

为list、listid时参数是：

{"keyWord":"","beginIndex":"0","count":"20"}

为update时参数是：

{"dutyID":"","dutyName":"","remark":""}

为add时参数是：

{"dutyName":"","remark":""}

* 1. RTCStream

RTCStream类提供的方法用于定义音视频对象实时通信的能力，例如音视频采集播放、编码配置等。

您可以使用DISPRTC.[createRTCStream](#createRTCStream)创建音视频对象，一个音视频对象包括通话中发布的本地音视频和远端的音视频流。

**Index**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Methods** | | |  |
|  | setRegisterProfile | play | startMediaRecording |  |
|  | setScreenProfile | stop | stopMediaRecording |  |
|  | setVideoProfile | isPlaying | on |  |
|  | init | hasVideo | once |  |
|  | destory | mute | off |  |
|  | open | unmute | offAll |  |
|  | close | takeSnapshot |  |  |
|  |  |  |  |  |
|  | **Events** | | |  |
|  | onRTCLocalConnectionStateChange |  |  |  |
|  | onRTCRemoteConnectionStateChange |  |  |  |
|  |  |  |  |  |

* + 1. setRegisterProfile

|  |
| --- |
| setRegisterProfile(phone:*string*; password:*string*): *Promise<void>* |

设置本地注册账号信息。

必须在RTCStream.init之前调用。

**Parameters**

**phone**:*string*

sip号码。

**password**:*string*

密码。

**Returns** *Promise<void>*

* + 1. setScreenProfile

|  |
| --- |
| setScreenProfile(frameRate:*number*; resolution:*number*): *Promise<void>* |

设置屏幕共享的属性。

该方法设置屏幕共享时屏幕的显示属性，必须在RTCStream.init之前调用。

该方法可选。缺省时，按RTCStream类默认属性进行。

该方法仅可对本地流调用。

**Parameters**

**frameRate**:*number*

设置本端屏幕共享视频帧率。参考[VideoFrameRate](#Types_VideoFrameRate)。

**resolution**:*number*

设置本端屏幕共享视频分辨率。参考[VideoResolutionQuality](#Types_VideoResolutionQuality)。

**Returns** *Promise<void>*

* + 1. setVideoProfile

|  |
| --- |
| setVideoProfile(frameRate:*number*; resolution:*number*): *Promise<void>* |

设置摄像头视频属性。

该方法可选，必须在RTCStream.init之前调用。缺省时，按RTCStream类默认属性进行。

该方法仅可对本地流调用。

**Parameters**

**profile**:{**frameRate**:*number*; **resolution**:*number*}

**frameRate**:*number*

设置本端视频帧率。参考[VideoFrameRate](#Types_VideoFrameRate)。

**resolution**:*number*

设置本端视频分辨率。参考[VideoResolutionQuality](#Types_VideoResolutionQuality)。

**Returns** *Promise<void>*

* + 1. init

|  |
| --- |
| init(): *Promise<void>* |

初始化本地音视频对象，并且向服务器去注册。

**Returns** *Promise<void>*

* + 1. destory

|  |
| --- |
| destroy(): *Promise<void>* |

销毁本地音视频对象。

**Returns** *Promise<void>*

* + 1. ~~open~~

|  |
| --- |
| open(options:{deviceId?:*string*; screenId?:*string*}): *Promise<void>* |

打开视频输入设备，如摄像头或屏幕共享。并且发布出去。

当打开新的输入设备时，SDK会自动关闭旧的输入设备。

**Parameters**

**options**:{**deviceId?**:*string*; **screenId?:***string*}

**deviceId**:*string [可选]*

指定要开启的摄像头设备ID，通过[getCameras](#getCameras)获取。

**screenId**:*string [可选]*

指定要开启的Election屏幕共享的数据源ID，自行获取。

**Returns** *Promise<void>*

* + 1. ~~close~~

|  |
| --- |
| close(): *Promise<void>* |

关闭视频输入设备。

**Returns** *Promise<void>*

* + 1. ~~play~~

|  |
| --- |
| play(view:*HTMLElement*, options:{audio?:*boolean*; video?:*boolean*}): *Promise<void>* |

播放音视频流。

**Parameters**

**view**:*HTMLElement*

div标签，播放画面的dom容器节点。

**options**:{**audio?**:*boolean*; **video?**:*boolean*}

**audio**:*boolean [可选]*

是否播放音频流。

**video**:*boolean [可选]*

是否播放视频流。

**Returns** *Promise<void>*

* + 1. ~~stop~~

|  |
| --- |
| stop(): *Promise<void>* |

停止音视频流。

**Returns** *Promise<void>*

* + 1. ~~isPlaying~~

|  |
| --- |
| isPlaying(type: “audio”|“video”): *Promise<boolean>* |

返回音视频当前是否在播放状态。

**Parameters**

**type**: “audio”|“video”

媒体流类型。

**Returns** *Promise<boolean>*

true：该音视频流正在渲染或播放。

false：该音视频流没有渲染。

* + 1. ~~hasVideo~~

|  |
| --- |
| hasVideo(): *Promise<boolean>* |

该方法用于确认当前是否视频通话。

**Returns** *Promise<boolean>*

true：视频通话。

false：音频通话。

* + 1. ~~mute~~

|  |
| --- |
| mute(): *Promise<void>* |

静音

**Returns** *Promise<void>*

* + 1. ~~unmute~~

|  |
| --- |
| unmute(): *Promise<void>* |

取消静音

**Returns** *Promise<void>*

* + 1. ~~stopCamera~~

|  |
| --- |
| stopCamera(): *Promise<void>* |

关闭摄像头

**Returns** *Promise<void>*

* + 1. ~~openCamera~~

|  |
| --- |
| openCamera(): *Promise<void>* |

打开摄像头

**Returns** *Promise<void>*

* + 1. takeSnapshot

|  |
| --- |
| takeSnapshot(): *Promise<void>* |

截图

**Returns** *Promise<void>*

* + 1. ~~startMediaRecording~~

|  |
| --- |
| startMediaRecording(): *Promise<void>* |

开始本地录像

**Returns** *Promise<void>*

* + 1. ~~stopMediaRecording~~

|  |
| --- |
| stopMediaRecording(): *Promise<void>* |

停止本地录像

**Returns** *Promise<void>*

* + 1. on

|  |
| --- |
| on(event:*EventName*, listener:*Function*): *void* |

在该实例上添加listener函数到名为event的事件监听。

**Parameters**

**event**:*EventName*

监听的事件名称。

**listener**:*Function*

事件的回调函数。

**Returns** *void*

* + 1. once

|  |
| --- |
| once(event:*EventName*, listener:*Function*): *void* |

单次在该实例上添加listener函数到名为event的事件监听。

当监听后事件第一次触发时，该监听和回调函数就会被立刻移除，也就是只监听一次指定事件。

**Parameters**

**event**:*EventName*

监听的事件名称。

**listener**:*Function*

事件的回调函数。

**Returns** *void*

* + 1. off

|  |
| --- |
| off(event:*EventName*, listener:*Function*): *void* |

取消一个指定事件的监听。

**Parameters**

**event**:*EventName*

指定事件的名称。

**listener**:*Function*

监听事件时传入的回调函数。

**Returns** *void*

* + 1. offAll

|  |
| --- |
| offAll(event?:*EventName*): *void* |

指定一个事件，取消其所有的监听。

**Parameters**

**event**:*EventName [可选]*

指定事件的名称，如果没有指定事件，则取消所有事件的所有监听。

**Returns** *void*

* + 1. onRTCLocalConnectionStateChange

订阅本地音视频端连接状态。

* + 1. onRTCRemoteConnectionStateChange

订阅远端音视频端连接状态。

* 1. RTSPStream

RTSPStream类提供的方法用于播放远端监控画面的视频图像。

* + 1. play

|  |
| --- |
| play(): *Promise<void>* |

播放RTSP视频流。

**Returns** *Promise<void>*

* + 1. ~~stop~~

~~isPlaying~~

~~takeSnapshot~~

~~startMediaRecording~~

~~stopMediaRecording~~

~~on~~

|  |
| --- |
| on(event:*EventName*, listener:*Function*): *void* |

在该实例上添加listener函数到名为event的事件监听。

**Parameters**

**event**:*EventName*

监听的事件名称。

**listener**:*Function*

事件的回调函数。

**Returns** *void*

~~once~~

|  |
| --- |
| once(event:*EventName*, listener:*Function*): *void* |

单次在该实例上添加listener函数到名为event的事件监听。

当监听后事件第一次触发时，该监听和回调函数就会被立刻移除，也就是只监听一次指定事件。

**Parameters**

**event**:*EventName*

监听的事件名称。

**listener**:*Function*

事件的回调函数。

**Returns** *void*

~~off~~

|  |
| --- |
| off(event:*EventName*, listener:*Function*): *void* |

取消一个指定事件的监听。

**Parameters**

**event**:*EventName*

指定事件的名称。

**listener**:*Function*

监听事件时传入的回调函数。

**Returns** *void*

~~offAll~~

|  |
| --- |
| offAll(event?:*EventName*): *void* |

指定一个事件，取消其所有的监听。

**Parameters**

**event**:*EventName [可选]*

指定事件的名称，如果没有指定事件，则取消所有事件的所有监听。

**Returns** *void*

close

|  |
| --- |
| play(): *Promise<void>* |

关闭RTSP视频流。

**Returns** *Promise<void>*

* 1. Types

全局类型的Interface定义。

**Index**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Interfaces** | | |  |
|  | [DeviceInfo](#Types_DeviceInfo) |  | [OperatorLog](#Types_Operatorlog) |  |
|  | [RTCStreamOptions](#Types_RTCStreamOptions) |  | [OperatorInfo](#Types_OperatorInfo) |  |
|  | [RTSPStreamOptions](#Types_RTSPStreamOptions) |  | [GroupInfo](#Types_GroupInfo) |  |
|  | [ConnectionState](#Types_ConnectionState) |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  | **Variables** | | |  |
|  | [ConnectionDisconnectedReason](#Types_ConnectionDisReason) |  |  |  |
|  | [VideoResolutionQuality](#Types_VideoResolutionQuality) |  |  |  |
|  | [VideoFrameRate](#Types_VideoFrameRate) |  |  |  |
|  | [DataAction](#Types_DataACtion) |  |  |  |
|  |  |  |  |  |

* + 1. DeviceInfo

媒体设备信息。

**Properties**

* **deviceId**: *string*

该设备所独有的设备ID。

* **label:***string*

设备标签，用于区分设备名称。

* + 1. RTCStreamOptions

RTCStream配置信息。

**Properties**

* **audio:** *boolean*是否采集音频。
* **video:** *boolean*是否采集视频。
* **cameraId:** *string [可选]*摄像头设备deviceId，通过[getCameras](#getCameras)获取。缺省默认。
* **microphoneId:** *string [可选]*麦克风设备deviceId，通过[getMicrophones](#getMicrophones)获取。缺省默认。
* **client:** [*Client*](#Client) *[可选]*要RTCStream绑定的client实例对象。默认是最初使用[createClient](#createClient)创建的Client实例。
* **server:** *string [可选]*服务器连接信息。缺省默认[createClient](#createClient)配置的server地址。
* **ringtone:** *HTMLElement [可选]*audio标签，振铃音的DOM容器节点。缺省默认。
* **ringbacktone:** *HTMLElement [可选]*audio标签，回铃音的DOM容器节点。缺省默认。
  + 1. RTSPStreamOptions

RTSPStream配置信息。

**Properties**

* **client:** [*Client*](#Client) *[可选]*要RTSPStream绑定的client实例对象。默认是最初使用[createClient](#createClient)创建的Client实例。
* **server:** *string [可选]*服务器连接信息。缺省默认[createClient](#createClient)配置的server地址。
  + 1. ConnectionState

客户端连接状态信息。

**Properties**

* **user\_id:** *string*

客户端座席ID，相等于OperatorID。

* **agentState:** *“Disconnected”|“Connecting”|“Connected-WorkStart”|“Connected-WorkStop” |“Connected-WorkAfterCall”|“ReConnecting”*

客户端连接状态。SDK和服务器有以下6种连接状态：

“Disconnected”: 连接断开，未登录状态。

“Connecting”: 正在连接中。

“Connected-WorkStart”: 已连接，登录成功，开班状态。

“Connected-WorkStop”: 已连接，关班状态。

“Connected-WorkAfterCall”: 已连接，无人值守状态。

“ReConnecting”: 正在重连中，音网络断开或切换而导致SDK与服务器的连接终端，SDK会自动重连，重连过程中连接状态变为“ReConnecting”。

* **unattendDevice:** *string [可选]*

值守号码。

* **reason:** [*ConnectionDisconnectedReason*](#Types_ConnectionDisReason) *[可选]*

当agentState为“Disconnected”，这里表示断开连接的原因。

* + 1. ConnectionDisconnectedReason

连接断开的原因。

**Enumeration members**

* **DISCONNECTED\_NONE**缺省。
* **DISCONNECTED\_LEAVE**用户正常退出。
* **DISCONNECTED\_OFFLINE**用户被下线。
* **DISCONNECTED\_ERROR**服务器返回错误，比如重连的密码错误。
  + 1. VideoResolutionQuality

分辨率设置。

**Enumeration members**

* **VIDEO\_QUALITY\_720P**720p，缺省。
* **VIDEO\_QUALITY\_1080P**1080p。
* **VIDEO\_QUALITY\_480P**480p。
* **VIDEO\_QUALITY\_180P**180p。
  + 1. VideoFrameRate

视频帧率设置。

**Enumeration members**

* **VIDEO\_FRAME\_RATE\_20**20帧，缺省。
* **VIDEO\_FRAME\_RATE\_25**25帧
* **VIDEO\_FRAME\_RATE\_15**15帧。
* **VIDEO\_FRAME\_RATE\_10**10帧。
* **VIDEO\_FRAME\_RATE\_5**5帧。
  + 1. DataAction

数据操作动作。

**Enumeration members**

* **ACTION\_ADD**
* **ACTION\_UPDATE**
* **ACTION\_DELETE**
* **ACTION\_SET**
* **ACTION\_GET**
* **ACTION\_LIST**
* **ACTION\_LISTID**
* **ACTION\_LISTSUB**
  + 1. OperatorInfo
    2. OperatorLog
    3. GroupInfo

1. API示例
2. 常见问题（FAQ）
3. 错误码

以下为SDK可能抛出的错误，请参考下表进行处理。

|  |  |  |  |
| --- | --- | --- | --- |
| **错误代码** | **值** | **描述** | **处理方法** |
| {"msg":"user status is error, not allow call","code":308} | 308 | 用户状态错误无法转移通话 | 是否与该操作员通话中 |
|  | 401 |  |  |
|  | 403 |  |  |
| {"msg":"username not exist","code":404} | 404 | 用户不存在 | 输入正确的账号 |
|  | 405 |  |  |
|  | 406 |  |  |
| {"msg":"mix type not equal source node","code":469} | 469 | sourceinfo与mixtype数不同 | 检查参数有无错误，检查个数有无错误。 |
| {"msg":"user\_has\_login in elsewhere","code":480} | 480 | 监听：用户已在其他地方登录 | 处理销毁或者退出操作 |
|  | 500 |  |  |
|  | 501 |  |  |
|  | 502 |  |  |
|  | 503 |  |  |
|  | 504 |  |  |
|  | 505 |  |  |
| {"msg":"signature error","code":701} | 701 | 密码不正确 | 输入正确的密码 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

伪代码：

主手柄启动选择：

以makecall为例

function makecall（）

{

if 只有左手柄 then

caller=主手柄

else if 左手柄==空闲 and 有手柄==空闲 then

caller=主手柄

else if 左手柄==离线 and 有手柄==空闲 then

caller== 右手柄

else if 左手柄==保持 then

caller==左手柄

}

组呼，选呼，应答等，除了以上的判断条件外，优先再判断手柄是否在会议中

if 只有左手柄 then

caller=主手柄

if 左手柄 in meet.id then

caller=左手柄

elseif 右手柄 in meet.id then

caller=右手柄

else if 左手柄==空闲 and 有手柄==空闲 then

caller=主手柄

else if 左手柄==离线 and 有手柄==空闲 then

caller== 右手柄

else if 左手柄==保持 then

caller==左手柄

应答：

answer（）

{

if rtc.注册号码=应答的主叫 andrtc.注册号码==在线 then

rtc.应答

}

clearconnection ()

{

if rtc.注册号码=拆线的主叫 and rtc.注册号码==在线 then

rtc.handup

}