会捷通客户端 SDK (windows 版) 开发文档

更新记录

日期	更新内容	版本
2020年8月6日	创建文档	V 1.0
2020年8月14日	添加接口列表和错误码列表	V 1.1

概述

会捷通客户端 SDK 是一款视频会议终端开发套件，开发者可以利用本 SDK 开发出具有清晰流畅的音视频体验和高清内容协作的音视频会议终端应用。

会捷通客户端 SDK 具备强大的网络适应性，和独特的音视频抗网络丢包算法，配合会捷通云视讯平台使用，可以保证在 30% 网络丢包环境下视频依然清晰流畅，即使网络丢包高达 50%，依然可以保证音频通畅。 会捷通客户端 SDK 提供了丰富，而且简单易用的 API 接口。开发者不需要掌握丰富的音视频和信令相关知识，也可以使用本 SDK 开发出专业的视频会议软终端应用。 本文档详细介绍了 SDK 的各项功能，以及它们的使用方法。

系统要求

- 操作系统要求：Windows 10，Windows 7 SP1 （32 位或64 位）
- 计算机配置推荐
    - 低级性能配置：Intel i3-8100 同级别处理器，4G 内存
    - 中级性能配置：Intel i5-8250 同级别处理器，8G 内存
    - 高级性能配置：Intel i5-9600K 同级别或高级别处理器，16G 内存

典型系统集成方案

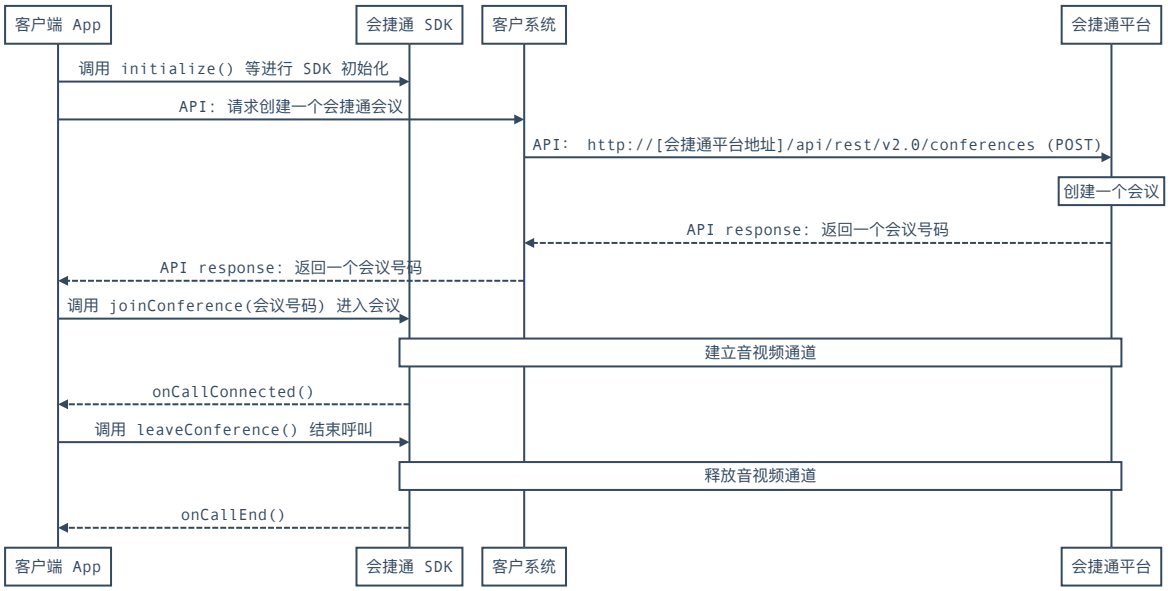
如果您开发的产品包含客户端应用和业务服务系统，需要为系统加入音视频通讯能力时，那么您只需要在客户端集成会捷通 SDK，同时通过您的业务服务系统和会捷通平台服务 (我们的公有云服务或者您自建的私有化部署会捷通平台) 进行集成。

- 客户端 App 集成会捷通 SDK，通过会捷通 SDK 实现客户端建立音视频呼叫的功能
- 业务服务系统通过使用会捷通平台的 REST API 使用平台提供的功能，包括创建会议、管理会议等。

一个典型的业务流程示例

下图简要说明了一个典型的业务流程

- 客户端 App 启动，进行会捷通 SDK 的初始化设置
- 客户端 App 向自己的业务系统申请创建一个视频会议
- 业务系统使用会捷通平台 API 创建一个视频会议，并把创建出的会议号码返回给客户端 App
- 客户端 App 调用会捷通 SDK 的接口方法加入到这个会议
- 客户端 App 调用会捷通 SDK 的接口方法离开会议



会捷通平台提供了丰富的 API 接口，您的业务系统可以根据需要选择使用。比如，在上边的例子中，可以在会议进行中通过 API 接口进行对与会者静音、开始录制会议等会议控制操作。

API接口列表

初始化和释放

接口	接口描述
ev::engine::IEVEngine* createEVEngine();	创建 EVEngine 实例
void deleteEVEngine(ev::engine::IEVEngine* engine);	删除 EVEngine实例
int initialize(const char *config_path, const char * config_file_name);	初始化 SDK
int release();	释放SDK
int setRootCA(const char * root_ca_path);	设置 rootca 文件路径
int setUserImage(const char * background_file_path, const char * user_image_path);	设置用户头像和入会后显示的背景图像
int enableSecure(bool enable);	使用SSL加密网络传输
int setUserAgent(const char * company, const char * version);	设置软件提供商信息
int registerEventHandler(IEVEventHandler * handler);	注册事件接收器
int unregisterEventHandler(IEVEventHandler * handler);	取消注册事件接收器

日志

接口	接口描述
void setLog(EV_LOG_LEVEL level, const char * log_path, const char * log_file_name, unsigned int max_file_size);	设置log输出级别，路径和最大文件大小
void setConsoleLog(EV_LOG_LEVEL level);	设置输出日志到控制台
void enableLog(bool enable);	开启日志
std::string compressLog();	压缩日志文件
int uploadFeedbackFilesWithAddress(std::vector std::string filePath, std::string contact, std::string description, std::string address)	上传日志到后台服务器。

用户登录登出

接口	接口描述
int loginWithLocation(const char * location_server, unsigned int port, const char * username, const char * encrypted_password) ;	登录
int logout();	退出登录
std::string EVEngine::encryptPassword(EV_ENCRYPT_TYPE type, const char * password, const char* akey);	加密方法
std::string descryptPassword(EV_DESCRYPT_TYPE type, const char * password, const char* akey, const char* iv);	解密方法
int downloadUserImage(const char * path);	下载用户头像到本地
int uploadUserImage(const char * path);	上传用户头像到服务器
int changePassword(const char * encrypted_oldpassword, const char *encrypted_newpassword);	修改密码
int changeDisplayName(const char * display_name);	修改显示名字
int getUserInfo(EVUserInfo & userinfo);	获取用户信息
std::string getDisplayName();	获取用户显示名字

会议能力设置

接口	接口描述
int setMaxRecvVideo(unsigned int num)	设置可以接收的最大入会视频流数量
int enableContent(bool enable)	设置会议中是否允许发送或接收双流
bool contentEnabled()	查询会议中是否有发送或接收双流的能力
int enableContentHighFPS(bool enable)	设置会议中是否发送高清双流
bool contentHighFPSEnabled()	获取当前设置的是帧率优先还是清晰度优先
enableRelayStreamDataCb(EV_STREAM_TYPE type, bool enable)	是否开启回调onAudioData和onVideoPreviewFrame
int enableAdaptiveResolution(bool enable)	设置是否根据CPU的使用情况，来动态调整分辨率。
bool adaptiveResolutionEnabled()	获取是否动态调整分辨率的设置
bool highFPSEnabled();	获取当前是否是帧率优先
int enableHighFPS(bool enable);	设置视频会议中帧率优先
int enableHD(bool enable);	设置视频会议中高清优先
bool HDEnabled();	获取当前设置是否是高清优先
int setBandwidth(unsigned int kbps);	设置视频会议的带宽
unsigned int getBandwidth();	获取设置的视频会议带宽

会议通讯

接口	接口描述
int joinConference(const char * conference_number, const char * display_name, const char * password);	加入会议
int joinConference(const char * conference_number, const char * display_name, const char * password, EV_SVC_CALL_TYPE type);	加入会议
int joinConference(const char * conference_name, V_SVC_CONFERENCE_NAME_TYPE name_type, const char * display_name, const char * password, EV_SVC_CALL_TYPE type);	加入会议
int joinConferenceWithLocation(const char * location_server, unsigned int port, const char * conference_number, const char * display_name, const char * password);	匿名入会
int joinConferenceWithLocation(const char * location_server, unsigned int port, const char * conference_name, EV_SVC_CONFERENCE_NAME_TYPE name_type, const char * display_name, const char * password);	匿名入会
int leaveConference();	离开会议
int terminateConference();	结束会议
int setInConfDisplayName(const char * display_name, int len)	会议中设置显示的会场名称

会议操作

接口	接口描述
bool cameraEnabled();	获取会议中摄像头打开/关闭状态
int enableCamera(bool enable)	打开/关闭摄像头
int switchCamera()	切换摄像头
bool micEnabled()	获取麦克风打开/关闭状态
int enableMic(bool enable)	打开/关闭麦克风
bool remoteMuted()	获取会控服务器的静音状态
int requestRemoteUnmute(bool val)	请求会控服务器解除静音
int getCallInfo(EVCallInfo & call_info)	获取呼叫信息
int setVideoActive(int active)	会议中切换视频模式和语音模式
int videoActive()	获取当前是视频模式还是语音模式
bool isConferenceHoster()	获取是否是会议主持人

会议视频流窗口设置

接口	接口描述
int setLocalVideoWindow(void * id)	设置本地视频窗口句柄
int setRemoteContentWindow(void * id)	设置远端内容窗口句柄
int setRemoteVideoWindow(void * id[], unsigned int size)	设置远端视频窗口句柄
int setLocalContentWindow(void * id, EV_CONTENT_MODE mode)	设置发送内容窗口句柄
void* getLocalVideoWindow()	获取本地视频窗口句柄
void* getRemoteVideoWindow(void * id[], unsigned int size)	获取远端视频窗口句柄
virtual void * getRemoteContentWindow()	获取接收远端内容窗口句柄
void* getLocalContentWindow()	获取本地发送视频窗口句柄
int repaintVideo()	通知sdk重绘远端视频内容

会议视频流布局

接口	接口描述
int setLayoutCapacity(EV_LAYOUT_MODE mode, EV_LAYOUT_TYPE types[], unsigned int size)	设置分屏的能力集
int setLayout(EVLayoutRequest & layout)	设置布局样式
int setRecvVideo(unsigned int num, EVVideoSize max_resolution)	设置会议中收到的People内容源的最大路数
int setPageSetting(EVPageSetting * setting)	翻页设置
int getPageInfo(EVPageInfo * info)	获取翻页状态
int setPage(int page_number)	翻到指定页数

会议内容流（双流）

接口	接口描述
int getContentInfo(EVContentInfo & content_info)	获取辅流信息
int sendContent()	开始发送双流
int stopContent()	停止发送双流
int enableContentAudio(bool enable)	发送双流时同时发送声音
bool contentAudioEnabled()	获取是否发送双流声音的设置
int setLocalContentSource(EVContentSourceInfo & info)	发送选择的内容流窗口
std::vector getContentSourceList()	获取当前可发送的内容源列表
getContentSourceBitmap(EVContentSourceInfo & info, EV_BITMAP_TYPE bitmapType, int maxWidth, int maxHeight, EVBitmap & bitmap)	获取内容源的窗口缩略图，图标
void releaseContentSourceBitmap	释放缩略图，图标等内存资源

设备操作

接口	接口描述
int EVEngine::setAudioFile(const char * play_file)	设置播放声音文件
int startAudioPlay()	开始播放声音
stopAudioPlay()	停止播放声音
EVVolume getVolume(EV_DEVICE_TYPE type)	获取当前音频设备音量
int setVolume(EV_DEVICE_TYPE type, EVVolume & vol)	设置选中音频设备的音量
std::vector getDevices(EV_DEVICE_TYPE type)	获取当前所有可以连接的设备列表
void setDevice(EV_DEVICE_TYPE type, unsigned int id)	选中指定的设备
EVDevice getDevice(EV_DEVICE_TYPE type)	获取当前已经选中的设备
virtual int setCameraCapture(EVCameraCaptureInfo & info)	设置摄像头参数
int getCameraCapture(EVCameraCaptureInfo & info)	获取摄像头参数
std::vector getCameraCaptureList()	获取已连接摄像头属性列表

美颜功能

接口	接口描述
int setEnhanceBrightness(int brightness)	设置提亮效果
int getEnhanceBrightness()	获取提亮设置数值
int setBeautifyFace(int beautify)	设置是否打开美颜
int getBeautifyFace()	获取美颜设置

网络状态统计

接口	接口描述
float getNetworkQuality()	获取网络信号强度
getStats(EVStats & stats)	获取视频流统计信息

设置和其它

接口	接口描述
int setPort(EV_STREAM_TYPE stream, int port)	设置各种媒体流传输端口
int getPort(EV_STREAM_TYPE stream)	获取媒体流传输端口
int setRenderType(EV_RENDER_TYPE type)	设置渲染的模式
EV_RENDER_TYPE getRenderType()	获取当前渲染的模式

开发入门

初始化 SDK

使用EVSDK时，需要连接库文件evsdk.dll, 编译选项添加 evsdk.lib。

程序中使用示例如下

```
IEVEngine * evengine = createEVEngine();
EVEventObsever evobsever;
evengine->registerEventHandler(&evobsever);
```

日志设置

EVSDK 会持续收集日志，打开日志收集需要做以下设置，可以设置log文件名字和大小。

```
evengine->setLog(EV_LOG_LEVEL_MESSAGE, "./", "evsdk", 20 * 1024 * 1024);
evengine->enableLog(TRUE);
```

如果不想设置日志输入到控制台，可以使用如下方式

```
evengine->setConsoleLog(ORTP_FATAL);
```

SDK Config 文件

设置config文件得路径和文件名字，用来保存应用配置和历史呼叫之类的信息。

```
evengine->initialize("./", "config");
```

rootca 文件

添加 rootca.pem 文件，将文件拷贝到某目录下边，设置给 SDK。

```
evengine->setRootCA("./rootca.pem");
```

> 如果不设置rootca，加密的注册会失败。

设备设置

SDK 提供方法来让用户选择自己的设备，具体如下

```
获取当前设备的列表
std::vector<EVDevice> devices1 = evengine->getDevices(EV_DEVICE_AUDIO_CAPTURE);
std::vector<EVDevice> devices2 = evengine->getDevices(EV_DEVICE_AUDIO_PLAYBACK);
std::vector<EVDevice> devices3 = evengine->getDevices(EV_DEVICE_VIDEO_CAPTURE);
获取当前已经选中的设备
EVDevice device1 = evengine->getDevice(EV_DEVICE_AUDIO_CAPTURE);
EVDevice device2 = evengine->getDevice(EV_DEVICE_AUDIO_PLAYBACK);
EVDevice device3 = evengine->getDevice(EV_DEVICE_VIDEO_CAPTURE);
```

设置呼叫基本能力

带宽

```
SVC 可以设置2M到4M带宽
int const bandwidth = 2048;

具体带宽设置需要根据使用场景的网络情况设定。
evengine->setBandwidth(bandwidth);
```

接收视频流的数量上限

```
int const maxRxCount = 16;
evengine->setMaxRecvVideo(maxRxCount);
```

登录

用户正常登录

```
/**
 用户定位登录（包含定位跟登录）

 @param location_server 定位服务器
 @param port 端口号
 @param username 用户名
 @param password 密码
 @return 返回API结果
 */
evengine->loginWithLocation(location_server, port, username, encrypted_password) ;
/** 登录后会收到SDK以下回调 */
//定位或者登录错误会回调
void onError(EVError & err);
//登录成功会回调
void onLoginSucceed(EVUserInfo & user);
```

匿名方式入会

匿名登录方式仅用于匿名呼叫的场景

```
/**
 用户执行匿名登录入会

 @param location_server 定位服务器
 @param port 端口号
 @param conference_number 会议号码
 @param display_name 用户在会议中的名称
 @param password 会议密码
 @return 返回 SDK 处理结果
 */
evengine->joinConferenceWithLocation(location_server, port, conference_number,
 display_name, password);
```

窗口设置

> 添加图片说明

EVSDK视频窗口分为三类： 远端主流窗口，本地视频窗口，内容流窗口 (辅流或双流)

1. 远端窗口设置

> 如上图所示，画面中显示了接收到的远端窗口视频，每一个格子就是一路远端窗口，上图中有九个远端窗口，UI需要设置一个窗口列表给SDK，然后由SDK负责把接收到的视频流解码渲染到窗口上。UI首先需要设置最大要接收的远端窗口数量，然后再设置这些窗口句柄。

```
/**
 设置远端窗口

 @param id 远端窗口句柄数组
 @param size 远端窗口数量
 @return 返回结果
 */
int const maxRxCount = 16;
evengine->setMaxRecvVideo(maxRxCount);

int setRemoteVideoWindow(void * id[], unsigned int size);

//示例代码(remoteArr 里的元素为view对象)
void * remoteArr[maxRxCount];
evengine->setRemoteVideoWindow(remoteArr,maxRxCount);
```

2. 本地窗口设置

> 本地窗口是指本地的摄像头采集到的视频图像。

```
/**
 设置本地窗口

 @param id 本地窗口句柄
 @return 返回结果
 */
int setLocalVideoWindow(void * id)
//示例代码
evengine->setLocalVideoWindow(view);
```

3. 内容窗口设置

> 内容窗口是指别的与会者发送的内容共享，设置内容窗口句柄，SDK会把接收到的内容流画到这个窗口上。

```
/**
 设置内容窗口

 @param id 内容窗口句柄
 @return 返回结果
 */
int setRemoteContentWindow(void * id)
//示例代码
evengine->setRemoteContentWindow(view);
```

实现视频通话

视频通话主要分为两种，一种是匿名入会（参考登录模块的匿名登录入会），一种是用户登录后入会

- 设置用户入会后的头像以及视频背景图片（关闭摄像头所展示的图片）

```
/**
 设置会议中用户的头像以及背景图片

 @param background_file_path 背景图片
 @param user_image_path 用户图片
 @return
 */
int setUserImage(const char * background_file_path, const char * user_image_path)
//示例
const char *bgimagepath = "背景图片路径";
const char *userimagepath = "头像图片路径";
evengine->setUserImage(bgimagepath, userimagepath);
```

- 用户登录后主动入会

```
/**
 用户加入会议

 @param conference_number 会议号码
 @param display_name 会议中显示的名称
 @param password 会议密码
 @return
 */
int joinConference(const char * conference_number, const char * display_name,
const char * password);
//示例 注：会议如果没有密码可以不用给SDK传值，不传display_name则SDK会取用户名称当作会议名称
evengine->joinConference("会议号码", "会议中显示的名称", "会议密码");
```

- 当调用了入会的 API 后，UI 会收到 SDK 回调：入会成功、入会失败。需要做出处理

```
//入会失败会回调
void onCallEnd(EVCallInfo & info);
//入会成功会回调
void onCallConnected(EVCallInfo & info);
```

分屏模式

分屏模式指的是应用的 UI 如何为接收到的远端视频做布局展示。典型分屏样式有：主讲视图 (即一大多小布局) 和画廊视图 (平均大小布局)。当布局样式为主讲视图时，一般希望大窗口显示的视频流分辨率比较其他远端视频流更高，而当布局样式为画廊视图时，一般希望每个视频流的分辨率尽量一致。因此，在 UI 需要改变布局样式时，需要通知 SDK，这样 SDK 可以根据需要向平台发出相应请求，告知平台 UI 希望呈现的分屏布局样式，从而平台可以尽量提供满足 app 需要的视频流。

另外，当平台侧通过会议控制，对分屏进行改变时，SDK 也会接收到变化通知，此时 SDK 通知 UI 分屏模式发生了何种变化，UI 应该根据变化的样式进行对应的布局改变。

相关细节可参考 API 的布局类型、分屏样式请求和布局变化事件回调部分。

- 主讲视图 (一大多小布局) 实现示例

主讲视图模式

```
/**
 设置视频模式（主讲模式、画廊模式）

 @param layout EVLayoutRequest对象
 @return
 */
int setLayout(EVLayoutRequest & layout);
//示例代码里主讲视图模式下 主视频窗口可以显示为1x5的窗口模式
EVLayoutRequest *layout = new EVLayoutRequest();
layout.mode = EVLayoutSpeakerMode;
evengine->setLayout(layout);
```

- 画廊视图 (平均大小布局) 实现示例

画廊视图模式

```
/**
 设置视频模式（主讲模式、画廊模式）

 @param layout EVLayoutRequest对象
 @return 返回结果(UI可不作处理)
 */
int setLayout(EVLayoutRequest & layout);
//示例代码里画廊视图模式下 主视频窗口可以显示为1x1、1x2、2x2、2x3、3x3的窗口模式（可根据需要调整窗口位置）
EVLayoutRequest *layout = new EVLayoutRequest();
layout.mode = EVLayoutGalleryMode;
evengine->setLayout(layout);
```

静音和解除静音

执行本地麦克风开启或者关闭

```
/**
 是否启用麦克风

 @param enable true为开启，false为关闭
 @return
 */

//示例代码
1.关闭麦克风
evengine->enableMic(false);
2.开启麦克风
evengine->enableMic(true);
```

关闭或打开本地视频

执行本地摄像头开启或者关闭

```
/**
 是否启用摄像头

 @param enable true为开启，false为关闭
 @return
 */
int enableCamera(bool enable)
 //示例代码
 1.关闭摄像头
 evengine->enableCamera(false);
 2.开启摄像头
 evengine->enableCamera(true);
```

接收和显示共享内容

收到共享，发送共享或者共享结束， SDK 都会通过回调的形式告诉UI。

```
/**
 接收到内容分享或者内容分享结束的回调

 @param info EVContentInfo对象
 */
void onContent(EVContentInfo & info);
//示例代码
void onContent(EVContentInfo & info)
{
   if (info.enabled)
   {
       if (info.type == EVStreamContent || info.type == EVStreamWhiteBoard)
       {
           //收 到分享（如果设置了内容窗口那么应该在此处显示出你的内容窗口，如果没有设置内容窗口SDK
将会弹出一个窗口来显示分享内容）
```

```
        }
    }
    else
    {
            //分享结束(应该在此处隐藏或者关闭你的分享窗口)
    }
}
```

发送共享内容

> 发送共享主要需要以下步骤：
>
> 1. 首先通过getContentSourceList获取可以共享的内容源列表供用户选择。
> 2. 然后通过setLocalContentSource将选择的内容源设置给SDK。
> 3. UI 之后调用sendContent来请求发送内容流视频，如果平台协商一致同意当前终端发送内容源，会收到onContent回调。

```
evengine->sendContent();
```

接收到分屏变化通知时应如何处理

```
void onLayoutIndication(EVLayoutIndication & layout) {
    //当参会人数发生改变时，比如有人参会，有人离开会议；或者speaker布局下，主讲人发生改变时，会得到这个通知，此时，UI需要重新布局所有窗口。
    //EV_LAYOUT_MODE mode; 当前应该展示的布局模式。
    //EV_LAYOUT_MODE setting_mode;服务器当前使用的布局模式。
    //EV_LAYOUT_TYPE type; 当前应该展示的布局具体类型。
    //bool mode_settable;  服务器的设置，用来通知客户端是否可以改变布局模式。
    //std::string speaker_name; 主讲人名字
    //int speaker_index; 主讲人所在窗口的下标
    //unsigned int sites_size; 一共有多少个可展示窗口
   // EVSite sites[EV_LAYOUT_SIZE]; 可展示窗口的数组。
}

void onLayoutSiteIndication(EVSite & site) {
    //当前展示的窗口属性发生变化时会得到这个通知，比如某个参会人名字或audio mute状态发生改变，通过device_id可以查找是哪个参会人发生了改变。UI需要更新相应的窗口状态。
}

void onLayoutSpeakerIndication(EVLayoutSpeakerIndication & speaker) {
    //在gallery布局样式下，如果主讲人发生变化，会得到这个通知，客户端可以在主讲人所在的view上更新样式，比如加黄边框。speaker_index对应于最近一次收到的onLayoutIndication中的可展示窗口数组的下标。
}
```

结束呼叫

在需要结束呼叫的时候调用 SDK 的离会 API

```
个人离开会议，但会议还在继续。
int leaveConference();
//示例代码
evengine->leaveConference();
挂断会议，会议中止，所有人都离开会议。只有会议主持人能有挂断会议的权限。
int terminateConference();

//退出成功后，会收到sdk的回调
void onCallEnd(EVCallInfo & info);
```

API 参考

事件回调

`class EV_CLASS_API IEVEventHandler` 类提供了 SDK 回调处理方法的定义，使用者需要实现该 Handler，并在 SDK 初始化阶段将 Handler 设置给 SDK，SDK 在发生事件时通过回调 Handler 的相应方法通知 app UI。

```
class EV_CLASS_API IEVEventHandler {
public:
    ......
};
```

> 以下各节介绍每一个 Handler 方法

登录和注册状态变化

```
virtual void onLoginSucceed(EVUserInfo & user) {
  (void)user;
}
```

```
virtual void onRegister(bool registered) {
  (void)registered;
}

virtual void onRegisterInfo(EVRegisterInfo & info) {
        (void)info;
}

virtual void onCallState(EVCallInfo & info) {
        (void)info;
}
```

呼叫状态变化

```
入会成功。
virtual void onCallConnected(EVCallInfo & info) {
  (void)info;
}
电话接通成功。
virtual void onCallPeerConnected(EVCallInfo & info) {
  (void)info;
}
会议结束。
virtual void onCallEnd(EVCallInfo & info) {
  (void)info;
}
```

分屏变化

```
分屏发生变化。
virtual void onLayoutIndication(EVLayoutIndication & layout) {
  (void)layout;
}

某一个分屏的内容发生变化，比如麦克风状态，分屏的显示名声发生变化。
virtual void onLayoutSiteIndication(EVSite & site) {
  (void)site;
}

会议中主讲人发生变化。
virtual void onLayoutSpeakerIndication(EVLayoutSpeakerIndication & speaker) {
  (void)speaker;
}

virtual void onSiteCellInfoIndication(EVCellSiteInfoIndication & indic) {
  (void)indic;
}
```

```
virtual void onNumberOfStandingIndication(EVNumberOfStandingIndication & nos) {
        (void)nos;
}
```

辅流状态变化

```
virtual void onContent(EVContentInfo & info) {
  (void)info;
}

virtual void onWhiteBoardIndication(EVWhiteBoardInfo & info) {
        (void)info;
}
```

发送双流时，当发送的窗口被遮挡或者最小化的时候，会收到此回调。

```
virtual void onContentSourceStatusChange( EVVideoSize size, int x, int y, bool
isOverlaidorMinimized, bool isAlive){
  (void)size;
  (void) x;
  (void) y;
  (void) isOverlaidorMinimized;
  (void) isAlive;
}
```

错误和异常

```
virtual void onError(EVError & err) {
  (void)err;
}

virtual void onWarn(EVWarn & warn) {
  (void)warn;
}
```

其他事件

```
// 检测到静音时讲话
virtual void onMuteSpeakingDetected() {
}

virtual void onCallLogUpdated(EVCallLog & call_log) {
  (void)call_log;
}

// 被平台静音通知
```

```
virtual void onMicMutedShow(int mic_muted) {
  (void)mic_muted;
}

// 摄像头被开启/关闭通知
virtual void onVideoMutedShow(int video_muted) {
  (void)video_muted;
}

virtual void onJoinConferenceIndication(EVCallInfo & info) {
  (void)info;
}

// 平台下发会议录制状态通知
virtual void onRecordingIndication(EVRecordingInfo & state) {
  (void)state;
}

// 平台下发字幕内容通知
virtual void onMessageOverlay(EVMessageOverlay & msg) {
  (void)msg;
}

// 与会者人数变化通知
virtual void onParticipant(int number) {
  (void)number;
}

//用户头像下载完成通知。
virtual void onDownloadUserImageComplete(const char * path) {
  (void)path;
}

//用户头像上传成功通知。
virtual void onUploadUserImageComplete(const char * path) {
  (void)path;
}

// 网络状态变化通知
virtual void onNetworkState(bool reachable) {
  (void)reachable;
}

// 网络质量变化通知
virtual void onNetworkQuality(float quality_rating) {
  (void)quality_rating;
}

// 当前呼入排队等待人数通知。
virtual void onWaitQueue(int queue_size) {
        (void)queue_size;
}
```

```
// 把音频数据返给上层
virtual void onAudioData(int sample_rate, void * data, int len) {
        (void)sample_rate;
        (void)data;
        (void)len;
}

//通知用户上传日志是否成功。成功返回100，  不成功返回-1.
virtual void onUploadFeedback(int number) {
        (void)number;
}

//设备添加
virtual void onDeviceAdded(EVDevice & device) {
        (void) device;
}

//设备移除
virtual void onDeviceRemoved(EVDevice & device) {
        (void) device;
}

//当前使用设备变更通知
virtual void onCurrentDeviceChanged(EVDevice & device) {
        (void) device;
}

//当页数发生变化时，通知UI。
virtual void onPageInfo(EVPageInfo & info) {
    (void)info;
}
```

接口方法

初始化和释放

- ev::engine::IEVEngine* createEVEngine();

  > 创建 EVEngine 实例
  >
  > 参数： 无
  >
  > 返回：EVEngine实例，全局只有一个，用于后续接口调用。

- void deleteEVEngine(ev::engine::IEVEngine* engine);

  > 销毁 EVEngine 实例
  >
  > 参数： 无

> 返回：无

- int initialize(const char *config_path, const char * config_file_name);

> 初始化 SDK
>
> 参数：config_path`- SDK config 文件路径
>
> config_file_name` - SDK config 文件名
>
> 返回：成功，返回EV_OK，
>
> 失败，返回EV_NG。

- int release();

> 释放SDK
>
> 参数：无
>
> 返回：成功，返回EV_OK，
>
> 失败，返回EV_NG。

- int setRootCA(const char * root_ca_path);

> 设置 rootca 文件路径。
>
> 参数：root_ca_path - rootca的路径。
>
> 返回：成功，返回EV_OK，
>
> 失败，返回EV_NG。

- int setUserImage(const char * background_file_path, const char * user_image_path);

> 设置用户头像和入会后显示的背景图像，在用户没有登录时，会显示这个设置的头像，登录后，头像是从服务器取到的。
>
> 参数：background_file_path - 用来设置用户入会后显示的的背景图像
>
> user_image_path - 用来设置没有登录时，显示的头像
>
> 返回：成功，返回EV_OK，
>
> 失败，返回EV_NG。

- int enableSecure(bool enable);

> 使用SSL加密网络传输
>
> 参数： true - 使用SSL加密网络传输，
>
> false：不使用SSL加密网络传输。
>
> 返回：成功，返回EV_OK，
>
> 失败，返回EV_NG。

- int setUserAgent(const char * company, const char * version);

  > 设置软件提供商信息
  >
  > 参数：company - 公司名
  >
  > version - 软件版本
  >
  > 返回：成功，返回EV_OK，
  >
  > 失败，返回EV_NG。

- int registerEventHandler(IEVEventHandler * handler);

  > 注册事件接收器
  >
  > 参数：handler - 事件处理实例，是继承类ev::engine::IEVEventHandler的示例变量。
  >
  > 返回：成功，返回EV_OK，
  >
  > 失败，返回EV_NG。

- int unregisterEventHandler(IEVEventHandler * handler);

  > 取消注册事件接收器
  >
  > 参数：handler - 事件处理实例，是继承类ev::engine::IEVEventHandler的示例变量。
  >
  > 返回：成功，返回EV_OK，
  >
  > 失败，返回EV_NG。

日志

- void setLog(EV_LOG_LEVEL level, const char * log_path, const char * log_file_name, unsigned int max_file_size);

  > 设置log输出级别，路径和最大文件大小
  >
  > 参数： level - 设置日志输出级别，比这个级别高的日志都能输出。
  >
  > log_path - 日志输出文件路径
  >
  > log_file_name - 日志输出文件名字
  >
  > max_file_size - 日志文件的最大空间，通常SDK会循环写入三个文件，每个文件的最大值是max_file_size 。

- void setConsoleLog(EV_LOG_LEVEL level);

  > 设置输出日志到控制台
  >
  > 参数： level - 设置日志输出级别，比这个级别高的日志都能输出。默认情况下控制台输出是打开的。如果想关闭控制台输出，需要设置EV_LOG_LEVEL_FATAL。

- void enableLog(bool enable);

  > 开启日志
  >
  > 参数：true - 开始采集日志
  >
  > false - 不采集日志

- std::string compressLog();

  > 压缩日志文件
  >
  > 返回：压缩后的文件名字，文件存储在日志同一个目录下。

- int uploadFeedbackFilesWithAddress(std::vector **std::string** filePath, std::string contact, std::string description, std::string address) ;

  > 上传日志到后台服务器。
  >
  > 参数：filePath - 日志文件路径
  >
  > contact - 上传日志的用户联系方式
  >
  > description - 日志的简单描述
  >
  > address - 日志上传到的服务器地址
  >
  > 返回：成功，返回EV_OK，
  >
  > 失败，返回EV_NG。

用户登录登出

- int loginWithLocation(const char * location_server, unsigned int port, const char * username, const char * encrypted_password)

  > 登录
  >
  > 参数： location_server - 定位服务器地址
  >
  > port - 定位服务器端口号
  >
  > username - 用户名
  >
  > encrypted_password - 密码
  >
  > 返回：成功，返回EV_OK，
  >
  > 失败，返回EV_NG。

- int logout();

  > 退出登录
  >
  > 返回：成功，返回EV_OK，

> 失败，返回EV_NG。

- std::string EVEngine::encryptPassword(EV_ENCRYPT_TYPE type, const char * password, const char* akey);

> 加密方法
>
> 参数： type - 加密类型，目前支持的加密方法是SHA，AES，DES。
>
> password - 要加密的字串
>
> akey - 加密的密钥
>
> 返回： 加密后的字串，base64编码。

- std::string descryptPassword(EV_DESCRYPT_TYPE type, const char * password, const char* akey, const char* iv);

> 解密方法
>
> 参数： type - 解密类型
>
> password - 要解密的字串
>
> akey - 解密的密钥
>
> iv - 解密的iv
>
> 返回：解密后的字串。

- int downloadUserImage(const char * path);

> 下载用户头像到本地
>
> 参数： path - 下载后的文件保存地址
>
> 返回：成功，返回EV_OK，
>
> 失败，返回EV_NG。

- int uploadUserImage(const char * path);

> 上传用户头像到服务器
>
> 参数： path - 上传的文件保存地址
>
> 返回：成功，返回EV_OK，
>
> 失败，返回EV_NG。

- int changePassword(const char * encrypted_oldpassword, const char *encrypted_newpassword);

> 修改密码
>
> 参数： encrypted_oldpassword - 原始密码
>
> encrypted_newpassword - 修改后的密码
>
> 返回：成功，返回EV_OK，

> 失败，返回EV_NG。

- int changeDisplayName(const char * display_name);

> 修改显示名字
> 参数：display_name - 修改后的显示名字
> 返回：成功，返回EV_OK，
> 失败，返回EV_NG。

- int getUserInfo(EVUserInfo & userinfo);

> 获取用户信息
> 参数：userinfo - 输出参数，用于获取用户信息。

- std::string getDisplayName();

> 获取用户显示名字
> 返回：获取显示名字

会议能力设置

- int setMaxRecvVideo(unsigned int num)

> 设置可以接收的最大入会视频流数量，需要在会议开始前设置才能生效。
> 参数：num - 最大视频流的数量
> 返回：成功，返回EV_OK，
> 失败，返回EV_NG。

- int enableContent(bool enable)

> 设置会议中是否允许发送或接收双流，需要在会议开始前设置才能生效。
> 参数：true- 设置可以发送双流
> false - 设置不发送双流
> 返回：成功，返回EV_OK，
> 失败，返回EV_NG。

- bool contentEnabled()

> 查询会议中是否有发送或接收双流的能力

- int enableContentHighFPS(bool enable)

> 设置会议中发送是否发送高清双流，TRUE表示帧率优先，更加流畅，FALSE表示清晰度优先，画质更好，需要在会议开始前设置才能生效。

- bool contentHighFPSEnabled()

> 获取当前设置的是帧率优先还是清晰度优先

- enableRelayStreamDataCb(EV_STREAM_TYPE type, bool enable)

> 是否开启回调onAudioData和onVideoPreviewFrame

- int enableAdaptiveResolution(bool enable)

> 设置是否根据CPU的使用情况，来动态调整分辨率，以提高视频会议中电脑的性能。

- bool adaptiveResolutionEnabled()

> 获取是否动态调整分辨率的设置

- bool highFPSEnabled();

> 获取当前是否是帧率优先

- int enableHighFPS(bool enable);

> 设置视频会议中帧率优先

- int enableHD(bool enable);

> 设置视频会议中高清优先

- bool HDEnabled();

> 获取当前设置是否是高清优先

- int setBandwidth(unsigned int kbps);

> 设置视频会议的带宽

- unsigned int getBandwidth();

> 获取设置的视频会议带宽

会议通讯

- int joinConference(const char * conference_number, const char * display_name, const char * password);

  > 加入会议

- int joinConference(const char * conference_number, const char * display_name, const char * password, EV_SVC_CALL_TYPE type);

  > 加入会议

- int joinConference(const char * conference_name, V_SVC_CONFERENCE_NAME_TYPE name_type, const char * display_name, const char * password, EV_SVC_CALL_TYPE type);

  > 加入会议

- int joinConferenceWithLocation(const char * location_server, unsigned int port, const char * conference_number, const char * display_name, const char * conf_password);

  > 匿名入会, 用户可以不登录，输入会议号码直接入会。如果会议设置了密码，则conf_password设置成会议密码，否则，则置成空。

- int joinConferenceWithLocation(const char * location_server, unsigned int port, const char * conference_name, EV_SVC_CONFERENCE_NAME_TYPE name_type, const char * display_name, const char * conf_password);

  > 匿名入会, 用户可以不登录，输入会议号码直接入会。如果会议设置了密码，则conf_password设置成会议密码。当name_type是EV_SVC_CONFERENCE_NAME_ID时，conference_name需要设置成会议号码；当name_type是EV_SVC_CONFERENCE_NAME_ALIAS时，conference_name设置成别名。

- int leaveConference()；

  > 离开会议

- int terminateConference();

  > 结束会议

- int setInConfDisplayName(const char * display_name, int len)

  > 会议中设置显示的会场名称

会议操作

- bool cameraEnabled();

> 获取会议中摄像头打开/关闭状态

- int enableCamera(bool enable)

> 打开/关闭摄像头。enable 为 true 时，主流视频为正常的输入采集视频，为 false 时主流视频改为视频被屏蔽的静止视频，帧率为 1 fps。

- int switchCamera()

> 循环设置getDevices得到的摄像头列表中的设备为视频输入设备。

- bool micEnabled()

> 获取麦克风打开/关闭状态，true为打开，false为静音状态。

- int enableMic(bool enable)

> 打开/关闭麦克风，enable为true时，表示打开麦克风；enable为false时，表示静音状态。

- bool remoteMuted()

> 获取会控服务器的静音状态

- int requestRemoteUnmute(bool val)

> 请求会控服务器解除静音

- int getCallInfo(EVCallInfo & call_info)

> 获取呼叫信息

- int setVideoActive(int active)

> 切换视频模式和语音模式，如果是true，表示当前是视频模式，如果是false，表示当前是语音模式。

- int videoActive()

> 获取当前是视频模式还是语音模式，如果是true，表示当前是视频模式，如果是false，表示当前是语音模式。

- bool isConferenceHoster()

> 获取是否是会议主持人， 如果是会议主持人，会有结束会议的权限。

会议视频流窗口设置

会议中，有两类窗口，一类是本地视频窗口，用于显示本地摄像头采集的视频内容；另一类是远端视频窗口，用于显示从服务器接收到的视频流内容。由于会议中，有多个人入会，所以需要一个远端视频窗口数组来展示视频内容，每一个远端窗口句柄会显示一个视频流，也就是一个与会者的视频。视频流又分为主流和内容流。主流主要是采集的人物的视频流，内容流主要是采集用户共享的桌面，应用或者白板的视频流。

- int setLocalVideoWindow(void * id)

  > 设置本地视频窗口句柄

- int setRemoteContentWindow(void * id)

  > 设置远端内容窗口句柄

- int setRemoteVideoWindow(void * id[], unsigned int size)

  > 设置远端视频窗口句柄

- int setLocalContentWindow(void * id, EV_CONTENT_MODE mode)

  > 设置发送内容窗口句柄

- void* getLocalVideoWindow()

  > 获取本地视频窗口句柄

- void* getRemoteVideoWindow(void * id[], unsigned int size)

  > 获取远端视频窗口句柄

- virtual void * getRemoteContentWindow()

  > 获取接收远端内容窗口句柄

- void* getLocalContentWindow()

  > 获取本地发送视频窗口句柄

- int repaintVideo()

  > 通知sdk重绘远端视频内容

会议视频流布局

- int setLayoutCapacity(EV_LAYOUT_MODE mode, EV_LAYOUT_TYPE types[], unsigned int size)

  > 设置分屏的能力集，可以分别设置主讲人模式和画廊模式下，所支持的分屏样式，平台会根据当前带宽，从这个集合中，协商出一个分屏样式发送给UI。

- int setLayout(EVLayoutRequest & layout)

  > 设置布局样式，用户可以设置主讲人模式或者画廊模式下所需要的分屏样式。会议过程中，可以动态切换布局样式.

- int setRecvVideo(unsigned int num, EVVideoSize max_resolution)

  > 在会议中临时设置收到的People内容源的最大路数和最大分辨率，如果需要恢复之前的设置,，则需要将参数设置成0。

- int setPageSetting(EVPageSetting * setting)

  > 翻页设置

- int getPageInfo(EVPageInfo * info)

  > 获取翻页状态

- int setPage(int page_number)

  > 翻到指定页数

在会议开始之前，首先要用setMaxRecvVideo设置最大接收的远端视频流，会议中，可以用setRecvVideo来动态调整想要接收的最大视频流，服务器和终端都可以设置layout, 当前layout是以最后设置的为准。

会议内容流（双流）

- int getContentInfo(EVContentInfo & content_info)

  > 获取辅流信息, 在会议中, 收到内容流时, SDK首先会发送回调onContent来通知UI当前接收到一路内容流, UI也可以通过这个接口来获取接收到的内容流信息。

- virtual int sendContent()

  > 开始发送内容流的请求, 在调用之前, 需要先设置要发送的内容源, 可以通过接口setLocalContentSource设置内容源, 也可以通过setLocalContentWindow来设置内容源, sendContent是用来请求我是否能发送内容流, 如果服务器同意发送了, 会给UI回调onContent, 并且status的值是EV_CONTENT_GRANTED, 如果服务器拒绝UI的双流请求, 则收到的回调status的值是EV_CONTENT_DENIED。

- virtual int stopContent()

  > 停止发送双流。

- virtual int enableContentAudio(bool enable)

  > 发送双流时同时发送声音。

- virtual bool contentAudioEnabled()

  > 获取是否发送双流声音的设置。

- virtual int setLocalContentSource(EVContentSourceInfo & info) ;

  > 发送用户选择的内容源窗口内容, 双流源可以从getContentSourceList获取。
  >
  > keepRatio表示需要保留原始视频比例，如果为true，则保留视频原有比例，添加黑边，如果为false，则有可能拉伸图像视频。
  >
  > captureTransparentWnd表示是否抓取透明窗口，如果有透明窗口，则需要设置这个属性为true。
  >
  > 可以设置发送用户图片缓存模式(EV_CONTENT_USER_SHARED_MODE)，此时需要设置图像缓存sharedImage，captureSize和captureStride。

- std::vector getContentSourceList()

  > 获取当前可发送的内容源列表。

- getContentSourceBitmap(EVContentSourceInfo & info, EV_BITMAP_TYPE bitmapType, int maxWidth, int maxHeight, EVBitmap & bitmap)

  > 获取铺流源的窗口缩略图，图标。UI可以展示这个缩略图来让用户选择。

- void releaseContentSourceBitmap

  > 释放缩略图，图标等内存资源

设备操作

- virtual std::vector getDevices(EV_DEVICE_TYPE type)

  > 获取当前所有可以连接的设备列表。

- virtual void setDevice(EV_DEVICE_TYPE type, unsigned int id)

  > 设置当前选中的设备。

- virtual EVDevice getDevice(EV_DEVICE_TYPE type)

  > 获取当前已经选中的设备。

  `EV_DEVICE_TYPE type` - 设备类型

- virtual int setCameraCapture(EVCameraCaptureInfo & info)

  > 设置摄像头参数

- virtual int getCameraCapture(EVCameraCaptureInfo & info)

  > 获取摄像头参数

- virtual std::vector getCameraCaptureList()

  > 获取已连接的摄像头参数集合的列表

- int setVolume(EV_DEVICE_TYPE type, EVVolume & vol)

  > 设置音量，取值范围0.0 ~ 3.0f。

- EVVolume getVolume(EV_DEVICE_TYPE type)

  > 获取当前设备音量

- int EVEngine::setAudioFile(const char * play_file)

  > 设置播放声音文件

- int startAudioPlay()

  > 开始播放声音

- stopAudioPlay()

  > 停止播放声音

美颜功能

- virtual int setEnhanceBrightness(int brightness) ;

  > 设置提亮效果, 取值范围是0-40。

- virtual int getEnhanceBrightness() ;

> 获取提亮设置数值。

- virtual int setBeautifyFace(int beautify) ;

> 设置是否打开美颜. 美颜取值是0-10。

- virtual int getBeautifyFace() ;

> 获取美颜设置。

网络状态统计

- float getNetworkQuality()

> 获取网络信号强度

- getStats(EVStats & stats)

> 获取视频流统计信息

设置和其它

- virtual int setPort(EV_STREAM_TYPE stream, int port)

> 设置各种媒体流传输端口。

- virtual int getPort(EV_STREAM_TYPE stream)；

> 获取媒体流传输端口。

- int setRenderType(EV_RENDER_TYPE type)

> 获取当前渲染的模式

附录

错误码表

服务器错误码

错误号	错误	描述
1000	EV_SERVER_API_VERSION_NOT_SUPPORTED	系统不支持您使用的API版本
1001	EV_SERVER_INVALID_TOKEN	无效的token
1002	EV_SERVER_INVALID_PARAMETER	无效的参数
1003	EV_SERVER_INVALID_DEVICESN	参数deviceSN为空或者不合法
1004	EV_SERVER_INVALID_MEDIA_TYPE	无效的媒体类型，只支持”application/json”
1005	EV_SERVER_PERMISSION_DENIED	您不具备执行此操作的权限
1006	EV_SERVER_WRONG_FIELD_NAME	JSON串内字段名拼写错误
1007	EV_SERVER_INTERNAL_SYSTEM_ERROR	系统内部错误
1008	EV_SERVER_OPERATION_FAILED	操作失败！
1009	EV_SERVER_GET_FAILED	获取失败！
1010	EV_SERVER_NOT_SUPPORTED	不支持！
1011	EV_SERVER_REDIS_LOCK_TIMEOUT	请重试
1019	EV_SERVER_LOCAL_ZONE_STOPPED	本地可用区已停止。
1100	EV_SERVER_INVALID_USER_NAME_PASSWORD	无效的用户名或密码
1101	EV_SERVER_LOGIN_FAILED_MORE_THAN_5_TIMES	登录失败超过5次，账号会被锁定，您还有{0}次机会
1102	EV_SERVER_ACCOUNT_TEMPORARILY_LOCKED	您的账户已被临时锁定，请稍后再试或直接联系管理员解锁。
1103	EV_SERVER_ACCOUNT_DISABLED	您的账户已被停用，请联系管理员。
1104	EV_SERVER_NO_USERNAME	没有找到这个用户名
1105	EV_SERVER_EMAIL_MISMATCH	邮箱和用户名不匹
1106	EV_SERVER_COMPANY_ADMINISTRATOR_NOT_IN_ANY_COMPANY	该公司管理员未分配到任何公司
1200	EV_SERVER_FILE_UPLOAD_FAILED	文件上传失败
1201	EV_SERVER_INVALID_LICENSE	无效许可证
1202	EV_SERVER_INVALID_IMPORT_USER_FILE	无效的导入用户文件
1300	EV_SERVER_INVALID_TIME_SERVICE_ADDRESS	无效的时间服务地址
1301	EV_SERVER_FAILED_UPDATE_SYSTEM_PROPERTIES	系统参数修改失败
1400	EV_SERVER_CONF_NOT_EXISTS	会议不存在
1401	EV_SERVER_NUMERICID_CONFLICTS	该会议号码已被占用
1402	EV_SERVER_CONF_UPDATING_IN_PROGRESS	正在修改会议
1403	EV_SERVER_CONF_DELETING_IN_PROGRESS	正在删除会议
1404	EV_SERVER_CONF_TERMINATING_IN_PROGRESS	正在结束会议

1405	EV_SERVER_CONF_LAUNCHING_IN_PROGRESS	正在开启会议
1406	EV_SERVER_CONF_NOT_IN_APPROVED_STATUS	会议不处于预约状态
1407	EV_SERVER_CONF_NUMERICID_ONGOING	会议已开启
1409	EV_SERVER_CONF_NOT_APPROVED_OR_ONGOING	会议已失效
1410	EV_SERVER_PARTICIPANT_NOT_EXISTS_IN_CONF	该终端未加入会议
1412	EV_SERVER_NUMERICID_ALREADY_IN_USE	该会议号码已被占用！
1415	EV_SERVER_INVALID_CONF_TIME	无效的会议时间
1418	EV_SERVER_INVALID_CONF_ID	无效的会议ID
1421	EV_SERVER_NOT_FOUND_SUITABLE_MRU	未找到合适的MRU.
1422	EV_SERVER_NOT_FOUND_SUITABLE_GATEWAY	未找到合适的Gateway
1424	EV_SERVER_FAILED_TO_CONNECT_MRU	连接MRU失败
1427	EV_SERVER_NOT_ALLOW_DUPLICATED_NAME	不允许重名
1430	EV_SERVER_NOT_FOUND_CONF_IN_REDIS	数据没有找到当前会议
1431	EV_SERVER_NOT_IN_LECTURER_MODE	当前不是主会场模式
1433	EV_SERVER_FAILED_TO_MUTE_ALL_PARTICIPANTS	全部静音失败
1436	EV_SERVER_FAILED_TO_CONNECT_PARTICIPANT	连接与会者失败
1439	EV_SERVER_FAILED_TO_DISCONNECT_PARTICIPANT	挂断与会者失败
1442	EV_SERVER_FAILED_TO_CHANGE_LAYOUT	分屏设置失败
1445	EV_SERVER_FAILED_TO_SET_SUBTITLE	字幕设置失败
1448	EV_SERVER_FAILED_TO_MUTE_PARTICIPANT_AUDIO	对与会者静音失败
1451	EV_SERVER_FAILED_TO_DELETE_PARTICIPANT	删除与会者失败
1454	EV_SERVER_FAILED_TO_INVITE_AVC_ENDPOINT	邀请AVC终端失败
1455	EV_SERVER_FAILED_TO_INVITE_SVC_ENDPOINTS	邀请SVC终端失败
1456	EV_SERVER_CONF_ROOM_COMPLETELY_FULL	会议室已满
1457	EV_SERVER_TIMEOUT_TO_GENERATE_NUMERICID	产生云会议室号失败，已超时
1460	EV_SERVER_NOT_FOUND_PROFILE_NAMED_SVC	未找到名为SVC的会议模板
1463	EV_SERVER_FAILED_TO_PROLONG_CONF	会议延时失败
1500	EV_SERVER_INVALID_MEETING_CONTROL_REQUEST	无效的会议控制请求
1600	EV_SERVER_NAME_IN_USE	终端名字以被占用
1601	EV_SERVER_EMPTY_ENDPOINT_NAME	终端名称不能为空
1602	EV_SERVER_EMPTY_ENDPOINT_CALL_MODE	终端呼叫模式不能为空
1603	EV_SERVER_EMPTY_ENDPOINT_SIP_USERNAME	终端的SIP号码不能为空
1604	EV_SERVER_EMPTY_ENDPOINT_SIP_PASSWORD	终端的SIP密码不能为空

1605	EV_SERVER_EMPTY_ENDPOINT_ADDRESS	终端的IP地址不能为空
1606	EV_SERVER_INVALID_SIP_USERNAME	无效的SIP号码
1607	EV_SERVER_INVALID_IP_ADDRESS	无效的IP地址
1608	EV_SERVER_ENDPOINT_NOT_EXIST	指定的终端不存在
1609	EV_SERVER_E164_IN_USE	该E164号码已被占用
1610	EV_SERVER_ENDPOINT_DEVICE_SN_EXIST	该序列号已被占用，请重新输入
1611	EV_SERVER_SIP_USERNAME_REGISTERED	该SIP号码已被占用
1612	EV_SERVER_ENDPOINT_E164_INVALID	无效的E164号
1613	EV_SERVER_NOT_FOUND_ENDPOINT_DEVICE_SN	该终端不存在
1614	EV_SERVER_NOT_FOUND_ENDPOINT_PROVISION_TEMPLATE	终端配置文件不存在
1615	EV_SERVER_DEVICE_SN_EXISTS	这些设备序列号已被分配了
1700	EV_SERVER_CAN_NOT_DELETE_USER_IN_RESERVED_MEETING	不能删除正在会议中的用户
1701	EV_SERVER_EMPTY_USER_PASSWORD	密码不能为空
1702	EV_SERVER_EMPTY_USERNAME	用户名不能为空
1703	EV_SERVER_EMPTY_USER_DISPLAY_NAME	姓名不能为空
1704	EV_SERVER_INVALID_USER_EMAIL	无效的邮箱地址
1705	EV_SERVER_INVALID_CELLPHONE_NUMBER	无效的手机号码
1706	EV_SERVER_ORIGINAL_PASSWORD_WRONG	原密码输入错误
1707	EV_SERVER_DUPLICATE_EMAIL_NAME	邮箱重复
1708	EV_SERVER_DUPLICATE_CELLPHONE_NUMBER	手机重复
1709	EV_SERVER_DUPLICATE_USERNAME	账号重复
1710	EV_SERVER_INVALID_CONF_ROOM_MAX_CAPACITY	无效的最大会议室容量
1711	EV_SERVER_SHOULD_ASSIGN_DEPARTMENT_TO_DEPARTMENT_ADMINISTRATOR	请为部门管理员选择所在部门
1712	EV_SERVER_EMPTY_USER_EMAIL	邮箱不能为空
1713	EV_SERVER_EMPTY_USER_CELLPHONE_NUMBER	手机号不能为空
1714	EV_SERVER_NOT_ORGANIZATION_ADMINISTRATOR	该用户不是公司管理员
1800	EV_SERVER_COMPANY_NOT_EXIST	该公司不存在
1801	EV_SERVER_SHORT_NAME_OF_COMPANY_USED	公司短名字已被占用
1802	EV_SERVER_FULL_NAME_OF_COMPANY_USED	公司全名已被占用
1803	EV_SERVER_COMPANY_NOT_EMPTY	清理该公司下用户和终端后再删除该公司
1804	EV_SERVER_EMPTY_COMPANY_SHORT_NAME	公司名称不能为空
2000	EV_SERVER_CONF_ROOM_EXPIRED	该云会议室已过期
2001	EV_SERVER_NOT_ACTIVED	没有被激活

2003	EV_SERVER_NOT_FOUND_SUITABLE_ROOM	未找到可用的云会议室
2005	EV_SERVER_NOT_FOUND_TEMPLATE_OR_ROOM	没有找到模板和房间
2006	EV_SERVER_CONF_ROOM_IN_USE	该云会议室已被使用
2009	EV_SERVER_CONF_ROOM_NUMBER_IN_USE	该云会议室号码已被占用
2012	EV_SERVER_CONF_ROOM_CAPACITY_EXCEEDS_LIMIT	会议数量超过最大容量
2015	EV_SERVER_INVALID_CONF_ROOM_CAPACITY	无效的云会议室方数
2018	EV_SERVER_INVALID_CONF_ROOM_NUMBER	无效的云会议室号码
2021	EV_SERVER_ROOM_NOT_EXISTS	该云会议室不存在
2031	EV_SERVER_ROOM_ONLY_ALLOW_OWNER_ACTIVE	只有会议主持人才能激活会议
2033	EV_SERVER_ROOM_NOT_ALLOW_ANONYMOUS_CALL	该会议不允许匿名入会
2035	EV_SERVER_TRIAL_ORG_EXPIRED	公司许可过期
2100	EV_SERVER_CAN_NOT_DELETE_DEPARTMENT_WITH_SUBORDINATE_DEPARTMENT	该部门有子部门，不能删除
2101	EV_SERVER_CAN_NOT_DELETE_DEPARTMENT_WITH_USERS_OR_ENDPOINTS	该部门内有用户或终端，不能删除
2200	EV_SERVER_INVALID_ACS_CONFIGURATION	无效的ACS配置

地址服务器错误码

错误号	错误	描述
10000	EV_LOCATE_FAILED_TO_READ_BODY	定位服务读取消息体出错
10001	EV_LOCATE_FAILED_TO_PARSE_BODY	定位服务解析消息体出错
10002	EV_LOCATE_LOCATION_TIMEOUT	定位服务定位超时
10003	EV_LOCATE_ERROR_INFO_GENERAL	定位服务通用信息错误
10004	EV_LOCATE_ERROR_INFO_BAD_FORMAT	定位服务信息格式错误
10005	EV_LOCATE_UNEXPECTED	定位服务异常
10006	EV_LOCATE_FAILED_TO_LOCATE_CLIENT	定位服务不能定位客户端
10007	EV_LOCATE_FAILED_TO_LOCATE_ZONE	定位服务不能定位可用区
10008	EV_LOCATE_NO_LOCATION_DOMAIN	地址服务没有域
10009	EV_LOCATE_ERROR_LOCATION_REQUEST	定位服务请求错误
10010	EV_LOCATE_ERROR_INFO_QUERY_ZONE_IP_IN_HIM	定位服务在HIM中查询域IP出错

会议错误码

错误	错误号	描述
11	EV_CALL_BYE_EP_NO_PACKET_RECEIVED	终端没有码流，断会
100	EV_CALL_BYE_MRU_NORMAL	MRU正常挂断会议
101	EV_CALL_BYE_MRU_OPERATOR_DISCONNECT	MRU管理员挂断终端
102	EV_CALL_BYE_MRU_NO_PACKET_RECEIVED	MRU没有收到码流，断会
1001	EV_CALL_INVALID_NUMERICID	无效的会议号码
1005	EV_CALL_INVALID_USERID	无效 的用户ID
1007	EV_CALL_INVALID_DEVICEID	无效的设备ID
1009	EV_CALL_INVALID_ENDPOINT	无效的终端
2001	EV_CALL_SERVER_UNLICENSED	服务器许可证过期
2003	EV_CALL_NOT_FOUND_SUITABLE_MRU	呼叫的云平台尚未激活
2005	EV_CALL_NEITHER_TEMPLATE_NOR_ONGOING_NOR_BINDED_ROOM	没有可用的会议端口资源
2007	EV_CALL_LOCK_TIMEOUT	呼叫超时
2009	EV_CALL_TEMPLATE_CONF_WITHOUT_CONFROOM	模板会议没有会议室
2011	EV_CALL_ROOM_EXPIRED	会议室超过最大数量
2015	EV_CALL_INVALID_PASSWORD	无效的密码
2017	EV_CALL_NO_TIME_SPACE_TO_ACTIVATE_ROOM	没有时间激活会议室
2023	EV_CALL_CONF_PORT_COUNT_USED_UP	当前入会人数已达上限
2024	EV_CALL_ORG_PORT_COUNT_USED_UP	当前与会人数已达组织端口上限
2025	EV_CALL_HAISHEN_PORT_COUNT_USED_UP	当前与会人数已达平台会议端口上限
2027	EV_CALL_HAISHEN_GATEWAY_AUDIO_PORT_COUNT_USED_UP	网关语音通话数量已达到云平台的许可上限
2029	EV_CALL_HAISHEN_GATEWAY_VIDEO_PORT_COUNT_USED_UP	网关视频通话数量已达到云平台的许可上限
2031	EV_CALL_ONLY_ROOM_OWNER_CAN_ACTIVATE_ROOM	只允许会议室拥有者激活会议室
2033	EV_CALL_NOT_ALLOW_ANONYMOUS_PARTY	该会议不允许匿名呼叫
2035	EV_CALL_TRIAL_ORG_EXPIRED	公司许可过期
2043	EV_CALL_LOCAL_ZONE_NOT_STARTED	可用区没有启动
2045	EV_CALL_LOCAL_ZONE_STOPPED	可用区停止
4057	EV_CALL_ROOM_BUSY	会议室忙碌
4064	EV_CALL_INVALID_FROM_URL	无效的URL地址