Meeing SDK(IOS)说明文档

1 集成 Meeing SDK

> 在您阅读此文档时，我们假定您已经具备了基础的 iOS 应用开发经验，并能够理解相关基础概念。

1.1 环境准备

1.1.1 请自行部署 MediaServer。或联系微议销售人员，获取微议云平台企业账号。

11.2 请准备一台 MAC 机，并安装 Xcode 6.3.1 或以上版本。

1.1.3 请到 微议官方网站 下载 Meeting SDK IOS 版。

1.2 SDK 目录讲解

1.2.1、IOS SDK 中有 2 个子文件夹:include、lib

- **Lib** 包含 sdk 的静态库文件
- **Include** 包含 sdk 的头文件。

1.2.2、主要介绍下 **include**，所有的接口都在这个文件夹中。

- **EmmKit.h** 公用的预定义；
- **MeetingSession.h** MeetingSession 类定义，提供所有的会场功能；
- **MeetingUserMgr.h** 用户管理类 MeetingUserMgr；

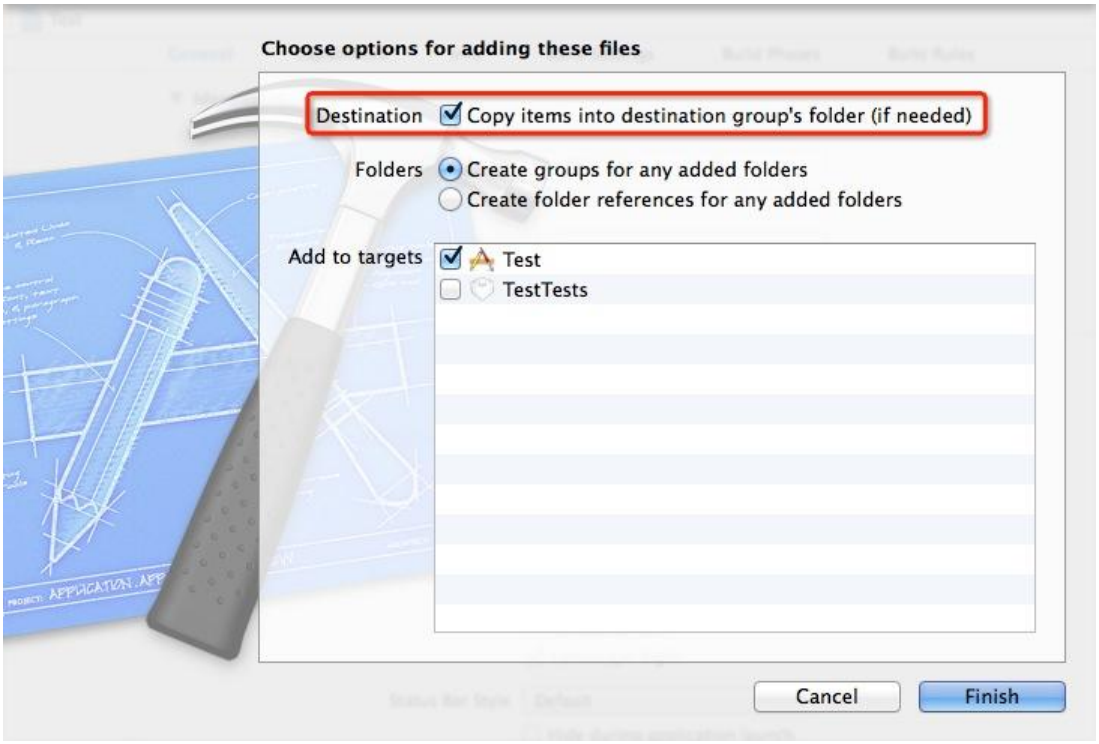
- **RtmpClientMgr.h** 这是 MeetingSession 的基类，提供基础的通讯功能。

具体接口讲解请转到 Apple Docs

1.3 配置工程

1.3.1. 导入 SDK

将下载好的 SDK 文件夹(sdk)拖入到项目中，并勾选上 Destination



1.3.2 设置工程属性

1.3.2.1 向 Build Phases -> Link Binary With Libraries 中添加 libs 下的所有.a 文件，已经下面列出的 framework：

- AudioToolbox.framework
- AVFoundation.framework
- CoreGraphics.framework
- CoreMedia.framework
- CoreVideo.framework
- FounDation.framework
- UIKit.framework
- libstdc++.6.0.9.dylib
- Libc++.1.dylib
- libz.dylib
- Libz2.dylib
- libiconv.dylib

1.3.2.2 请将 Build Settings->Apple LLVM 6.1 - Language - C++->C++ Standard Library 改为“libstdc++”

1.3.2.3 请在 Build Seetings->Architectures->Valid Architectures 中，去掉 i386 和 x86_64

1.4 编译工程

以上步骤进行完后，编译工程，如果没有报错，恭喜你，集成 sdk 成功，可以进行下一步了。

2 原理简介

> 此 SDK 基于 rtmp 协议与服务器和其他客户端进行通讯 （Real Time Messaging Protocol（实时消息传送协议协议）是 Adobe Systems 公司为 Flash 播放器和服务器之间音频、视频和数据传输开发的协议）。

2.1 系统结构

完整的会议系统由客户端(有网页、pc 客户端、手机客户端等各种形式)、媒体服务器(Media Server，**以下简称 MS**）、web 服务器和数据库组成。MS、web 和数据库可以分开部署，也可能都部署在同一台服务器。可以有多台集群，也可能没有（web 和数据库）。任何时候，客户端和媒体服务器是必须的；**然而对于即时会议，web 系统和数据库可能不是必须的。**

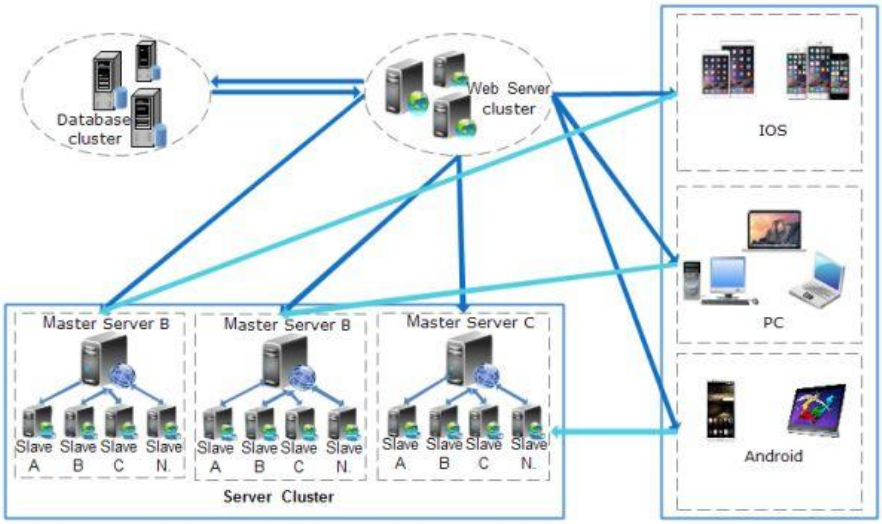
数据库：存储需要持久化的信息，如用户、预约的会议、会议中的文档、聊天、录制等信息。由 web 系统负责执行对它的读写；

Web 系统：管理系统、webAPI(会议、用户的管理和查询等 ,参见《会议系统接口文档_web》)、clientAPI（参见《会议接口文档.doc》）。另外，当 MS 需要将一些持久化信息（如录制件信息、人员进出会议、聊天信息持久化等）写入数据库时，也需要交由 Web 系统执行；

媒体服务器（MS）：负责会场的状态维护、信令响应及媒体中转。对 MS 来说，所有的会议都是即时的，存在于内存中。一旦会议结束，这场会议的信息就自动从 MS 中消失；

客户端：你懂的。

下图是最完整的部署场景，有数据库集群（Database cluster）、Web 服务器集群（Web Server cluster）和有主从级联架构的媒体服务器集群（Server Cluster）。



什么情况下我需要 web 系统和数据库？简单地说，如果需要
1 使用 web 管理系统
2 会议中的一些信息持久化（如会议录像、聊天信息保存、进出会场记录保存等）
就需要 web 系统和数据库了。**（为了预防客户之间会议号的冲突，如果租用微议云平台，则必须通过 web 系统来预约会议并生成全网唯一的会议号）**

2.2 会场

2.2.1 客户端与 MS 建立连接后，通过信令请求加入某个会场（传入该会场的会议号，自己的昵称等信息）；

2.2.2 MS 验证客户端信息无误，为该会议创建一个实例（如果已经创建则忽略），以下我们将这个内存中存在的实例称为会场；

2.2.3 MS 通知客户端加入会场成功，并返回当前会场中的状态信息（如已经有哪些人在会场中、谁是主席、会场是否已经在同步视频，等）。

2.3 信令机制

RTMP 协议能轻松传递控制命令并传递各种类型的参数。SDK 将会场内的信令进行了封装，并将各类参数以 json 格式提供，方便用户调用和响应回调。**如果觉得 SDK 已封装好的信令不够用，可以根据此信令机制自行编写会场信令——这提供了极大的灵活性，您可以自由地发挥想象。详见章节 6：自定义信令。**

会场中的信令主要有 2 种形式：

2.3.1 函数调用

SDK 提供的函数调用机制，是指用户在进入会场后，可以调用会场中的任何客户端的某个函数，并传入以 json 格式封装的任意参数。调用时需要指定：

2.3.1.1 调用谁？<必须> 可以是某一个个人，或包括自己的所有人，或除自己外的所有其他人。被调用方会收到一个回调，包括这是来自谁的调用，以及调用函数的名称和参数；

2.3.1.2 调用哪个函数？<必须> 函数名称，为字符串；

2.3.1.3 参数是什么？<不必须> 自定义的 json 数据。

函数调用就像会场中的定向广播。

2.3.2 消息发布

函数调用虽然方便，但它有“收到后即消失”的特性。有些会场状态（比如当前主席是谁？当前文档翻到第几页了？等）需要一直保持，并自动同步给刚刚进入会场的人。这时“定向广播”不够用了，需要一个“留言板”，这就是消息发布。

任何客户端都可以发布一条消息到会场中，发布时需要指定：

2.3.2.1 谁能看到？<必须>（可以对某个人可见，也可以对所有人公开，或除自己外的所有其他人——这样自己不会收到这条消息已发布的通知）；

2.3.2.2 消息的名字和 ID？<必须> 用以标识某条消息；

2.3.2.3 消息参数？<不必须> 消息附带的 json 数据；

2.3.2.4 绑定某个用户？<不必须> 当一个消息绑定到某个用户时，当该用户退出会场，这条消息也会自动被删除；

2.3.2.5 绑定某条消息？<不必须> 当一个消息绑定到另外一条消息时，当被绑定的消息被删除，这条消息也会自动被删除；

除了绑定消息可能被自动删除外，用户也可以手动删除一条消息。无论哪种情况，消息被删除时，与它相关的用户会收到一个回调通知。

> 如果会场内已经对所有人发布了一些消息，则刚进入会场的用户会立刻收到这些消息的发布通知。

2.4 身份与权限

2.4.1 主席（ChairMan）：控制会场，能够进行会场锁定/解锁、视频同步、指定发言、踢出会场等功能；

2.4.2 普通参会者，拥有查看和发布视频、发言的权力；

2.4.3 数据控制权（主席、普通参会者都可能持有，在一个会场内同一时间只能有一个人拥有此权限）：共享桌面、上传文档并翻页；

2.4.4 旁听用户（在直播会议中）：只能被动观看主席指定的视频，没有发布视频和发言的权力。

2.5 音视频

音视频数据基于 RTMP 协议的发布/订阅模型实现。SDK 对此机制进行了封装，并自动处理音视频发布/订阅相关的信令，降低了用户的使用难度。

音频：用户主动请求发言。对于自由会议，如果当前会议中已在发言人数未达到会场中最大同时发言人数（9 人），则自动开始发布自己的音频，否则进入等待队列，当有人停止发言时，按先来先到的原则，等待队列中的人开始自动发布音频。会场中的其他人会自动接收用户发布的音频。对于主席控制模式下的会议，请求发言后，需由主席允许，才会发布音频。无论是自由会议还是控制模式下主席可随时指定某人发言或停止某人的发言。

视频

2.5.1 观看者发出观看视频的请求给被看者，并向 MS 订阅该用户的视频；

2.5.2 被看者如果当前已经在发布视频，则什么也不做；否则自动发布自己的视频，并记录当前有谁在观看自己；

2.5.3 当观看者不愿再看某人，则取消向 MS 对该用户视频的订阅，并发出停止观看的信令给被看者；

2.5.4 当被看者发现当前已经没人在观看自己时，自动停止发布。

3 进入会议

3.1 初始化 SDK

引入相关头文件 #import”MeetingSession.h”

在工程的 AppDelegate 中的以下方法中，调用 SDK 对应方法:

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        [MeetingSession initializeAll];//全局初始化，单个进程只需执行一次
    });
}
```

在会议相关的 ViewController 中，创建 SDK 实例（创建 SDK 实例前必须确保已经实现了 MeetingDelegete 的代理方法）：

```
@interface EmmViewController () <MeetingDelegate>
//...

 - (void)viewDidLoad
{
    //...
    _session = [[MeetingSession alloc] initWithDelegate:self ClientCode:@"demo"
Serial:@""];//ClientCode 和 Serial 是为每个使用 SDK 的第三方用户分配的串号和验证码。测试时可以使用@”demo”和@””
    //...
}
```

3.2 进入会议

关于会议号（MeetingId），如果是微议云平台用户或需要将会议数据存数据库时，请调用 webapi 先创建会议并获得一个会议号。否则请自行指定一个会议号，会议号应该是数字或字母组成的字符串。

接口调用

```
// 进入会议
// ip:          服务器地址（ip 或域名）
// port:        服务器端口
// nickname:    自己的昵称
// meetingid:   会议名
// userid:      调用者自己指定的用户 id，用于在会议中分辨身份
    [_session enterMeeting:@“192.168.0.123”
                      Port:443
                  NickName:@“yourname”
                 MeetingID:@“201506290001”
                    UserID:userId ServerMix:NO];
```

连接完成后的通知（MeetingDelegete 的代理方法）：进入会场成功，会依次收到 onConnect，onUserIn（这是用户进入的通知，有任何用户进入会场，都会调用这个方法。自己进入时也会。），onPresentComplete 回调。

```
/**
typedef enum
{
    CONNECT_SUCCESS = 0,//连接成功
    CONNECT_FAILED = 1,//连接失败
    CONNECT_EXCEPTION = 2,//网络断开
    CONNECT_LOCKED  =3//会议室已锁定
}CONNECT_RESULE;
// 连接完成。status=0 时连接服务器成功，否则失败。失败原因见 CONNECT_RESULE
- (void)onConnect:(int)status
```

```
{
    //...
}


- (void) onUserIn:(uint32_t)userID Self:(bool)bself
{
    if (bself)
    {
        _myPeerID = userID;
    }
    else
    {
        // 获取用户昵称
        EmmUser *user = [[MeetingUserMgr instance] getUser:_myPeerID];
        if (user != nil)
        {
            NSLog(@"%@'s in", [user getName]);
        }
    }
}


- (void) onPresentComplete
{
    //...
    [[UIApplication sharedApplication] setIdleTimerDisabled:YES];//禁用自动锁屏
    //...
}
```

关于 onUserIn 的重要说明：假设会场中已有用户 A 和 B。当用户 C 进入会场，其他用户（A 和 B）会同时收到 onUserIn 回调，说明 C 已进入；C 自己在进入会场时会收到 3 次 onUserIn 回调，分别来自 A、B 和 C 自己。收到 onUserIn 后，即可通过 userid 从 MeetingUserMgr 获取用户的属性了（详见 MeetingUserMgr.h）。

3.4 退出

退出会场分两种类型：主动退出和网络异常

- **主动退出会场：调用 SDK 的退出接口，**当收到代理方法 onDisconnect 时，可以释放会议界面；
- 网络异常： 代理会收到 onConnect:CONNECT_EXCEPTION，此时可以释放会议界面

退出方法

```
[_session exitMeeting];
```

退出成功后自己会收到 onDisconnect 回调，其他用户会收到 onUserOut 回调。

4 音视频

```
我们推荐用户在手机终端上使用“画中画”方式查看视频，即一路自己，一路别人的视频。
```

4.1 发言

调用 requestSpeaking 方法即可申请发言（用户如果需要进入后即自动发言，则把 requestSpeaking 放在 onPresentComplete 中调用即可）：

```
//自己要求发言
[_session requestSpeaking:_myPeerID];
```

```
//主席要求某人发言（需要先成为主席）（主席要求自己发言时，即使当前发言人数已达上限，仍可不进行排队直接发言）
[_session requestSpeaking:peerID];
```

4.2 停止发言

```
//自己取消发言
[_session cancelSpeaking:_myPeerID];


//主席取消某人发言（需要先成为主席）
[_session cancelSpeaking:peerID];
```

4.3 发言状态变化的回调

无论是其他人还是自己的发言状态变化了，应用都会收到这个回调。SDK 会自动去订阅已经发言的用户的音频流，上层界面只需要根据状态更新界面即可。

```
//userID 谁的发言状态发生变化
//status 新的发言状态
//enum SpeakStatus
//{
//    m_RequestSpeak_Disable= 0,//没发言
//    m_RequestSpeak_Allow,//发言中
//    m_RequestSpeak_Pending//申请发言状态，未决状态
//};
- (void) onSpeakChange:(uint32_t)userID Status:(SpeakStatus)status;
```

4.4 观看某人的视频

```
// peerID：该用户的 peerID
// VideoIndex：预留参数，填 0 即可
```

```
// Window：用来观看视频的 UIView 指针
[_session watchVideo:peerID VideoIndex:0 Window:videoView];


// 当播放视频的页面要被隐藏时，应用要暂停视频的播放，然后在页面恢复时再调用一次 playVideo
[_session pausePlayVideo:peerID Complete:nil];
```

4.5 不观看某人的视频

```
// 参数与 wateVideo 雷同，只少了个 Window
[_session unWatchVideo:peerID VideoIndex:0];
```

4.6 关于主席同步视频

旁听用户无权主动观看某个用户的视频，那么他们如何看到会场中的视频呢？这就需要由主席来强制所有人都开始/停止观看某一路视频。这就是主席同步视频。对于主席侧，这个功能分 2 步执行（假设当前用户已经成为主席）：

```
// 1 主席将会议设置为自动同步视频模式
[_session syncVideo:YES AutoSync:YES];


// 2 主席观看某人视频
[_session watchVideo:peerID1 VideoIndex:0 Window:videoView];


// 2 主席不观看某人视频
[_session unWatchVideo:peerID2 VideoIndex:0];
```

对于被观看者，SDK 会自动发布视频。
对于所有人（包括被观看者），会收到 onSyncWatchVideo 回调。在收到通知时需要根据主席的意图开始/停止观看：

```
- (void) onSyncWatchVideo:(uint32_t)userID VideoIndex:(uint32_t)videoIndex
Watch:(bool)bWatch
{
    if (userID == _myPeerID)
        return;
```

```
    if (userID == _currentWatchingID)//如果当前正在观看此人
    {

    }

    if (bWatch)
    {
        if (userID == _currentWatchingID) return;
        else if(_currentWatchingID != 0)
        {
            [_session unWatchVideo:currentWatchingID VideoIndex:0];
        }
        [_session watchVideo:userID VideoIndex:0 Window:videoView];
        _currentWathingID = userID;
    }
    else if (userID == _currentWatchingID)
    {
        [_session unWatchVideo:userID VideoIndex:0];
        _currentWatchingID = 0;
    }
}
```

5 会场内的信令

5.1 主席与会场控制

用户成为主席后，将有权对会场进行会场锁定/解锁、视频同步/取消同步、控制发言、踢出用户等操作。详情请参考 MeetingSession.h。视频同步请参考 4.6 关于主席同步视频。

5.1.1 成为主席

```
// 将自己或别人设置为主席
[_session changeChairMan:peerID];
```

```
// 则会场内所有人会收到回调
- (void) onChairManChange:(uint32_t)oldChairMan NewChairMan:(uint32_t)newChairMan
{
    if(oldChareMan)
        NSLog(@"%u 已经把主席身份移交给%u", oldChairMan, newChairMan);
    else
        NSLog(@"%u 已经成为了主席", newChairMan);
}
```

5.1.2 会场锁定

```
// 锁定会场
[_session changeChairMan:peerID];

// 则会场内所有人会收到回调
- (void) onLockRoomChange:(bool)bLock
{
    NSLog(@"会场已被%@", bLock ? @"锁定" : @"解锁");
}
```

5.1.3 主讲模式和自由模式

主讲模式下，所有人请求发言后，其发言状态会变为“请求发言”，经主席同意后才能发言；自由模式下，只要当前会场发言人数未达最大限制，则立刻发言成功。请参考章节 2.5 音视频。

```
// 切换模式
MeetingMode mode = MeetingMode_ChairmanControl;//或 m_MeetingMode_Free
[_session changeMode:mode];

// 则会场内所有人会收到回调
- (void) onMeetingModeChange:(MeetingMode)mode
{
    NSLog(@"会场已被设置为%@", mode == m_MeetingMode_ChairmanControl ? @"主席控制模式" : @"自由会议模式");
}
```

5.2 数据操作

要进行数据操作，首先需要获取数据控制权。数据控制权可以自行申请，也可由主席授予。

5.2.1 获得数据控制权

```
// 申请数据控制权
[_session requestHost:_myPeerID];


// 则会场内所有人会收到回调
- (void) onHostChange:(uint32_t)userID Status:(HostStatus)status
{
    switch(status)
    {
        case m_RequestHost_Disable:
            NSLog(@"%u 释放了数据控制权", userID);
            break;
        case m_RequestHost_Allow:
            NSLog(@"%u 获取了数据控制权", userID);
            break;
        case m_RequestHost_Pending:
            NSLog(@"%u 申请数据控制权", userID);
            break;
    }
}
```

5.2.1 共享屏幕

PC 获取数据控制权后，可以共享自己的屏幕给会场内的其他人。其他人在收到信令后做如下响应：

```
// 屏幕共享
- (void) onAppShareMsg:(uint32_t)hostID Start:(bool)bStart
```

```
{
    //...
    if(bStart)
    {
        [_session playScreen:hostID View:view Complete:nil];
    }
    else
    {
        [_session unPlayScreen:hostID Complete:nil];
    }
}

// 当页面播放屏幕共享的页面要被隐藏时，应用要暂停屏幕共享的播放，然后在页面恢复时再调用一次
playScreen
[_session pausePlayScreen:hostID Complete:nil];
```

5.3 文字聊天

```
// 发送文字聊天
EmmUser *user = [[MeetingUserMgr instance] getUser:_myPeerID];
if (user != nil)
{
    NSString *text = [NSString stringWithFormat:@"i'm %@ (%@ v:%@)", [user getName],
[[UIDevice currentDevice] name], [[UIDevice currentDevice] systemVersion]];
    //peerID ： 发给谁，大于 0 的整数代表某个用户的 peerid，0 代表除自己外的所有人
    //Text：要发送的文字
    //Name：自己的昵称
    //TextFormat：NSDictionary，用户自定义的字体信息，可以为空
    [_session sendTextMessage:0 Text:text Name:@"iphone" TextFormat:nil];
}

// 则会场内所有人会收到回调
// fromid：发送者 id
// type：0:公聊，2：私聊
- (void) onReceiveTextMsg:(uint32_t)fromid Type:(int)type Text:(NSString*) msg
TextFormat:(NSDictionary*)textFromat
{
    EmmUser *user = [[MeetingUserMgr instance] getUser:fromid];
```

```
    [self log:[NSString stringWithFormat:@"%@ says %@", [user getName], msg]];
}
```

6 自定义信令

> 您可以利用自定义信令扩展您自己的视频会议。相关介绍请见章节 2.3 信令机制

6.1 函数调用

```
/**
 *  调用其他客户端的函数
 *
 *  @param name 函数名
 *  @param toWhom=0:除自己外的所有人，-1:包括自己的所有人，其他值:某个参会者的 peerID
 *  @param params 用户自定义的参数，可以是 nil，也可以是 NSNumber、NSString、NSArray 或
NSDictionary，如果是个容器，则容器内部包含的只能是 NSNumber 和 NSString
 */
- (void)callClientFunction:(NSString*)name To:(int32_t)toWhom
Params:(NSObject*)params;

/**
 *  收到函数调用
 *
 *  @param 参数与上面 callClientFunction 一致
 */
- (void)ClientFunc_Call:(NSString*)name PeerID:(int)fromID Params:(NSObject*)params;
```

这是个例子：

```
// 调用侧
NSDictionary *param = @{@”name”:@”老张”, @“age” :@30, @”gender”:@”男”};
[_session callClientFunction:@“HereComesNewGuy” To:0 Params:param];
```

```
// 则其他用户会收到：
- (void)ClientFunc_Call:(NSString*)name PeerID:(int)fromID Params:(NSObject*)params
{
    if ([name isEqualToString:@"HereComesNewGuy"] && [params isKindOfClass[NSDictionary class]])
    {
        NSDictionary *dic = (NSDictionary*)params;
        NSLog(@"新人%@，现年%@岁，%@性。", dic[@"name"], dic[@"age"], dic[@"gender"]);
    }
}
```

6.2 消息发布

```
/**发布消息到会场
 * 消息属性：
 * toID:接受者 ID uint——0 为除自己以外所有用户,0xFFFFFFFF 为包括自己的所有用户，0xFFFFFFFE 为不转发此消息，0xFFFFFFFD 为所有非旁听用户，其余为会议用户 ID
 * name: 消息名称 string
 * id：消息 ID string——消息的唯一 ID，每个消息拥有自己唯一的 ID 作为标识
 * associatedUserID：消息关联用户 ID uint——消息关联的用户，如果此用户退出，则此消息被释放
 * associatedMsgID：消息关联消息 ID string——消息管理的消息 ID，如果 associatedMsgID 指定的消息被释放，那么此消息被释放
 * body：消息内容 Object——可以是 nil，也可以是 NSNumber、NSString、NSArray 或 NSDictionary，如果是个容器，则容器内部包含的只能是 NSNumber 和 NSString
 */
- (void) publishMessage:(NSString*)name ToID:(uint32_t)toID AssociatedUserID:(uint32_t)associatedUserID Body:(NSObject*)body MsgID:(NSString*)id AssociatedMsgID:(NSString*)associatedMsgID;

// 删除消息（参数与发布消息相同）
- (void) deleteMessage:(NSString*)name ToID:(uint32_t)toID AssociatedUserID:(uint32_t)associatedUserID Body:(NSObject*)body MsgID:(NSString*)id AssociatedMsgID:(NSString*)associatedMsgID;

/**消息处理函数
 * 消息属性：
 * msg.senderID：发送者 ID uint
 * msg.toID：接受者 ID uint——0 为除自己以外所有用户，0xFFFFFFFF 为包括自己的所有用户，0xFFFFFFFE 为不转发此消息，0xFFFFFFFD 为所有非旁听用户，其余为会议用户 ID
```

```
 * msg.name：消息名称 string
 * msg.id：消息 ID string——消息的唯一 ID，每个消息拥有自己唯一的 ID 作为标识
 * msg.associatedUserID：消息关联用户 ID uint——消息关联的用户，如果此用户退出，则此消息被释放
 * msg.associatedMsgID：消息关联消息 ID string——消息管理的消息 ID，如果 associatedMsgID 指定的消息被释放，那么此消息被释放
 * msg.body：消息内容 Object
 */
- (void) onRemotePubMsg:(NSDictionary*)msg;

// 消息被删除时的回调，参数与 onRemotePubMsg 同
- (void) onRemoteDelMsg:(NSDictionary*)msg;
```

这是个例子：

```
// 调用侧
// 发布一条消息，名字和 MsgID 都是“TomorrowWeather”，内容是“晴”，发给会场内的所有人，且这条消息会随着自己退出会议而被自动删除（因为 AssociatedUserID 指定了自己的 id），且这条消息不会随着其他消息的删除而被删除（因为 AssociatedMsgID 为空）
[_session publishMessage:@”TomorrowWeather” ToID:0 AssociatedUserID:_myPeerID Body:@”晴” MsgID:@”TomorrowWeather” AssociatedMsgID:nil];

// 则其他用户（包括在自己之后进入会场的用户）会收到：
- (void) onRemotePubMsg:(NSDictionary*)msg
{
    if ([msg[name] isEqualToString@”TomorrowWeather”])
    {
        NSLog(@”明天的天气是：%@”, msg[@”body”]);
    }
}

// 当自己（发布消息者）退出会议，或调用 deleteMessage 时，会场内其他用户会收到：
- (void) onRemoteDelMsg:(NSDictionary*)msg
{
    if ([msg[name] isEqualToString@”TomorrowWeather”])
    {
        NSLog(@”天气预报员停止了天气预报”);
    }
}

// 此后再新进入会场的用户将不再收到这条消息的通知。
```

7 FAQ

Q: SDK 太大了，怎么办？

A: SDK 大不影响打包成 ipa，打包出 ipa 安装包后，一般只会增加 5MB 左右；

Q: SDK 是否支持 64 位？

A: 支持；

Q: SDK 编译报错？

A: 此 SDK 目前不支持模拟器调试，请在 Build Seetings->Architectures->Valid Architectures 中，去掉 i386 和 x86_64。如果仍然报错，按章节 1.3 检查工程配置；

Q: 在会议进行中锁屏后，再返回应用，会有断点？

A: 请在 main.m 中如下位置加上如下代码。加上后，联机 debug 时仍会有断点，直接按 Command+Shift+Y 跳过即可。