Gensee.Inc

RtSDK

目 录

1 修订记录

修订日期	版本号	描述	修订人
2015/6/1		文档创建	Gaojin Hsu
2015/6/18		新增举手接口	Gaojin Hsu
2015/7/10		新增文档显示模式，增强文档模块稳定性	Gaojin Hsu
2015/7/15		紧急修复聊天模块接受第三方表情崩溃问题	Gaojin Hsu
2015/7/21		修复问答中答案内容对象过早释放，用户信息类中角色不匹配等 bug，新增用户信息类判断状态的几个常用属性	Gaojin Hsu
2015/8/7		音频麦克风打开无声音 bugfix，桌面共享清晰度优化，接受第三方 emoji 优化，添加日志处理	Gaojin Hsu
2015/10/23		优化 swift 工程引用 SDK 时的稳定性，视频 View 新增显示模式选项，文档支持 ppt 页内过度动画	Gaojin Hsu
2016/4/6		新增第三方验证机制	Gaojin Hsu
2016/6/3		支持 ipv6－only 网络环境	Gaojin Hsu
2016/12/14		1. 升级 https 网络访问请求 2. 优化文档 View 手势 3. 新增美颜滤镜 4. fix 若干 bug	Gaojin Hsu
2017/4/11		1. 优化内部登录 http 接口 2. 支持发布 960， 540 分辨率 3. ffmpeg 更新至 3.2 4. 新增聊天审核	Gaojin Hsu

2 文档介绍

2.1 文档目的

众多客户希望在使用我公司的直播服务的同时，能拥有更多的 app 定制能力，特别是对于拥有开发人员资源的公司，他们希望拥有自己风格的 app。为此，特提供 iOS RtSDK，以便客户可以针对自己的业务场景设计最符合自己业务场景的客户端。

2.2 术语与缩写解释

编号	术语	解释
1.		
2.		
3.		
4.		
5.		
6.		
7.		
8.		
9.		

3 SDK 使用准备

3.1 温馨提示

欢迎客户公司开发人员随时和我们沟通，在相互理解的基础上，共同配合完成开发任务。但是为了提高开发效率，恳请客户公司开发人员在开发之前和开发过程中仔细阅读开发文档和查看 Demo 代码。如果发现文档中有难以理解或者文档中没有记载的问题，请随时找相关负责人沟通。

Demo 中的代码用以示范 SDK 中的接口调用。不保证最符合客户的需求。请勿照抄 Demo 中的代码，请在理解 SDK 机制的基础上修改。如果发现任何疑似 bug 的现象，欢迎及时与相关负责任沟通。

如果在 Demo 中发现了 bug 或者发现了 SDK 的 bug，在告知我公司开发人员时，请将系统版本，硬件版本，产生 bug 的操作流程一并告知我公司开发人员，必要时需要提供日志。

3.2 工程设置

3.2.1 build settings

1. *请将工程中的 AppDelegate.m 改成 AppDelegate.mm 或者在 other linker 里新增-lc++*
2. *支持系统 ios6+ ,若使用视频硬件加速编解码则支持 ios8+*
3. *other linker 设置加上 –ObjC*
4. *目前 RtSDK 可支持真机编译 和 不完美支持模拟器编译（目前在模拟上声音失效）*
5. *不支持 bitcode*

3.2.2 系统库

1. *AVFoundation.framework*
2. *OpenGLES.framework*
3. *AudioToolbox.framework*
4. *CoreMedia.framework*
5. *GLKit.framework*
6. *libz.tbd*
7. *libiconv.tbd*
8. *libicucore.tbd*

3.2.3 直播库

1. *RtSDK.framework*

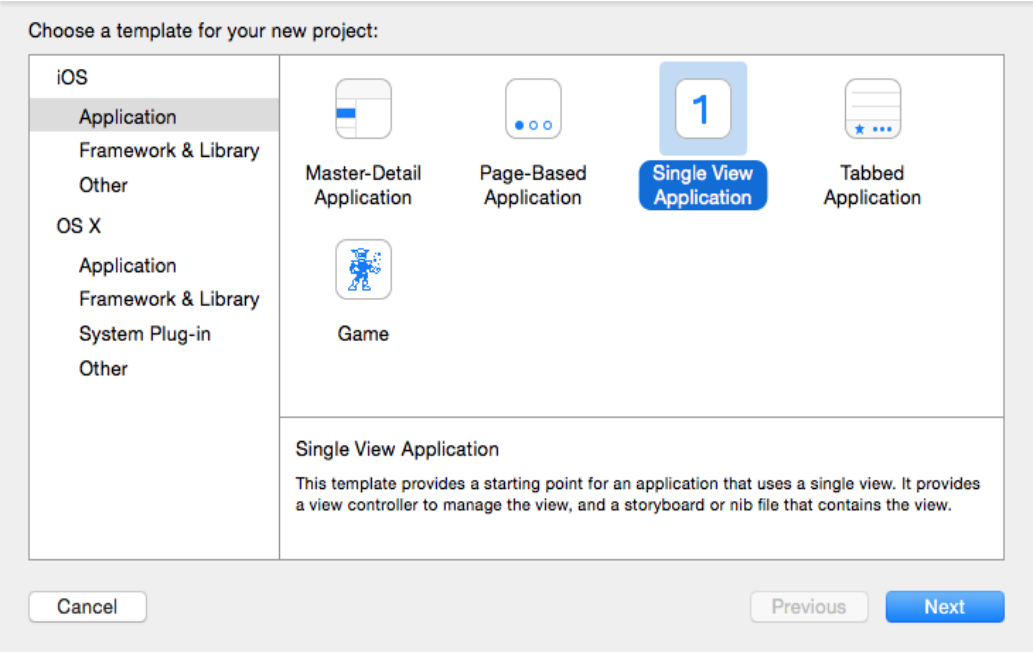
3.2.4 其他资源

1. 需要添加 RtSDK.bundle 到工程中, 里面包含所有的资源文件，比如表情图片等

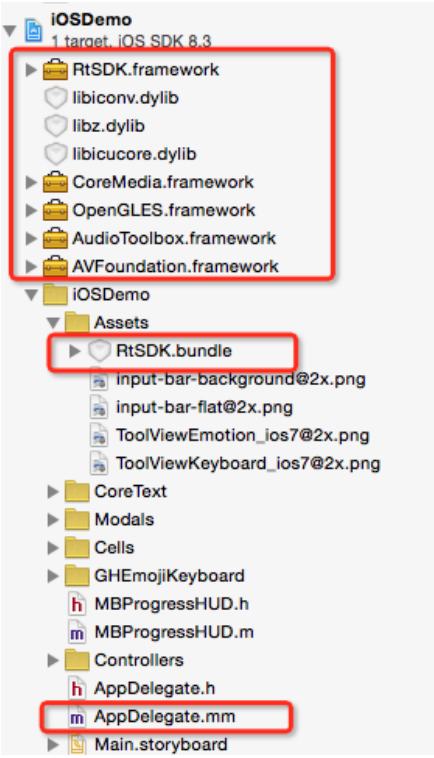
4 SDK 快速入门

考虑到很多用户在看文档时不知道从何开始，现在在这里写一个快速搭建直播的教程。

第一步：新建一个工程， 取名为 iOSDemo



第二步： 按照文档中使用准备章节中所述，把所有的库和资源添加到工程, 并将 AppDelegate.m 改为 AppleDelegate.mm 或者在 other linker flag 中加入-lc++



第三步：创建直播实例，加入直播

1. 在 viewDidLoad 方法中创建 GSBroadcastManager 的单实例，将直播代理（broadcastRoomDelegate）设置成该 ViewController，用来接受直播信息的回调。如图，红色框内为添加的代码。

```
#import "ViewController.h"
#import <RtSDK/RtSDK.h>

@interface ViewController () <GSBroadcastDelegate>
@property (nonatomic, strong)GSBroadcastManager *manager;
@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.

    _manager = [GSBroadcastManager sharedBroadcastManager];

    _manager.broadcastDelegate = self;
```

2. 实例创建好之后就要开始连接直播了，注意，这里所谓的连接直播，是指和直播建立了联系，并不是加入到直播中，这是两个概念（后面会涉及到加入操作）。

```
- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.


    _manager = [GSBroadcastManager sharedBroadcastManager];

    _manager.broadcastDelegate = self;

    GSConnectInfo *connectInfo = [GSConnectInfo new];
    connectInfo.domain = @"demo.gensee.com";
    connectInfo.serviceType = GSBroadcastServiceTypeWebcast;
    connectInfo.roomNumber = @"29111638";
    connectInfo.nickName = @"ios";
    connectInfo.watchPassword = @"333333";

    [_manager connectBroadcastWithConnectInfo:connectInfo];


}
```

这里新建了一个 GSConnectInfo 对象，这个对象封装了一些连接直播所需要的参数：

1.) domain 即域名,不能为空;

2.) serviceType 为服务类型，如果你的站点是 webcast 那么就要设置为 GSBroadcastServiceTypeWebcast，如果是 training，则设置为 GSBroadcastServiceTypeTraining。如果没有设置，则默认为 GSBroadcastServiceTypeWebcast;

3.) roomNumber 即房间号，还有一个 webcastID（直播 ID），这两个只需要填写一个即可，假如两个都写了，将优先使用 webcastID，如果都没有写，则会产生错误

4.) nickName 为显示的昵称，可以自行设置

5.) watchPassword 即观看密码

这些参数都可以在类似于下图这样的页面中找到

webcast：

training：

3. 创建好了 GSBroadcastManager 对象，发出了连接直播的请求，然后就要在收到直播连接结果的时候做相应处理，大致上的逻辑就是，如果连接成功了，那就要加入直播，如果连接失败，则做相应的报错。下面开看代码。

```
- (void)broadcastManager:(GSBroadcastManager*)manager didReceiveBroadcastConnectResult:(GSBroadcastConnectResult)result
{
    switch (result) {
        case GSBroadcastConnectResultSuccess:

            // 直播连接成功，加入直播
            [_manager join];

            break;

        case GSBroadcastConnectResultInitFailed:

        case GSBroadcastConnectResultJoinCastPasswordError:

        case GSBroadcastConnectResultWebcastIDInvalid:

        case GSBroadcastConnectResultRoleOrDomainError:

        case GSBroadcastConnectResultLoginFailed:

        case GSBroadcastConnectResultNetworkError:

        case GSBroadcastConnectResultWebcastIDNotFound:
            break;


        default:
            break;
    }
}

/*
 直播连接代理
 rebooted为YES，表示这次连接行为的产生是由于根服务器重启而导致的重连
 */
- (void)broadcastManager:(GSBroadcastManager*)manager didReceiveBroadcastJoinResult:(GSBroadcastJoinResult)joinResult selfUserID:(long long)userID
    rootSeverRebooted:(BOOL)rebooted;
{

}
```

上图中有两个代理方法，这两个方法都是 GSBroadcastRoomDelegate 协议中的，第一个就是反馈连接直播结果的。正常情况下，在正确设置了代理之后调用了 connectBroadcastWithConnectInfo：方法后就会触发这个代理，返回连接结果，在连接成功后就要尝试调用 join 方法加入到直播中。连接失败了有各种错误说明，如果你觉得不需要这么详细，可以在最后的 case 里做笼统的报错。第二个代理方法是相对与 join 方法的，反馈的是加入直播的结果。如果 joinResult 等于 GSBroadcastJoinResultSuccess，就表示加入成功， userID 就是自己的用户 ID ，rebooted 参数表示这次连接是否是因为根服务器重启导致的（服务器上会记录直播中的某些状态，根服务器重启这些状态将会丢失，所以建议在本地记录自己的状态，比如自己的麦克风是否被打等，然后在重连成功的时候主动恢复）。这时，你如果查看 PC 的直播客户端，就会发现你已经是在直播中的用户了。

第四步：添加自己需要的代理，处理自己感兴趣的数据

在完成了第三步之后，你会发现虽然你已经加入了直播，但是什么也干不了，什么也看不了，这里我们以音频为例，介绍如何与直播端进行交互。

```
@interface ViewController () <GSBroadcastDelegate, GSBroadcastAudioDelegate>

@property (nonatomic, strong)GSBroadcastManager *manager;

@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.


    _manager = [GSBroadcastManager sharedBroadcastManager];

    _manager.broadcastDelegate = self;
    _manager.audioDelegate = self;

    GSConnectInfo *connectInfo = [GSConnectInfo new];
    connectInfo.domain = @"demo.gensee.com";
    connectInfo.serviceType = GSBroadcastServiceTypeWebcast;
    connectInfo.roomNumber = @"29111638";
    connectInfo.nickName = @"ios";
    connectInfo.watchPassword = @"333333";

    [_manager connectBroadcastWithConnectInfo:connectInfo];


}
```

首先，我们需要设置音频相关的代理 audioDelegate，在 GSBroadcastManager 中，不同的模块都有对应的代理，只有设置之后，才会收到相关模块的数据回调。

然后在相关回调中做相应的处理。

```
// 音频模块连接代理
- (void)broadcastManager:(GSBroadcastManager*)manager didReceiveAudioModuleInitResult:(BOOL)result
{
    if (result) {
        // 打开扩音器
        [_manager activateSpeaker];

        AVAudioSession *audioSession = [AVAudioSession sharedInstance];
        [audioSession setActive:YES error:nil];
        [audioSession setCategory:AVAudioSessionCategoryPlayAndRecord
                      withOptions:AVAudioSessionCategoryOptionDefaultToSpeaker error:nil];
    }
}
```

每个模块代理中都有一个返回模块初始化结果的代理。这里我们想要听到从直播客户端传来的声音，只需要在确认模块正确初始化后，打开喇叭就可以了。下面设置 AVAudioSession 的代码是为了让声音从扩音器中传播出来，默认是从听筒中传出。其他模块的处理方式也很类似，详见附带的 Demo 和文档。

第五步：退出直播

如果想要退出直播，主要的方法有两个，

```
[_manager leaveAndShouldTerminateBroadcast:NO];

[_manager invalidate];
```

调用 leaveAndShouldTerminateBroadcast：方法后，用户就会离开直播。当然之后你还可以调用 join 方法再进直播，布尔值参数的意思是，在离开直播后是否结束整个直播，如果你是直播的组织者，在组织者离开直播后可以结束整个直播，但是一般的直播参加者离开直播后不应该关闭整个直播。

这里还有一个 invalidate 方法，如果调用了这个方法，那将会断开与直播的连接，清理相应的资源，在调用了 invalidate 方法后再也无法靠调用 join 方法进入直播了，要重新从 connect 方法开始。

注意：sdk 中除了 invalidate 方法，绝大部分 GSBroadcastManager 的方法都是有回调的，比如 leaveAndShoudTermindateBroadcast：方法，会产生如下回调，最正确的做法是在方法的回调里去释放直播资源。

```
// 自己离开直播代理
- (void)broadcastManager:(GSBroadcastManager*)manager didSelfLeaveBroadcastFor:(GSBroadcastLeaveReason)leaveReason
{
    [_broadcastManager invalidate];
```

5 SDK 中的类

6 代码示例

参见 RtSDKDemo，RtSDKDemo 只是示范 SDK 中接口的使用方式，并不保证使用方式最符合用户的需求，请用户在充分理解 SDK 机制的前提下，根据自己的情况处理数据。对照搬 RtSDKDemo 代码引起的性能问题和其他非 SDK bug，不负任何责任。

7 常见的问题

7.1 我该如何填写参数，使 Demo 正确运行?

请开发人员联系负责和我公司交接直播端（网页入口，启动的 PC 客户端）的同事，让他提供对应的参数信息或者用 websdk 从服务器获取带有参数的 json 数据。

7.2 为什么视频全屏之后不会充满整个屏幕，为什么视频边上有留白？

由于视频原数据有一定的宽高比，而且该宽高比在绝大多数情况下和手机屏幕的宽高比不相等，为了保证视频在显示的时候不被拉伸而导致画面扭曲，一般会根据视频原数据本身的宽高比进行计算，得出最大的并且具有正确宽高比的视频显示区域。

假如，一个视频数据的宽是 320，高为 240， 你创建的用于渲染该视频的 videoView 的宽为 320，高为 300，那么显然，用这样的 videoView 去渲染视频，在视频的上下两边会有留白。如果强行实现所谓的全屏，那就会产生下面两种结果中的一种：

1. 画面部分会被截取，在这个例子中，显然是视频的横向两边会被截掉
2. 画面没有被截取，但是会被拉伸。

如果想实现全屏应该怎么办?

1. 尽量保证视频的宽高比和 videoView 的宽高比一致
2. 终端设备种类多，宽高比有很多种的情况下，要考虑适当允许视频被截取

7.3 如何设置在放视频的时候不自动锁屏?

这个需要在上层处理，和 SDK 并无关系。

请参考：http://stackoverflow.com/questions/9904306/disable-automatic-screen-lock-in-ios-5-1

7.4 如果统计直播中的人数

```
/**
 *  其他用户加入房间代理
 *
 *  @param manager  触发此代理的GSBroadcastManager对象
 *  @param userInfo 用户信息
 *  @see GSBroadcastManager
 *  @see GSUserInfo
 */
- (void)broadcastManager:(GSBroadcastManager*)manager didReceiveOtherUser:(GSUserInfo*)userInfo;

/**
 *  其他用户离开房间
 *
 *  @param manager 触发此代理的GSBroadcastManager对象
 *  @param userID  离开直播的用户ID
 *  @see GSBroadcastManager
 */
- (void)broadcastManager:(GSBroadcastManager*)manager didLoseOtherUser:(long long)userID;
```

自己定一个变量，在收到一次 didReceiveOtherUser 的时候 增 1，在收到 didLoseOtherUser 的时候减 1 就是当前直播里的总人数

7.5 如果获取直播的状态，比如，直播暂停，直播结束

```
/**
 * 直播状态改变代理，比如暂停或者恢复；如果是直播结束，将从 -(void)broadcastManager:didSelfLeaveBroadcastFor:回来， 因为直播结束，默认都会被踢出
 *
 * @param manager 触发此代理的GSBroadcastManager对象
 *
 * @param status 枚举值表示状态
 *
 * @see GSBroadcastManager
 *
 */
- (void)broadcastManager:(GSBroadcastManager*)manager didSetStatus:(GSBroadcastStatus)status;
```

加入直播时，默认的状态都要设置为直播未开始，然后根据此接口返回的状态修改，直播结束的状态不会从这里来，因为直播结束自己会被踢出直播间，所以会从

```
/**
 * 自己离开直播代理
 *
 * @param manager 触发此代理的GSBroadcastManager对象
 *
 * @param leaveReason 枚举值，表示离开直播的原因
 *
 * @see GSBroadcastManager
 *
 * @see GSBroadcastLeaveReason
 *
 */
- (void)broadcastManager:(GSBroadcastManager*)manager didSelfLeaveBroadcastFor:(GSBroadcastLeaveReason)leaveReason;
```

这个接口返回，reason 为直播结束。

7.6 如果获取直播时间

自己起一个定时器，可以每秒钟执行一次获取直播时间

```
228 /**
229  *  获取直播运行时间，暂停等时间不计入。
230  *
231  *  @return 时间值，单位秒
232  */
233 - (unsigned int)broadcastRunningTime;
234
```

7.7 RtSDK 和第三方库的冲突

因为 RtSDK 会用到一些经典的音视频开源库（比如 ffmpeg，webrtc），以及 Objective-C 的语言特性， 如果你的工程中包含两个以上音视频相关的第三方 sdk 很可能会产生冲突，这些冲突展现出来的形式有一下几种：

1. 编译不过，报错
2. 运行时崩溃

这些冲突需要提供 sdk 的双方配合解决，不排除有解决不了的或者解决代价太大的冲突，所以请尽量不要在同一个工程里使用两个音视频相关的 sdk。

7.8 如何设置发布视频的分辨率 和 码率

分辨率在收到如下代理时设置

```
- (BOOL)broadcastManager:(GSBroadcastManager *)manager querySettingsInfoKey:(NSString *)key numberValue:(int*)value
{
    if ([key isEqualToString:@"save.video.width"]) {
```

key 的值分别为 save.video.width 和 save.video.height

当前分辨率支持 352 X 288， 640 X 480， 1280 X 720，注意设置分辨率的时候，手机采集时的状态，横屏采集和竖屏采集宽高刚好相反。

码率：

在收到此代理时设置，可自行调节，码率大视频更清晰，但更占带宽，自行取舍。

```
/**
 *  设置采集比特率
 *
 *  @param manager 触发此代理的GSBroadcastManager对象
 *  @param session session
 */
- (void)broadcastManager:(GSBroadcastManager *)manager didSetupCustomBitRate:(VTCompressionSessionRef)session;
```

```
    VTSessionSetProperty(session, kVTCompressionPropertyKey_AverageBitRate, (__bridge CFTypeRef)@(400 * 1024));
    VTSessionSetProperty(session, kVTCompressionPropertyKey_DataRateLimits, (__bridge CFArrayRef)@[@(500 * 1024/ 8), @1]);
}
```

7.9 如何发送聊天，发送表情，显示聊天表情

聊天对象的结构比较简单

```
/**
 *  封装了文本聊天信息
 */
@interface GSChatMessage : NSObject

/**
 *  聊天数据的富文本字符串
 */
@property(strong, nonatomic)NSString* richText;

/**
 *  聊天数据的普通文本字符串
 */
@property(strong, nonatomic)NSString* text;
```

假如现在要发送一个  大家好！ 的表情+文字的聊天信息，那么

1.) richText 为

<SPAN style=\"FONT-SIZE: 10pt; FONT-WEIGHT: normal; COLOR: #000000; FONT-STYLE: normal\"><IMG src=\"emotion\\emotion.smile.gif\" custom=\"false\">大家好！ </SPAN>

richtext 中消息内容前后要用固定的标签包括：

<SPAN style=\"FONT-SIZE: 10pt; FONT-WEIGHT: normal; COLOR: #000000; FONT-STYLE: normal\">消息内容</SPAN>

<IMG src=\"emotion\\emotion.smile.gif\" custom=\"false\"> 为微笑的表情，每个表情都有不同的 src 字段，这个对应关系可以在 RtSDK.bundle 的 plist 文件资源中找到。

2.) text 为

【你好】大家好！

text 的聊天内容前后要加上，你好的表情对应【你好】，对应关系从 RtSDK.bundle 的 plist 文件资源中找到。

text 字段是为了兼容老版本的 web 端

接收显示表情也是按照这个规则去解析，然后用图文混排的方式显示，幸运的是可以找到很多第三方库，比如 YYText。