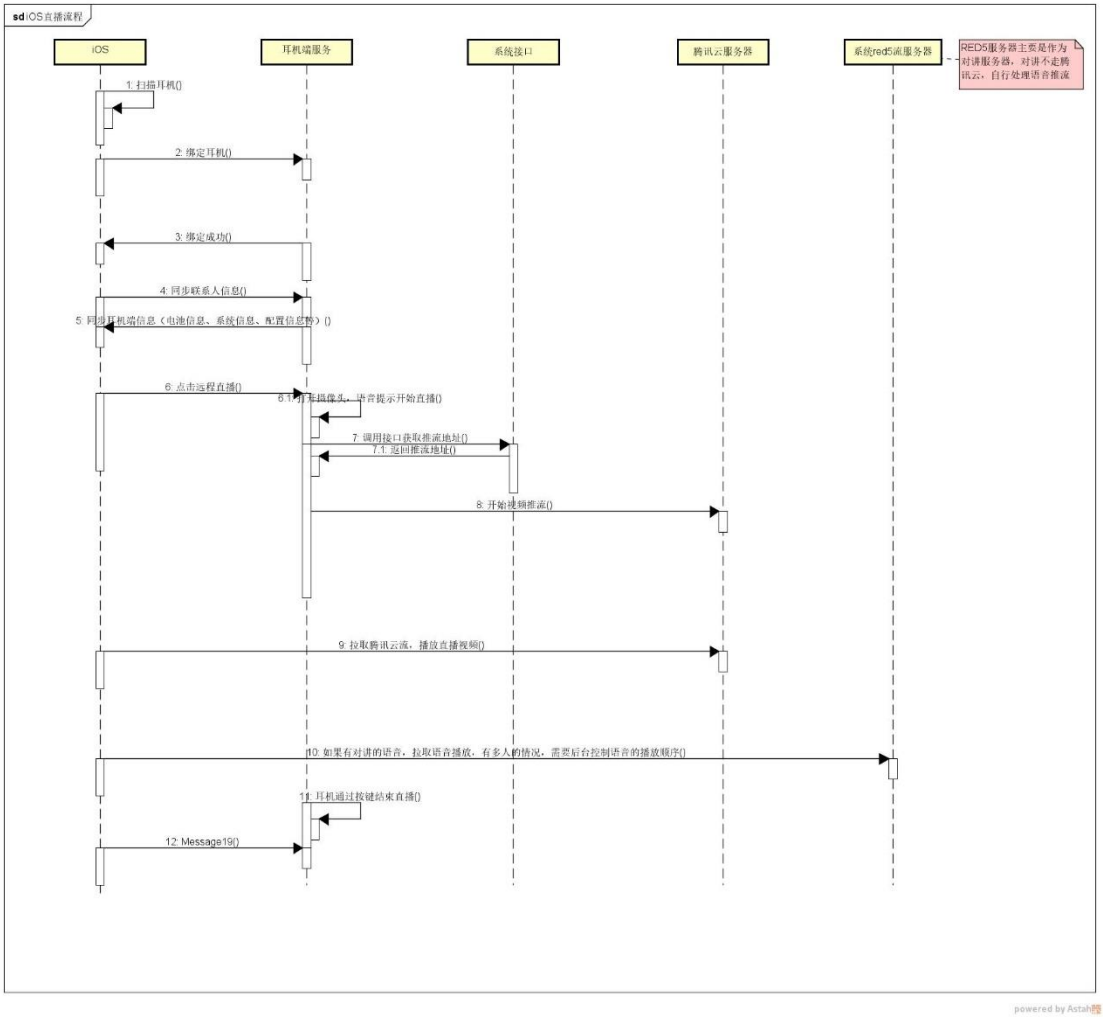
iOS 开发文档

一、iOS 处理流程概述

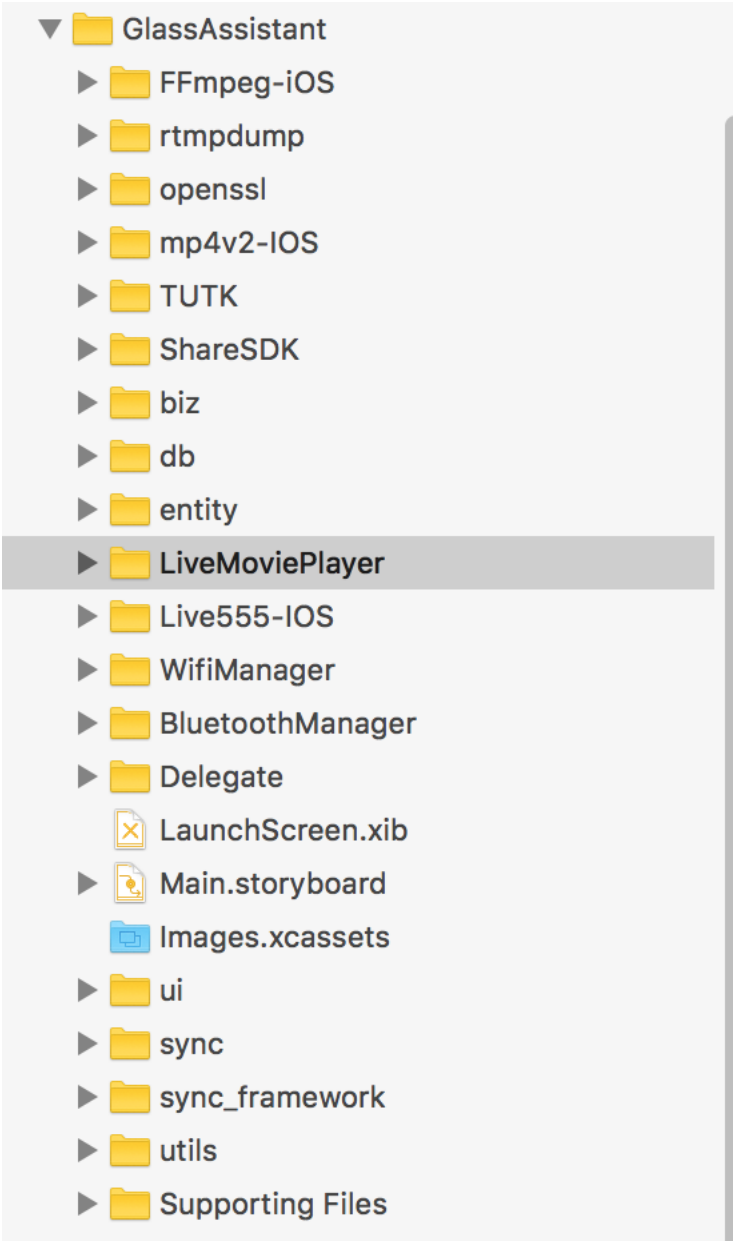
iOS 端 App 核心的地方在于通过蓝牙与智能耳机的通信问题，直播部分直接通过拉取流进行视频的推送，播放时通过拉取推送地址进行直播流的播放。iOS 整体的操作流程如下：



说明：1、腾讯云相关的是通过后台系统进行管理，包括直播房间失效，在线房间管理；2、red5 服务器主要是用来管理直播中语音互动的部分，语音走的是自行搭建的 red5 流服务器（也可以采用熟悉的流服务器替换掉腾讯云服务器）。

二、代码核心部分

项目使用 Objective-C 开发，源码目录如下：



代码结构目录中，已经完成基础的蓝牙、wifi 封装。Demo 中采用 storyboard 的方式进行页面布局，页面结构在 Main.storyboard 中。

FFmpeg-iOS 处理视频编码解码相关

Rtmpdump 处理 rtmp 推流相关

TUTK 处理视频在线播放

GAMainViewController---app 主页面

在主页面中，开启 WiFi 以及蓝牙的操作，会自动同步数据，具体结合代码，

```
self.wifiManager = [GAWifiManager shareInstance];
    [self.wifiManager resumeListenWifiState];
    self.bluetoothManager = [GABluetoothManager
shareInstance];
    self.btStateAlertView = [[UIAlertView
alloc]initWithTitle:NSLocalizedString(@"str_alertTitle_info", nil)
message:nil delegate:self cancelButtonTitle:nil

otherButtonTitles:NSLocalizedString(@"str_alertSetButton_title",
nil), nil];
    // 监听进入退出后台事件
    [[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(applicationWillResignActive:)

name:UIApplicationWillResignActiveNotification object:nil];
    [[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(applicationDidBecomeActive:)

name:UIApplicationDidBecomeActiveNotification object:nil];
```

GAIOTCLiveLoginViewController---功能选择

Demo 中该页面提供远程在线直播、本地直播、获取 rmtp 推荐播放三个功能。
在线直播是由 iOS 端发起远程在线直播，耳机开启摄像头推流到服务器。

本地直播不通过腾讯云，直接与手机端连接，播放。

RMTP 获取正在直播的房间流地址，点击观看直播。

主要用到以下三个 Class

GAIOTCLiveLoginViewController----管理页面，提供远程直播、本地直播、rmtp 推流功能

IOTCLiveViewController-----直播页面

--- GASearchQRViewController –直播页面通过二维码扫描发起直播

--- GALiveSelectUIDViewController-通过 UID 输入方式发起直播

--- GARTMPSpeakerViewController –直播中对讲页面

GAMultiMediaVideoItemViewController---耳机视频同步页面

视频同步页面主要流程是通过蓝牙以及 WiFi，耳机与 app 处于同一网络中，通过 WiFi 数据同步视频，前提是需要先通过蓝牙将 WiFi 做好配置。在开发过程中需要考虑到 App 上架的问题，wifi 部分是需要申请苹果额外证书。

视频同步完成后，可以对视频进行管理，删除，同步删除耳机上的视频或者单独删除同步到 app 中的视频，详细流程参考代码实现

GAMultiMediaPictureItemViewController---耳机图片同步页面

--- GAPicturePreviewViewController 图片预览，点击图片进行预览。

同步完耳机上页面也可与视频操作一致，对图片进行删除。

GAPersonalCenterViewController---耳机设置页面

耳机设置页面有调用接口耳机信息拍照，开始录制视频

对应的按钮事件：

```
#pragma mark   拍照

- (IBAction)takePictureAction:(id)sender {
```

```
    [[[GAPhotoModule alloc] initWithName:
PHOTO_MODULE_NAME] send_take_photo];
    self.btnTakePicture.enabled = NO;
    [self.btnTakePicture setBackgroundImage:[UIImage
imageNamed:@"btn_takephoto_select.png"]
forState:UIControlStateDisabled];
    [self.handler sendMessage:
MSG_BTNTAKEPICTURE_ENABLED delay:
TIME_ENABLE_MEDIABTNS];
}
```

对应的详细的执行过程参考项目中代码。

```
#pragma mark  录像
- (IBAction)recordAction:(id)sender {
    [[[GAPhotoModule alloc] initWithName:
PHOTO_MODULE_NAME] send_record];
    [self.btnRecord setBackgroundImage:[UIImage
imageNamed:@"btn_videorecord_select.png"] forState:1];
    self.btnRecord.enabled = NO;
    [self.handler sendMessage: MSG_BTNRECORD_ENABLED
delay: TIME_ENABLE_MEDIABTNS];
}
```

GASettingHotSpotViewController---设置 wifi 热点，同步到耳机端

```
/**设置按钮点击事件*/
- (IBAction)btnSetAction:(id)sender {

    // 保存用户输入的密码，下次自动填写。
    NSUserDefaults *userDefaults = [NSUserDefaults
standardUserDefaults];
    [userDefaults setValue: self.txtPwd.text forKey:
MOBILE_AP_PWD];
    [userDefaults synchronize];

    //获取本级的昵称，即个人热点的ssid。
    NSString* ssid = [[UIDevice currentDevice] name];
    [[GAWifiModule shareInstance] sendAPName:ssid
andPwd:self.txtPwd.text andIsphoneAp:YES];
    /*
     * 获取屏幕状态栏的高度 ：当高度 = 20 时说明没
     * 有开启个人热点，当高度= 40时，说明开启个人热点。
     * 若开启已经开启热点则不跳转到蜂窝网络设置界面。
     */
```

```
    CGRect rectStatus = [[UIApplication sharedApplication]
statusBarFrame];
    if(rectStatus.size.height != 40){
        [[UIApplication sharedApplication] openURL
         :[NSURL
URLWithString:@"prefs:root=MOBILE_DATA_SETTINGS_ID"]];
    }
    [[AVAudioSession sharedInstance] setActive:YES error:nil];
    [self dismissViewControllerAnimated: YES completion: nil];
}
```

GASettingLanguageViewController---设置语言

```
_rowARYLanguage = [[NSMutableArray alloc]
initWithObjects:NSLocalizedString(@"str_language_itemTitle_zh"
, nil)
                        ,NSLocalizedString(@"str_language_item
Title_en", nil)
                        ,NSLocalizedString(@"str_language_item
Title_ar", nil)
                        ,NSLocalizedString(@"str_language_item
Title_de", nil)
```

```
                        ,NSLocalizedString(@"str_language_item
Title_es", nil)
                        ,NSLocalizedString(@"str_language_item
Title_fa", nil)
                        ,NSLocalizedString(@"str_language_item
Title_fr", nil)
                        ,NSLocalizedString(@"str_language_item
Title_it", nil)
                        ,NSLocalizedString(@"str_language_item
Title_pt", nil)
                        ,NSLocalizedString(@"str_language_item
Title_ru", nil)
                        ,NSLocalizedString(@"str_language_item
Title_th", nil)
                        ,nil];
    GAAboutModule *aboutModule = [GAAboutModule
shareInstance];
    aboutModule.didRetriveLanguage = ^(int language) {
        _currLanguage = language;
        [self.tableViewLanguage reloadData];
```

完成设置发送请求：

```
#pragma mark  完成按钮发送更改语言请求
```

```
- (void)EnsureChange{
    if(_currLanguage == (_currSelectRow+1)) return;
    GALanguageModule *languageModule =
[GALanguageModule shareInstance];
    [languageModule
sendChangeRequest:(int)(_currSelectRow+1)];
    [self dismissViewControllerAnimated: YES completion: nil];
}
```

蓝牙发送修改语言数据核心代码：

```
- (BOOL)send:(GASyncData *)data {
    data.moduleName = self.name;
    GASerialData *sd = [data getSerialDatas];
    NSLog(@"syncData len:%d bytes:%p", sd.len, sd.data);
    NSData *adata = [[NSData alloc] initWithBytes:sd.data
length:sd.len];
    if(sd.len <= 150){
        return [self.bluetoothManager writeData:adata];
    }else{
        NSString *uuid = [GAUtils getUUIDString];
        NSMutableData *da = [[NSMutableData alloc] init];
```

```
            [da appendData:[uuid
dataUsingEncoding:NSUTF8StringEncoding]];
            [da appendBytes:sd.data length:sd.len];
            uLong crc = crc32(0L, Z_NULL, 0);
            //生成32位crc校验码
            crc = crc32(crc, [da bytes], da.length);
            NSString *crcString = [NSString
stringWithFormat:@"%lx",crc];
            while (crcString.length < LONGDATA_CRC_LENGTH) {
                crcString = [NSString
stringWithFormat:@"0%@",crcString];
            }
            NSLog(@"crc32:%@,",crcString);

            //count 包数目
            int count = sd.len/LONGDATA_EFFECTIVE_LENGTH
+(sd.len % LONGDATA_EFFECTIVE_LENGTH > 0? 1 : 0);
            int countBool =0;
            NSLog(@"count=%d",count);
            Byte index[2] = {0};
            index[0] = (Byte)count;
```

```
            for(int i = 0; i < count;i++){
                NSMutableData *sendData = [[NSMutableData
alloc] init];
                //前八位是校验码,然后是包数量，包序号（0开始）
                [sendData appendData:[crcString
dataUsingEncoding:NSUTF8StringEncoding]];
                index[1] = i;
                [sendData appendBytes:index length:2];
                if(i != count -1){
                    [sendData appendData:[adata
subdataWithRange:NSMakeRange(LONGDATA_EFFECTIVE_LENG
TH*i, LONGDATA_EFFECTIVE_LENGTH)]];
                }else{
                    [sendData appendData:[adata
subdataWithRange:NSMakeRange(LONGDATA_EFFECTIVE_LENG
TH*i, sd.len-LONGDATA_EFFECTIVE_LENGTH*i)]];
                }
                NSLog(@"sendData 3 length:%d",sendData.length);
                if([self.bluetoothManager writeData:sendData]){
                    countBool ++;
                }
            }
```

```
            if(countBool == count)return YES;
            return NO;
        }
}
```

GASettingWifiViewController----设置耳机端 WLAN

```
_wifiModule.didRetriveWifiClose = ^(){
            [self.handler
removeMessages:MSG_CLOSE_WIFI_TIMEOUT];
            // 点击关闭 WLAN 按钮后通知 WifiManager 眼镜端已断
开 WLAN，否则 WifiManager 会认为眼镜端还处于 CONNECT_OK
状态。
            [self.wifiManager didGlassConnecteToAPStateChanged:
DISCONNECT extra: nil ssid:nil];
            [self refreshTableViewConnectSsid:@""];
            [self
refreshAlertViewMessage:NSLocalizedString(@"str_alertmsg_wifi
_close", nil)];
        };
        [self.handler removeMessages:MSG_CLOSE_WIFI_TIMEOUT];
```

```
    [self.handler sendMessage:MSG_CLOSE_WIFI_TIMEOUT
delay:CLOSE_WIFI_TIMEOUT_SECS];
    [_wifiModule closeWifi];
```

GAGuideViewController---耳机绑定

设置功能部分比较重要的地方，耳机绑定方式可以通过二维码方式扫描进行绑定，也可以通过一键操作完成绑定。

扫描操作：

```
#pragma mark 扫描绑定点击事件
- (void)onClickScanBind {
    if (![self.bluetoothManager isBtON]){
        [self.alertOpenBT show];
    }else
        [self performSegueWithIdentifier: @"showScanQRCode"
sender: self];
}
#pragma mark 扫描结果处理
- (void)scanQRResult:(NSString *)result{
    if(nil == result){
        [self.imageGlassCircle setHidden:YES];
        [self.imageScanBinding setImage:[UIImage
imageNamed:@"bind_fail.png"]];
        [self.imageScanBinding setHidden:NO];
```

```
            [self.bindState setHidden:NO];
            self.bindState.text =
NSLocalizedString(@"str_item_state_title_scan_failure", nil);
        }else{
            self.scanQRResult = [NSString
stringWithFormat:@"%@-nc",[result substringFromIndex:1]];
            [self startBindingAnimation:YES];
            //扫描蓝牙设备
            [self.bluetoothManager startScan];
            [self.handler sendMessage:MSG_SCAN_TIMEOUT
delay:SCAN_TIMEOUT_SECS];
        }
}
```

```
#pragma mark 扫描到蓝牙设备
- (void)didDiscoverDevice:(GABluetoothDevice *)device {
    GALog(@"didDiscoverPeripheral
//////////////////device=%@",device.name);
    if(self.scanQRResult != nil){
        if([self.scanQRResult isEqualToString:device.name]){
            self.bindState.text =
NSLocalizedString(@"str_item_state_title_binding", nil);
```

```
                [self.bluetoothManager stopScan];
                [self.bluetoothManager bind:device];
                self.scanQRResult = nil;
            }
        }else {
            if (BIND_FIT_ALLDEVICEPREFIX_SWITCH) {
                NSString *deviceName = [device.name
uppercaseString];
                for (NSString *prefixName in self.bluetoothNames) {
                    if ([deviceName hasPrefix:prefixName] &&
[device.name hasSuffix:BIND_BLUETOOTH_NAME_SUFFIX]){
                        self.bindState.text =
NSLocalizedString(@"str_item_state_title_binding", nil);
                        [self.bluetoothManager bind:device];
                        [self.bluetoothManager stopScan];
                    }

                }
            } else {
                if ([device.name
hasPrefix:BIND_BLUETOOTH_NAME_PREFIX] && [device.name
hasSuffix:BIND_BLUETOOTH_NAME_SUFFIX]){
```

```
                    self.bindState.text =
NSLocalizedString(@"str_item_state_title_binding", nil);
                    [self.bluetoothManager bind:device];
                    [self.bluetoothManager stopScan];
                }
            }
        }
}
```

扫描完成后，连接到设备：

```
#pragma mark 连接某个设备
- (void)didConnectDevice:(GABluetoothDevice *)device {
    [self.handler removeMessages:MSG_SCAN_TIMEOUT];
    [self goToBindView];
//      [self syncSetDefaultState];
    // 绑定时进行联系人同步
    GABLContacts *blContacts = [GABLContacts shareInstance];
    [blContacts initialSyncContacts];
    [blContacts syncContacts:YES];
}
```

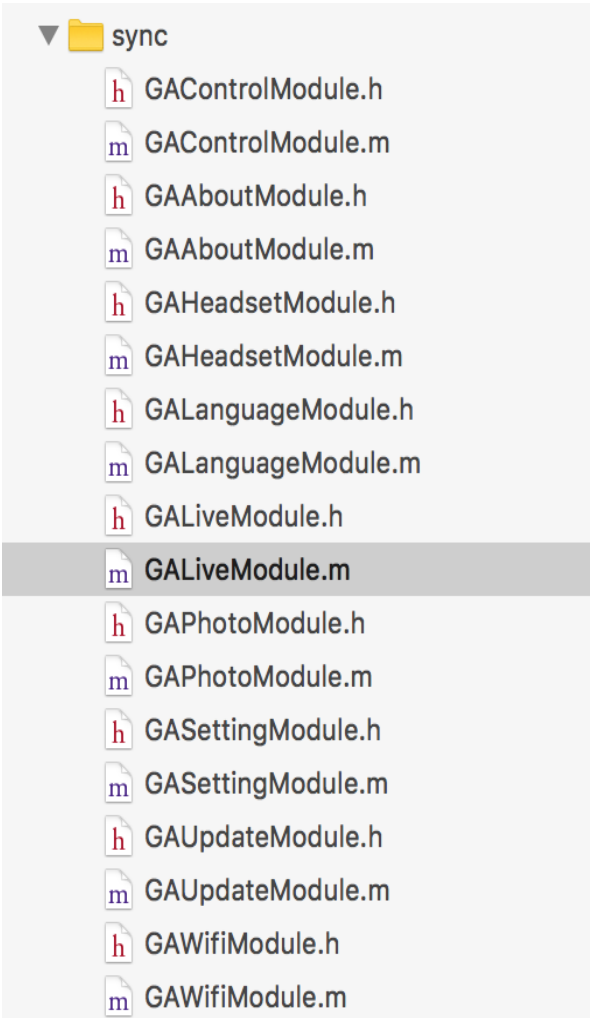
```
#pragma mark 绑定眼镜失败回调
- (void)didFailToConnectDevice:(GABluetoothDevice *)device
error:(NSError *)error {
```

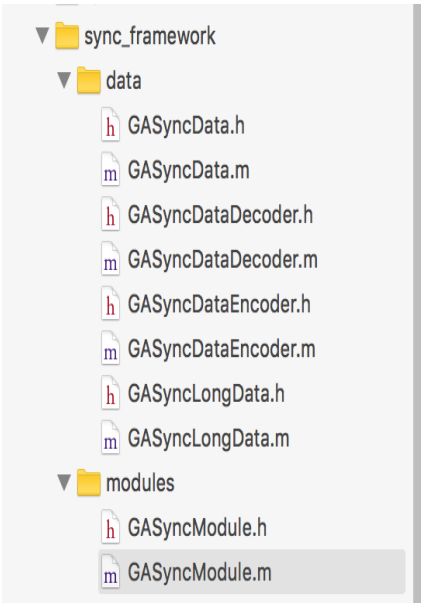
```
        [self startBindingAnimation:NO];
        self.bindState.text =
NSLocalizedString(@"str_item_state_title_bind_failure", nil);
        if (error && [error.domain isEqualToString:@"bond_error"]) {
            if (error.code == BOND_ERROR_GLASS_HAS_BONDED) {
                NSLog(@"对端眼镜已绑定其他手机");


                return;
            } else if (error.code == BOND_ERROR_TIMEOUT) {
                NSLog(@"与眼镜连接超时");
                return;
            }
        }
}
```

设置功能其他相关的包括水印设置，获取耳机信息，蓝牙通话设备，这部分参考代码实现，较为简单。

三、封装核心处理代码类

蓝牙/WIFI 同步拉取数据相关





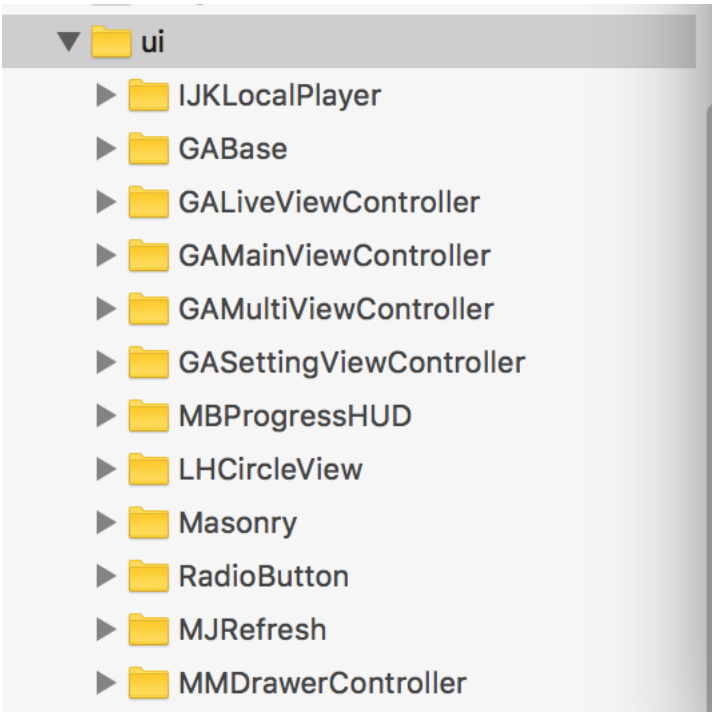
简单说明：

sync Group 下是与设备端通信的核心的代码

GALiveModule 为处理直播相关的，其他是相应功能的通信部分，具体代码就不贴了，参考 Demo 源码。

Sync_framework 是基础代码，这部分参考源码。

UI 部分



UI 源码部分核心直播页面（GALiveViewController），视频图片同步页面（GAMultiViewViewController），设置页面（GASettingViewController），