实时音视频

客户端基本功能

产品文档

腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2018 腾讯云版权所有

本文档著作权归腾讯云单独所有，未经腾讯云事先书面许可，任何主体不得以任何形式复制、修改、抄袭、传播全部或部分本文档内容。

【商标声明】

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。

【服务声明】

本文档意在向客户介绍腾讯云全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或模式的承诺或保证。

文档目录

版权所有：腾讯云计算（北京）有限责任公司

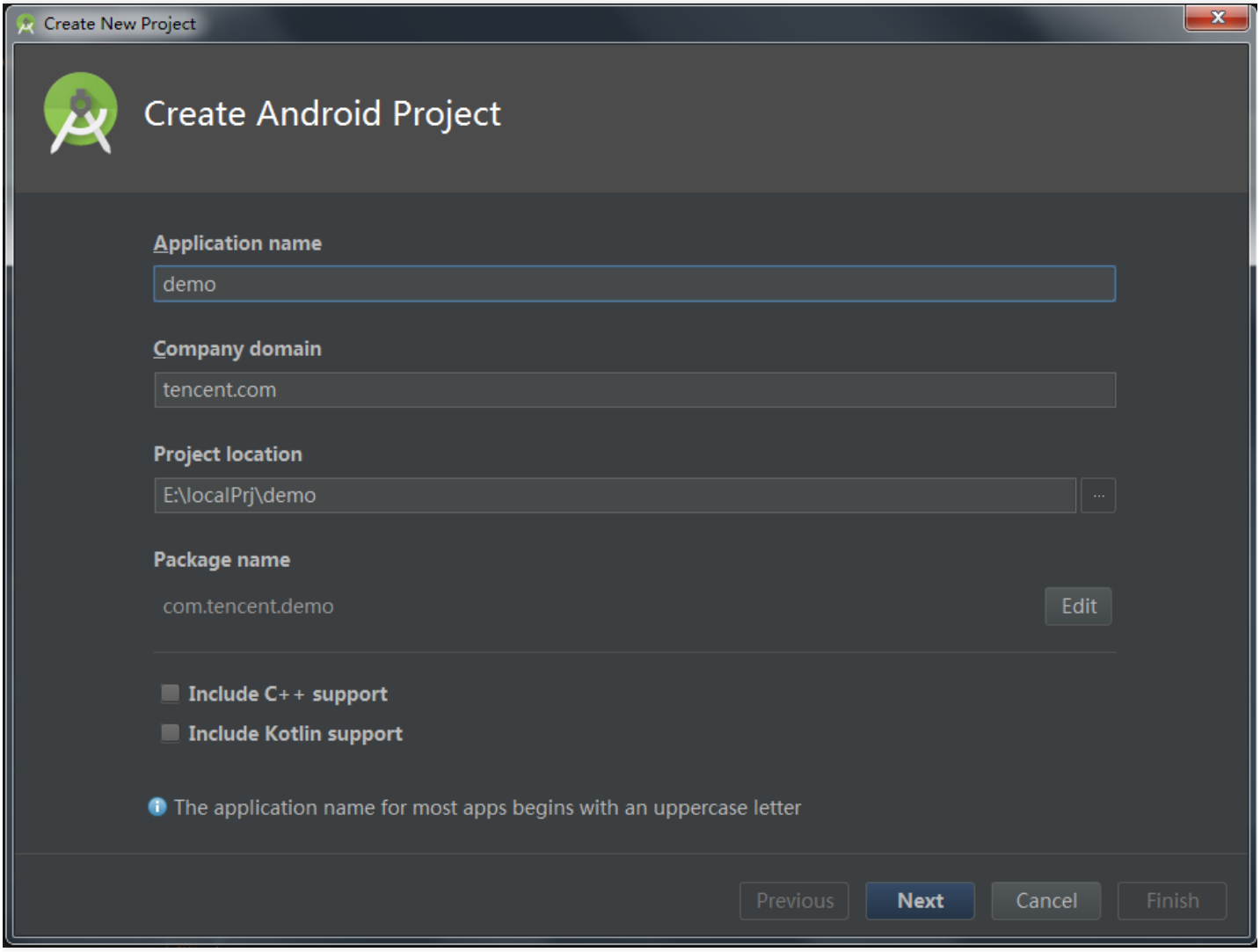
客户端基本功能
集成 SDK
集成 SDK（Android）

最近更新时间：2018-09-28 16:58:43

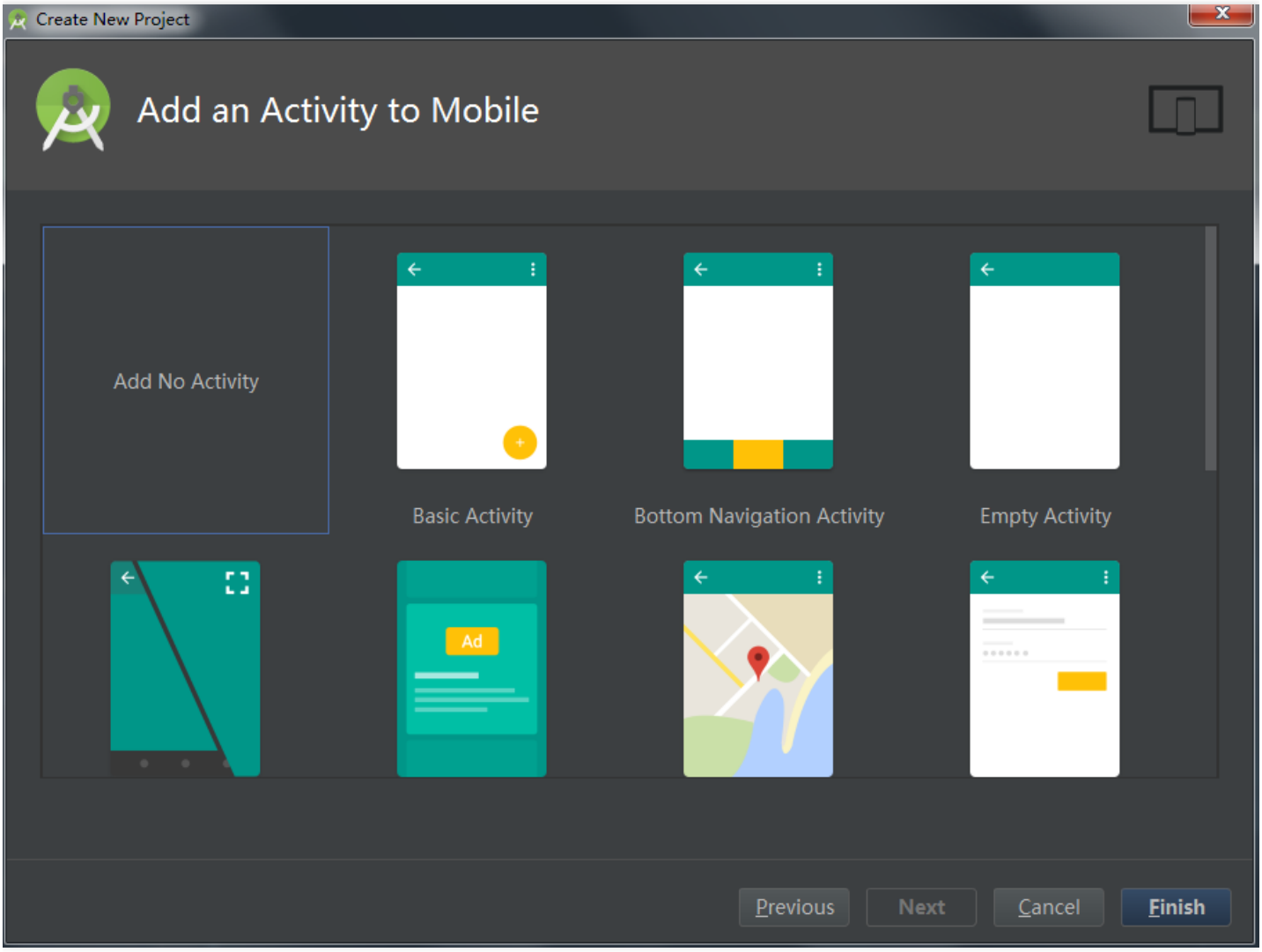
操作步骤

创建一个 Android 工程

打开 Android Studio，单击【File】菜单选择【New Project】新建一个工程：



创建一个空工程：



添加依赖(集成 SDK)

修改 build.gradle 文件，在 dependencies 中添加 iLiveSDK 的依赖：

```
compile 'com.tencent.ilivesdk:ilivesdk:latest.release' //其中latest.release指代最新iLiveSDK版本号
```

最新版本说明

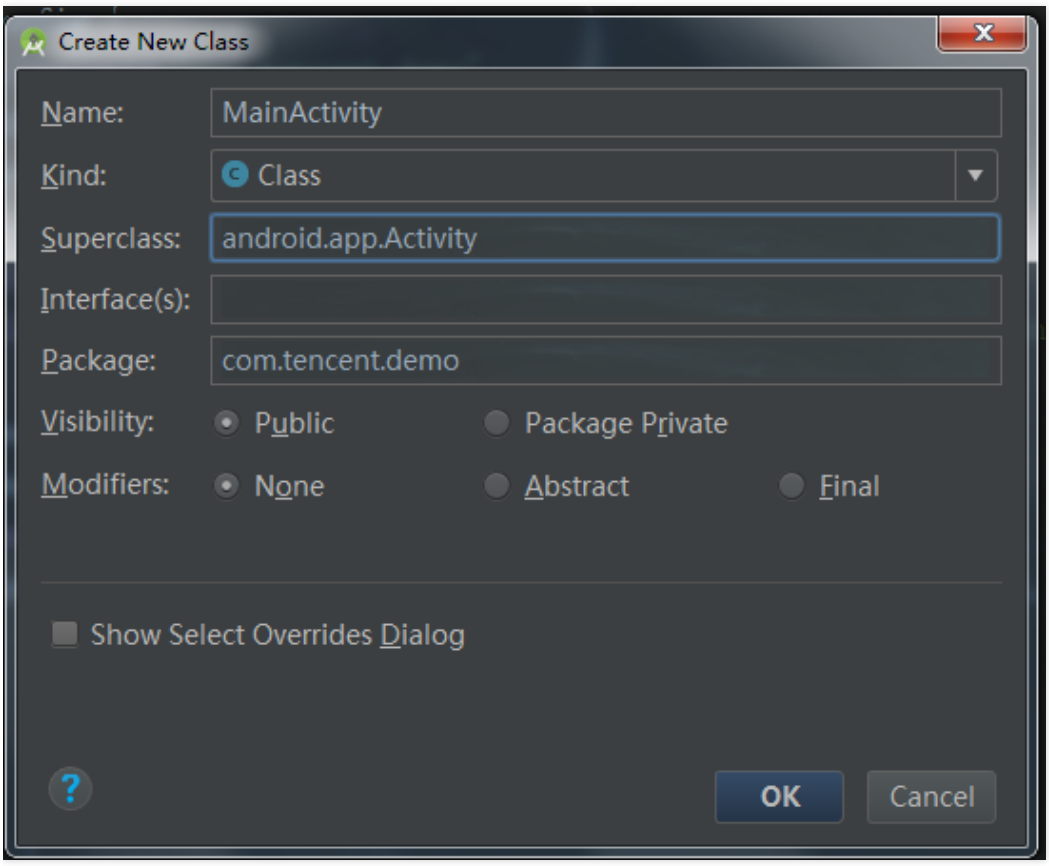
创建一个应用

在 layout 中创建一个简单布局 main.xml 资源：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent">
```

```
<TextView
android:id="@+id/tv_version"
android:textColor="@color/colorAccent"
android:layout_width="match_parent"
android:layout_height="wrap_content" />
</LinearLayout>
```

同时创建一个 Activity 应用：



在应用创建中使用布局并输出 iLiveSDK 的版本号：

```
public class MainActivity extends Activity {
private TextView tvVersion;

@Override
protected void onCreate(@Nullable Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
tvVersion = (TextView)findViewById(R.id.tv_version);

tvVersion.setText(" iLiveSDK: "+ILiveSDK.getInstance().getVersion()+"\n IMSDK:"+
TIMManager.getInstance().getVersion()+"\n AVSDK:"+
AVContext.sdkVersion);
```

```
}
}
```

声明应用

在 AndroidManifest.xml 中声明应用：

```
<activity android:name=".MainActivity" android:screenOrientation="portrait" >
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
```

编译运行

编译运行，可以看到效果：

```
iLiveSDK:1.8.5
IMSDK:2.5.6.11073.11080
AVSDK:1.9.8.2
```

恭喜，至此说明 iLiveSDK 已经成功集成。

常见问题

下载 aar 失败

```
Error:Could not resolve all files for configuration ':app:debugCompileClasspath'.
> Could not resolve com.tencent.ilivesdk:ilivesdk:1.8.3.
Required by:
project :app
> Could not resolve com.tencent.ilivesdk4:ilivesdk:1.8.3.
> Could not get resource 'https://jcenter.bintray.com/com/tencent/ilivesdk/ilivesdk/1.8.4/ilivesdk-1.8.3.pom'.
> Could not GET 'https://jcenter.bintray.com/com/tencent/ilivesdk/ilivesdk/1.8.4/ilivesdk-1.8.3.pom'.
> Connect to jcenter.bintray.com:443 [jcenter.bintray.com/75.126.118.188] failed: Connection timed out: connect
```

先检测网络是否正常，并通过上面链接，确认可以访问 jcenter 网站，同时如果网络需要代理检测是否有在 gradle.properties 中配置。

混淆导致方法找不到

由于内部有一些接口调用需要，在用户工程需要混淆时，请添加以下配置：

```
-keep class com.tencent.**{*;}
-dontwarn com.tencent.**

-keep class tencent.**{*;}
-dontwarn tencent.**

-keep class qalsdk.**{*;}
-dontwarn qalsdk.**
```

多架构导致Crash

目前只支持 armeabi 架构(1.0.5 版本之后支持 arm-v7a)，如果工程(或依赖库)中有多架构，需要在 build.gradle 中添加以下配置：

```
android{
defaultConfig{
ndk{
abiFilters 'armeabi', 'armeabi-v7a'
}
}
}
```

联系邮箱

如果对上述文档有不明白的地方，请反馈到trtcfb@qq.com

集成 SDK（iOS）

最近更新时间：2018-10-09 10:03:01

本文将指导您完成在 iOS 端下实时音视频客户端功能的 SDK 集成。

源码下载

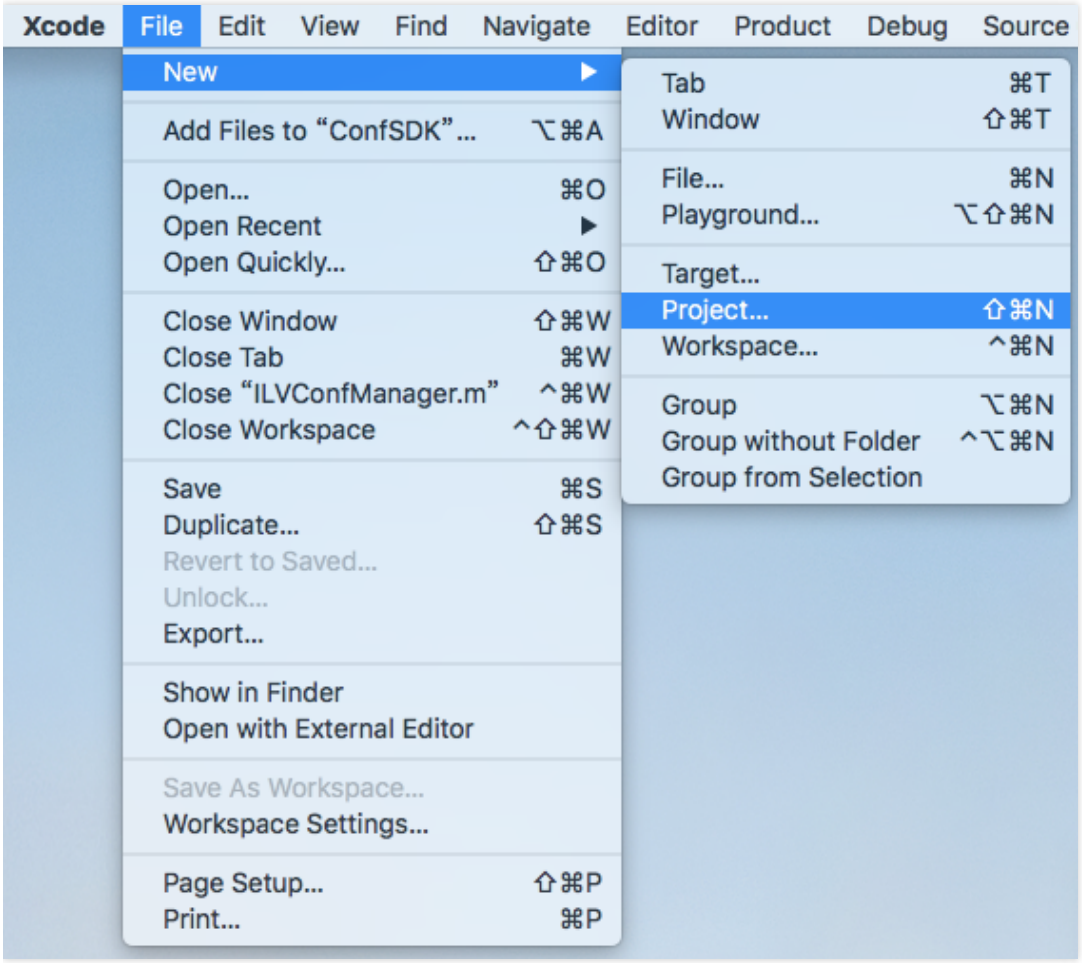
在此我们提供以下所讲到的完整 Demo 代码，如有需要请您自行下载。
Demo 代码下载

操作步骤

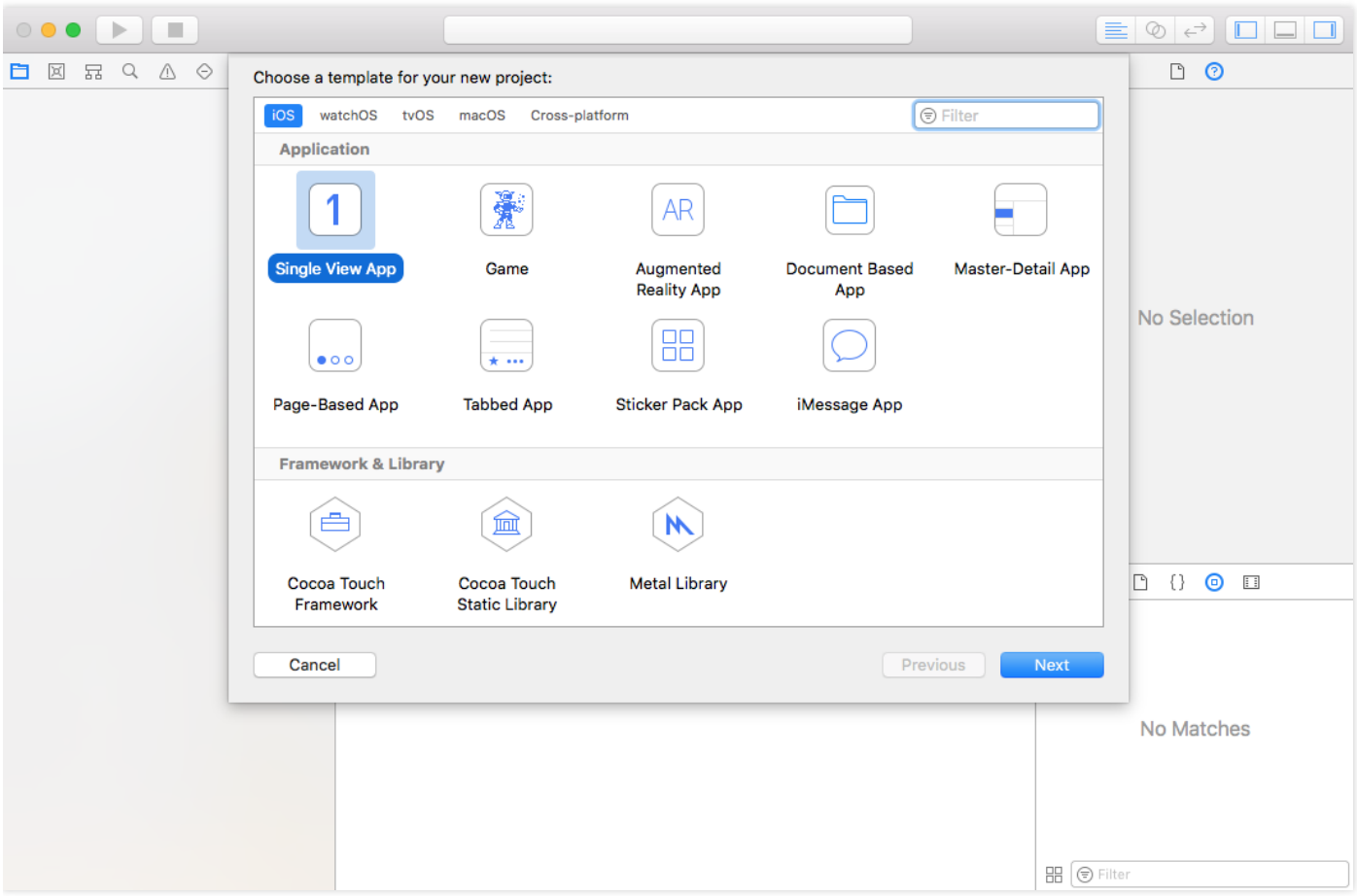
创建 iOS 工程

> 如果您已经有一个工程待集成，请直接跳到下一步 集成 SDK。

首先，使用 Xcode 创建一个新的工程用来集成我们的 SDK。
打开 Xcode，【 File 】-【 New 】-【 Project 】：



选择【 Signle View App 】



设置工程名为 Demo01_集成 SDK，语言选择为 Objective-C，Team、Organization Name 和 Organization Identifier 根据自身情况填写（也可随便填写）,然后选择下一步选择项目存放地址，单击 【 create 】即可 。

集成 SDK

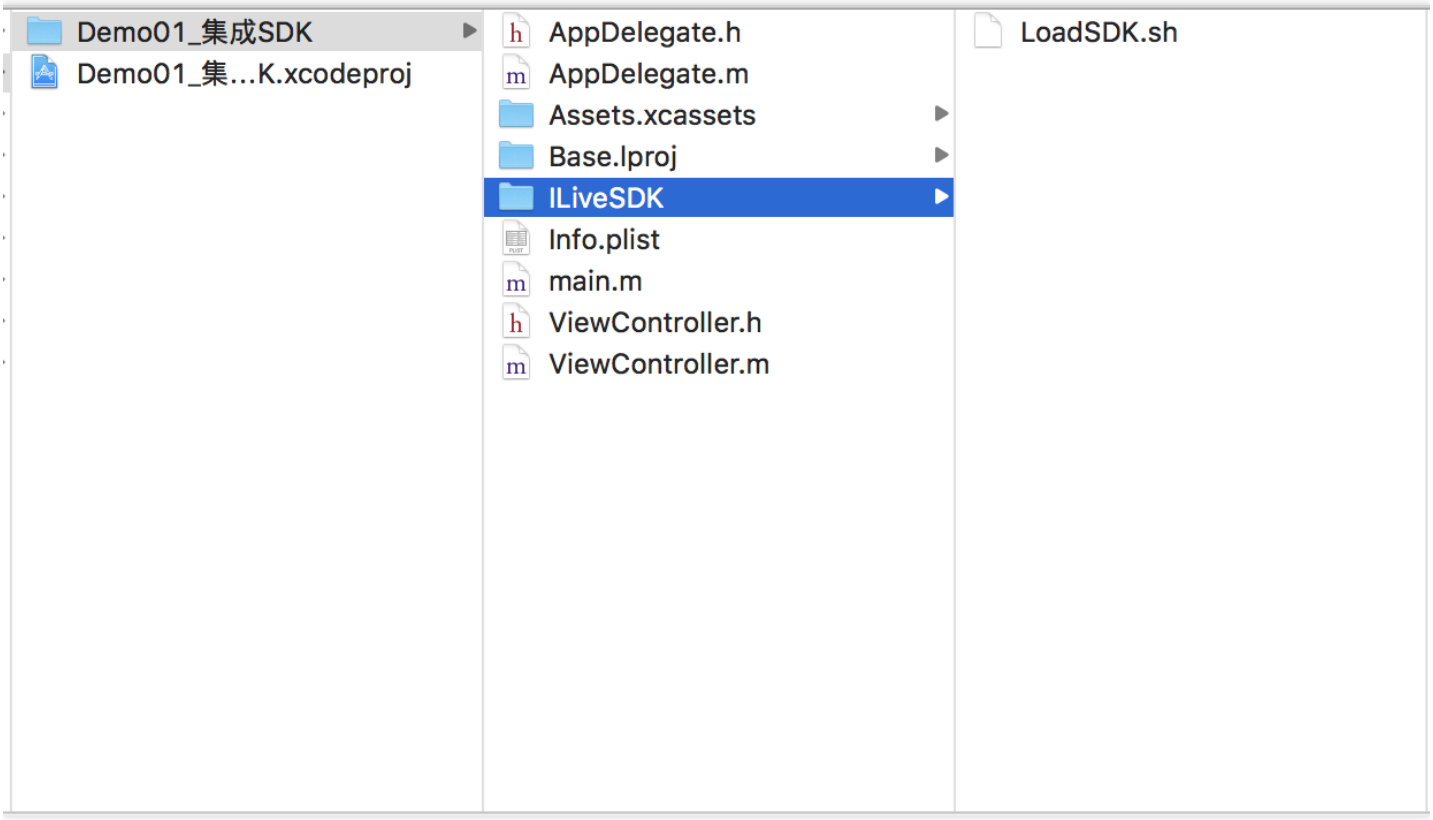
获取SDK

ILiveSDK 其实是一套SDK的集合，其中包含了以下一些子 SDK：

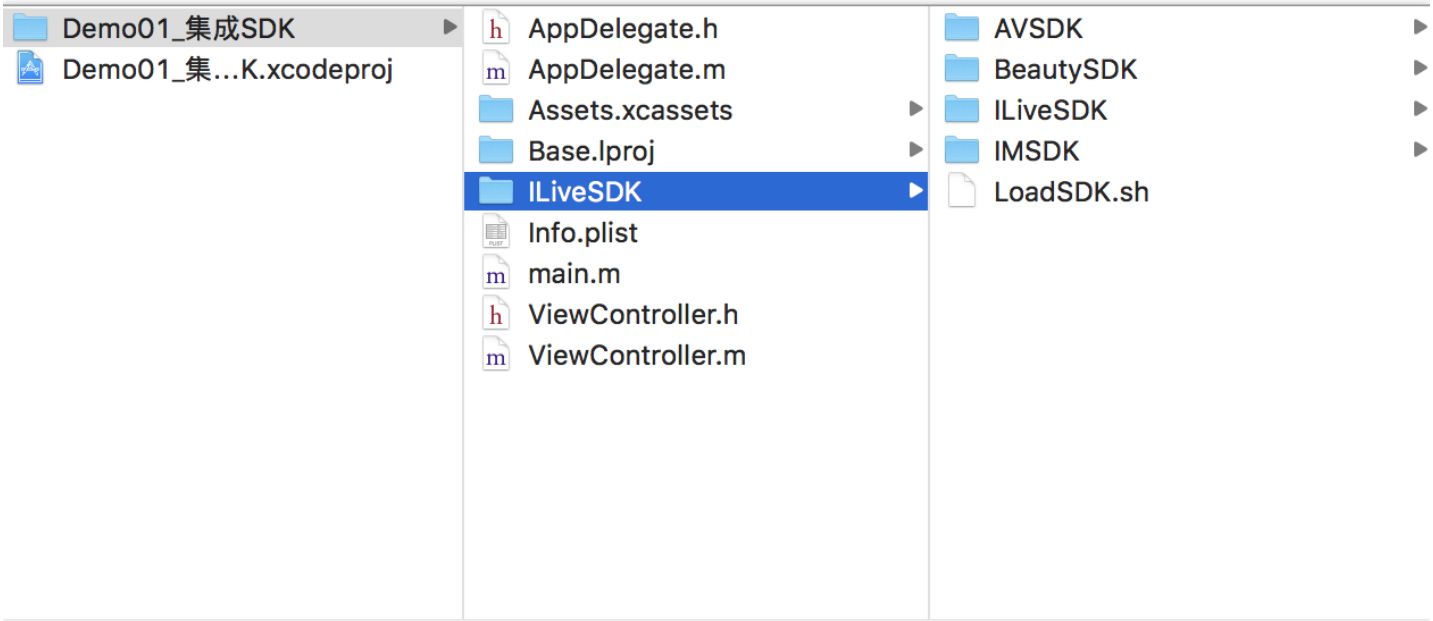
- BeautySDK：提供美颜预处理功能
- IMSDK：提供 IM 即时通信功能
- AVSDK：提供底层音视频功能
- ILiveSDK：在 AVSDK 基础上封装而成，提供更简单易用的音视频功能接口
- TILLiveSDK：在 ILiveSDK 的基础上，针对直播场景相关接口进行的封装，方便快速实现直播相关功能

我们先在工程目录中新建一个名为 ILiveSDK 的文件夹，用来存放我们的 SDK，由于 ILiveSDK 包含若干个子 SDK，所以我们提供了一个 SDK 下载脚本方便获取所有的SDK。

单击下载脚本，将其放置于刚才创建的 ILiveSDK 文件夹下：



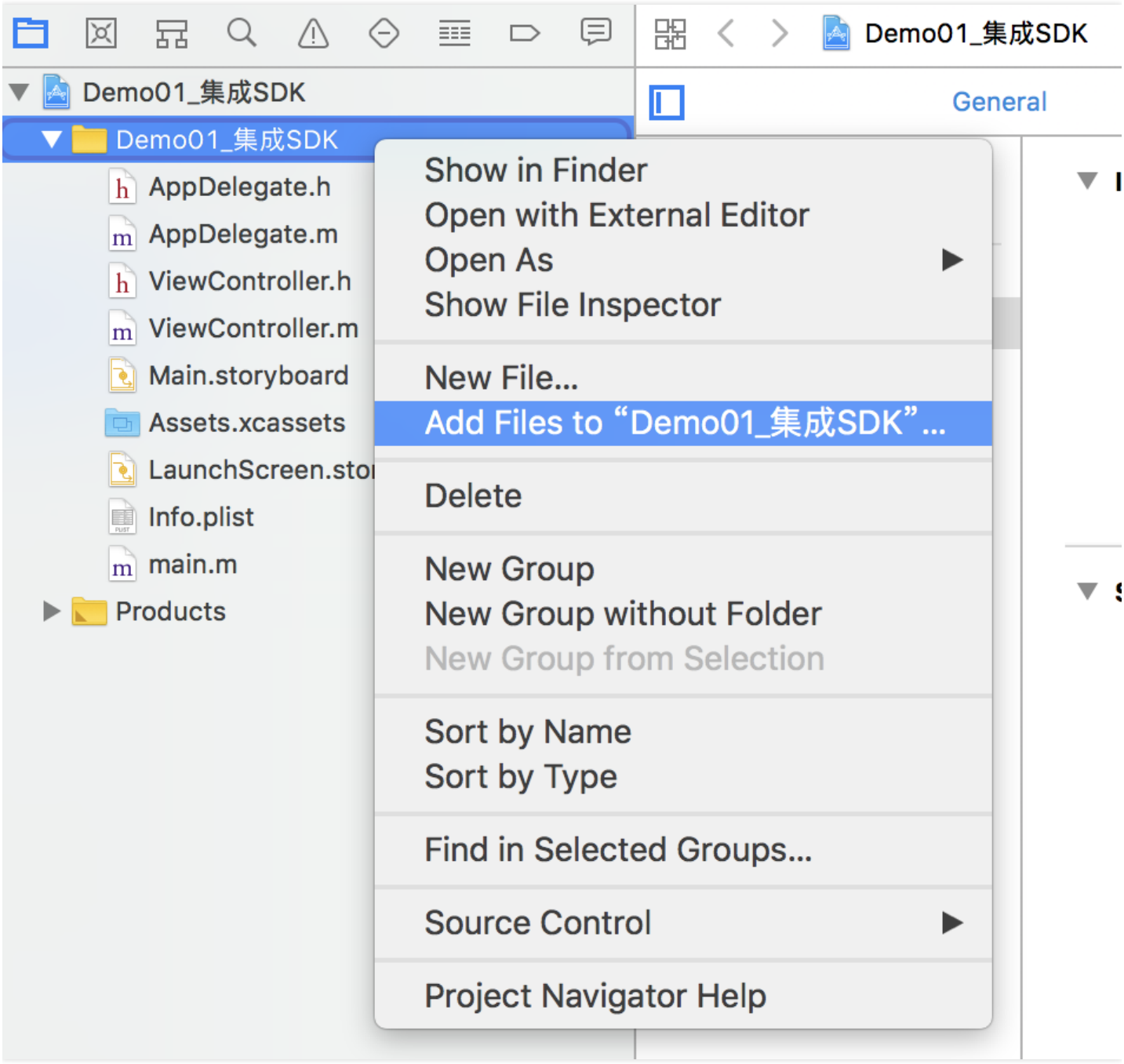
运行下载脚本（ 打开终端，cd 命令进入 Frameworks 目录下，运行命令 `sh LoadSDK.sh` ），就会自动下载所有SDK，下载完成之后会自动解压，并删除下载的压缩包，稍等片刻即可，解压完成之后文件目录如下：



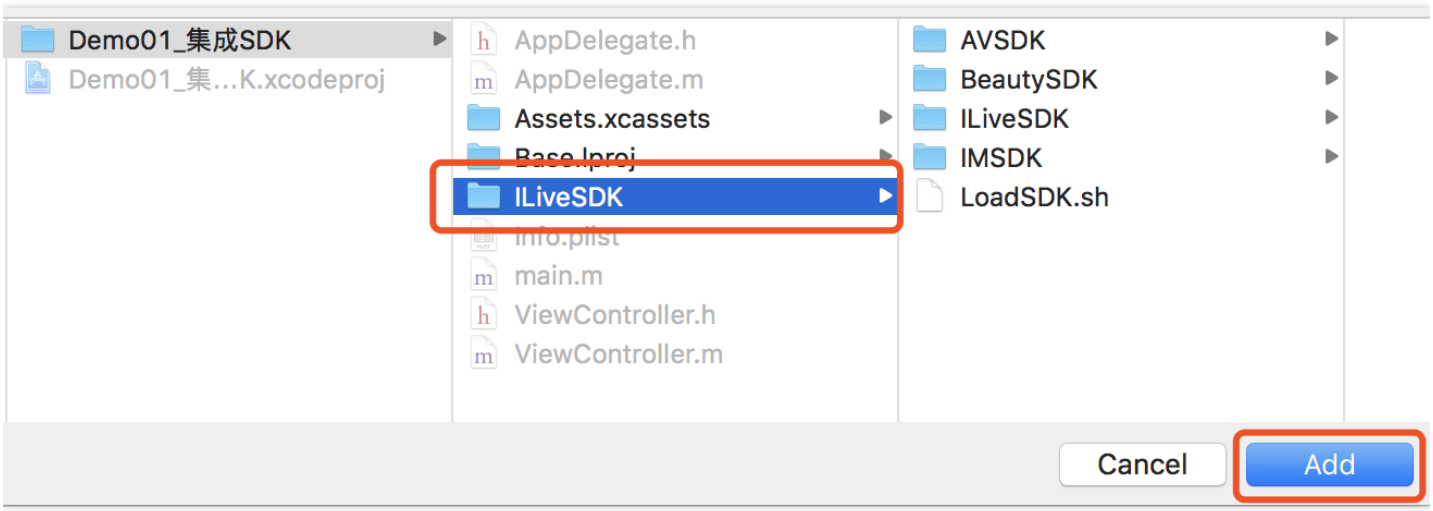
最新版本说明

导入SDK

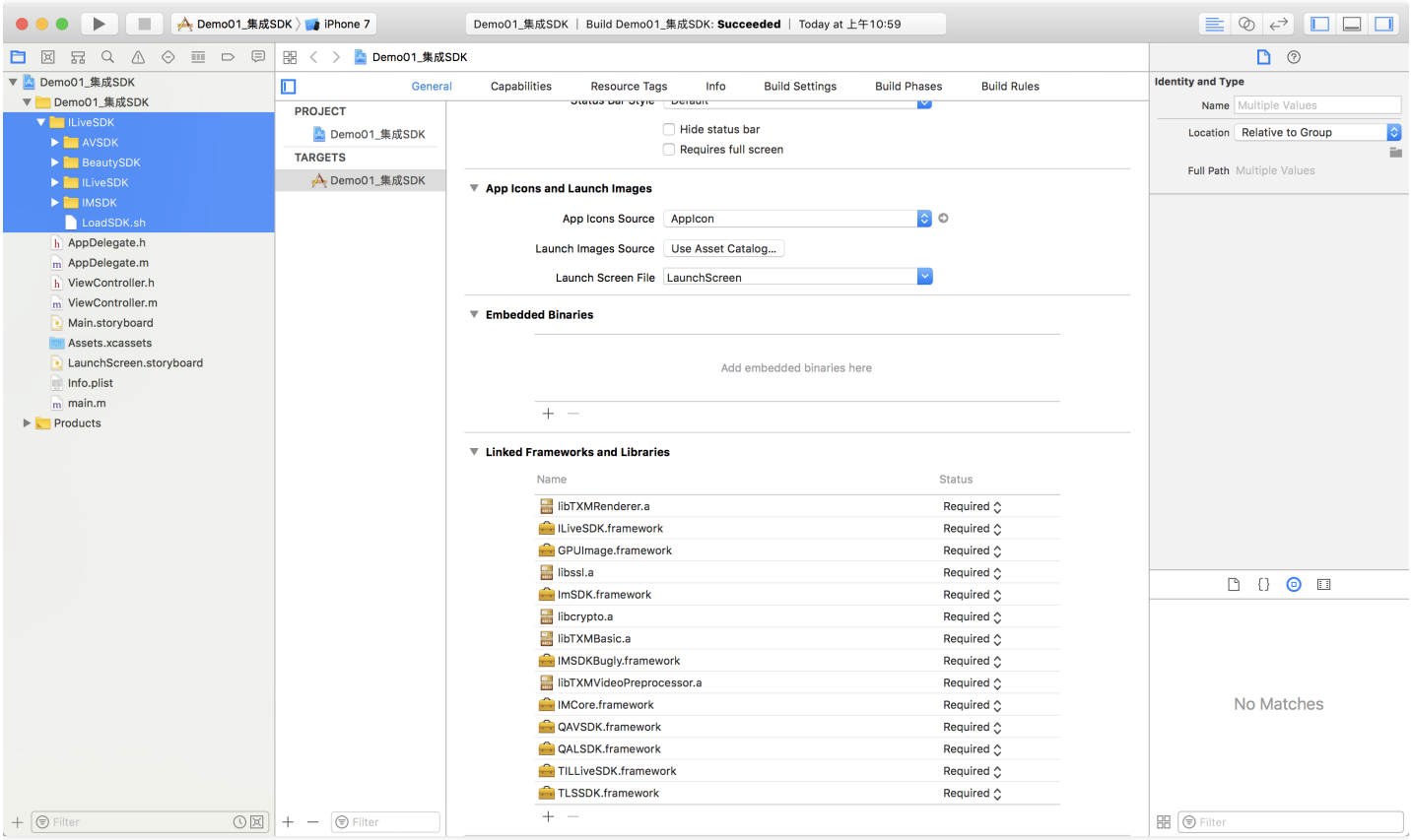
下载完成后，我们需将 SDK 导入工程，在工程根目录上单击右键->【 Add Files to "Demo01_集成SDK" 】：



在弹出的目录选择框中选择我们刚刚创建的【 ILiveSDK 文件夹 】->【 Add 】：



添加完成后的工程目录：

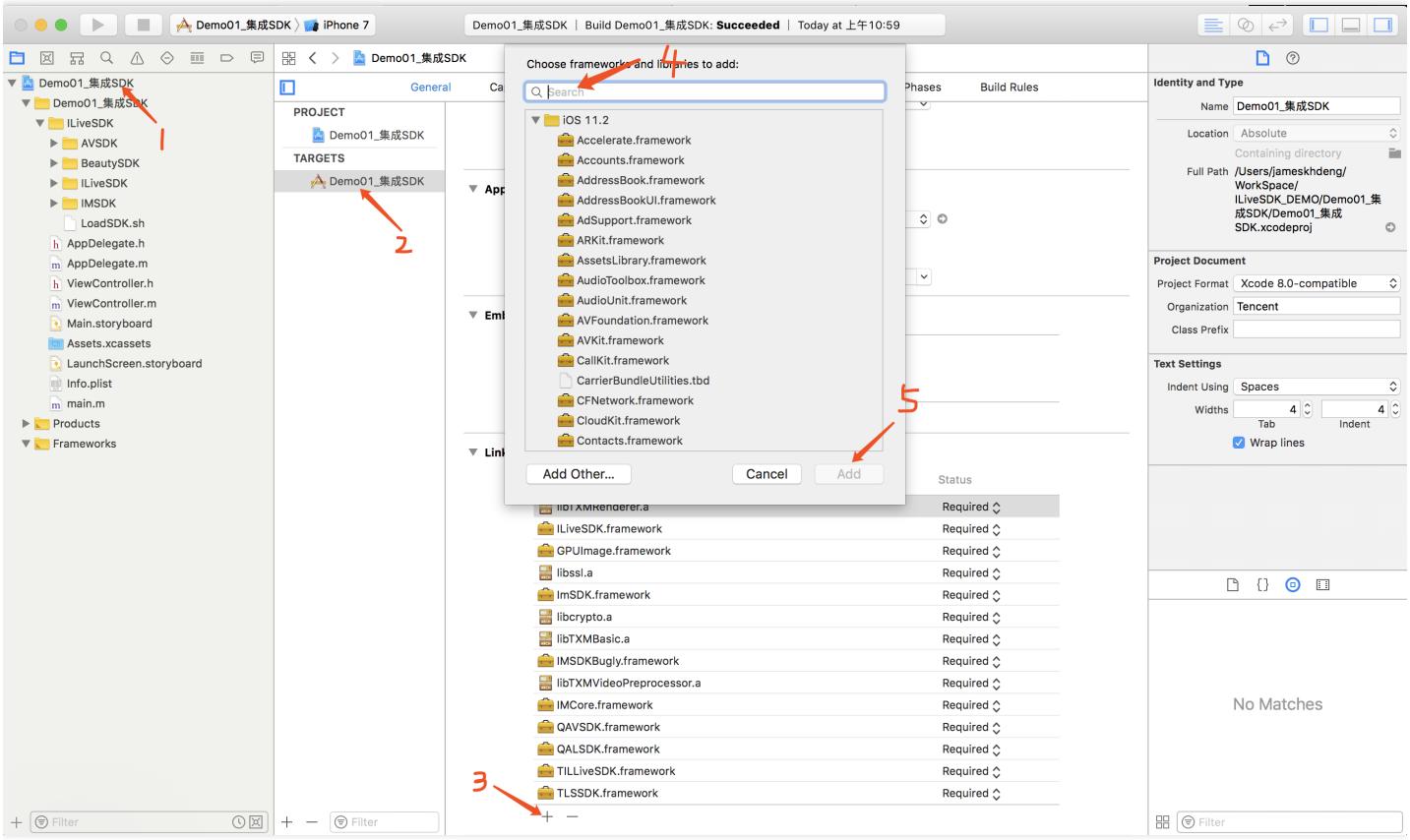


添加系统依赖库

ILiveSDK 中的 SDK 依赖了一些系统库，我们还需要将这些系统库添加到项目中来。

单击【 项目文件 】-【 targets 】-【 Genaeral 】- 拉到最下面的 Linked Frameworks and Libraries 区域 - 单击【 + 】号 - 输入系统库名称 - 单击【 add 】添加。

版权所有：腾讯云计算（北京）有限责任公司



需添加的系统库清单
Accelerate.framework
AssetsLibrary.framework
AVFoundation.framework
CoreGraphics.framework
CoreMedia.framework
CoreTelephony.framework
CoreVideo.framework
ImageIO.framework
JavaScriptCore.framework
OpenAL.framework
OpenGLES.framework
QuartzCore.framework

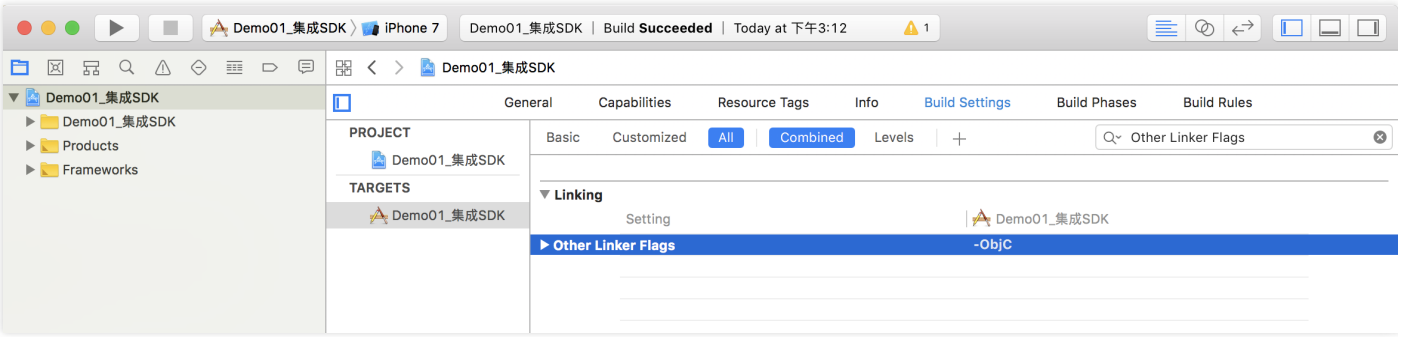
需添加的系统库清单
SystemConfiguration.framework
VideoToolbox.framework
libbz2.tbd
libc++.tbd
libiconv.tbd
libicucore.tbd
libprotobuf.tbd
libresolv.tbd
libsqlite3.tbd
libstdc++.6.tbd
libstdc++.tbd
libz.tbd

添加系统库后，项目中会多出一个 Frameworks 文件夹，里面放的就是我们添加的系统库，由于需要添加的系统库较多，一个方便的方法是，直接在文末下载我们给出的 Demo 代码（单击下载），将其中的系统库直接拖拽到您的工程中（ 从 Frameworks 文件夹直接拖到您的项目的 Linked Frameworks and Libraries 区域 ）。

工程配置

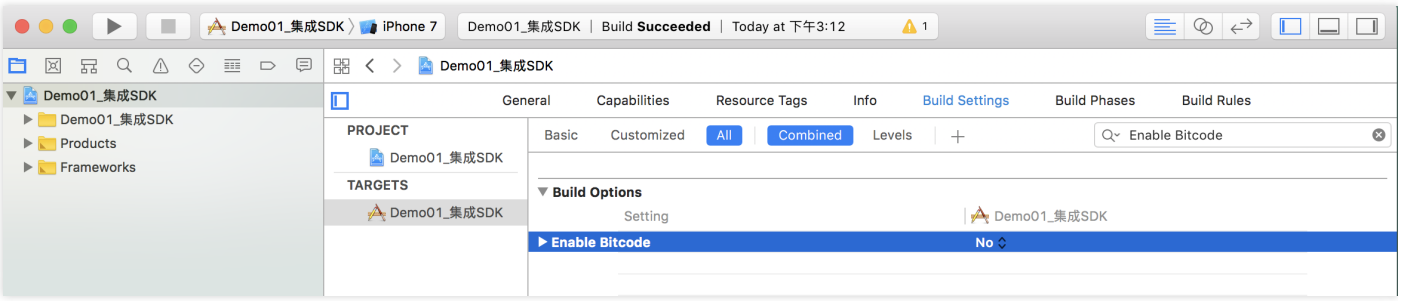
为了能够正常使用 SDK，还需要进行一些工程配置：

1. -ObjC 配置
   【 Build Settings 】 -> 【 Other Linker Flags 】 -> 【 -ObjC （注意大小写）】：



2. Bitcode 配置

【 Build Settings 】 ->【 Enable Bitcode 】 -> 【 NO（设置为 NO）】：



运行检查

如果以上步骤均无错误，这时我们就能正常使用 ILiveSDK 了，在 ViewController.m 的 viewDidLoad 函数中添加获取版本号的代码：

```
//导入头文件
#import <ILiveSDK/ILiveCoreHeader.h>

//获取版本号
NSLog(@"ILiveSDK version:%@",[[ILiveSDK getInstance] getVersion]);
NSLog(@"AVSDK version:%@",[QAVContext getVersion]);
NSLog(@"IMSDK version:%@",[[TIMManager sharedInstance] GetVersion]);

//打印结果
2018-03-27 15:22:37.187181+0800 Demo01_集成SDK[8182:16625633] ILiveSDK version:1.8.3.13017
2018-03-27 15:22:37.187692+0800 Demo01_集成SDK[8182:16625633] AVSDK version:1.9.6.47.OpenSDK_1.9.6 - 34109
2018-03-27 15:22:37.189444+0800 Demo01_集成SDK[8182:16625633] IMSDK version:v2.5.6.11389.11327
```

恭喜，至此说明 ILiveSDK 已经成功集成。

集成 SDK（PC）

最近更新时间：2018-09-28 17:01:18

本文将指导您完成在 PC 端下实时音视频客户端功能的 SDK 集成。

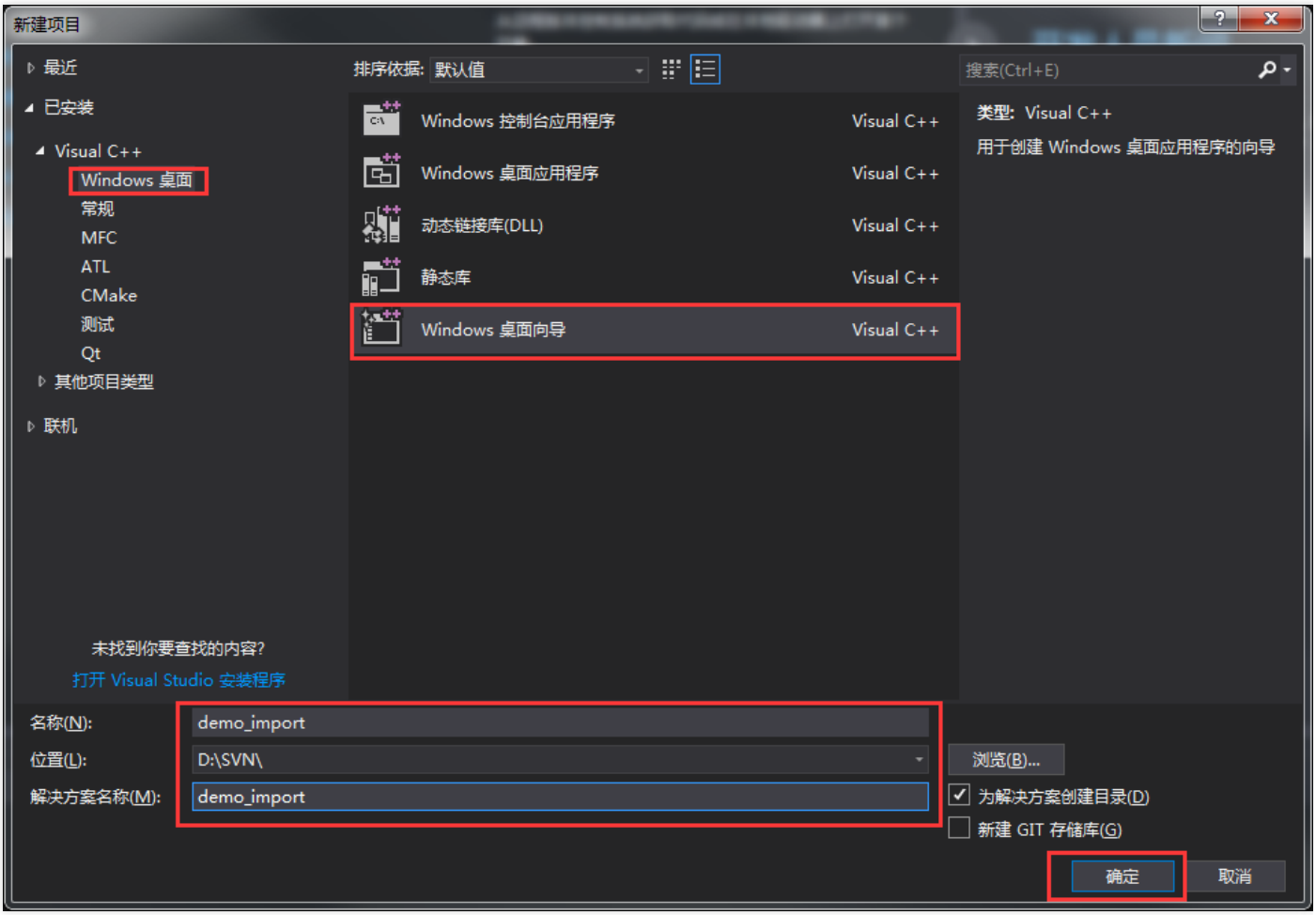
源码下载

在此我们提供以下所讲到的完整 Demo 代码，如有需要请您自行下载。
Demo 代码下载

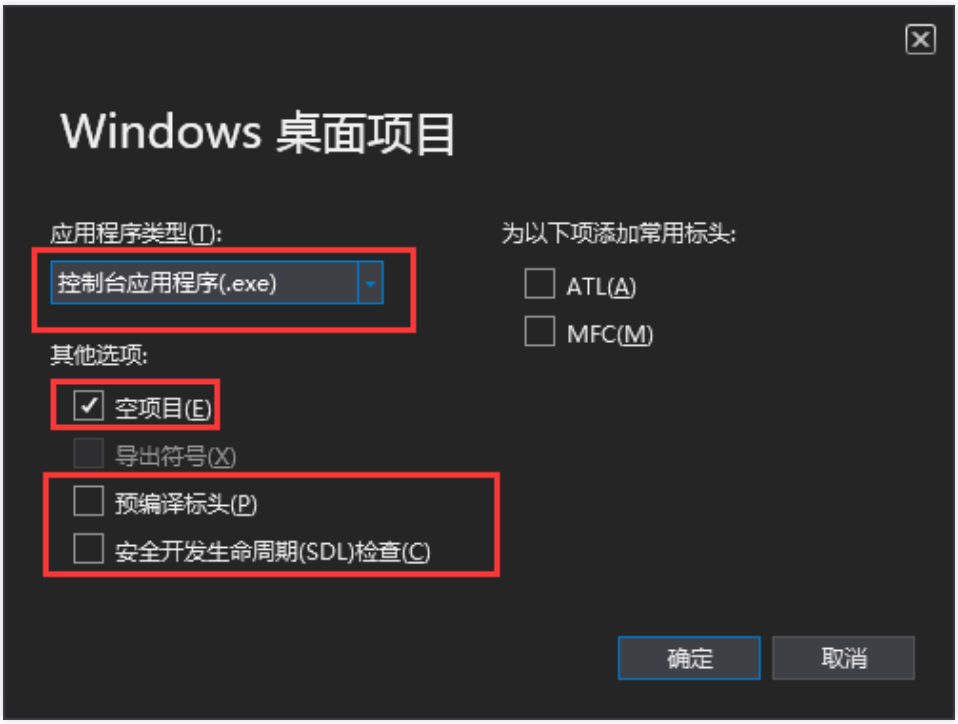
操作步骤

创建 Win32 Console 工程

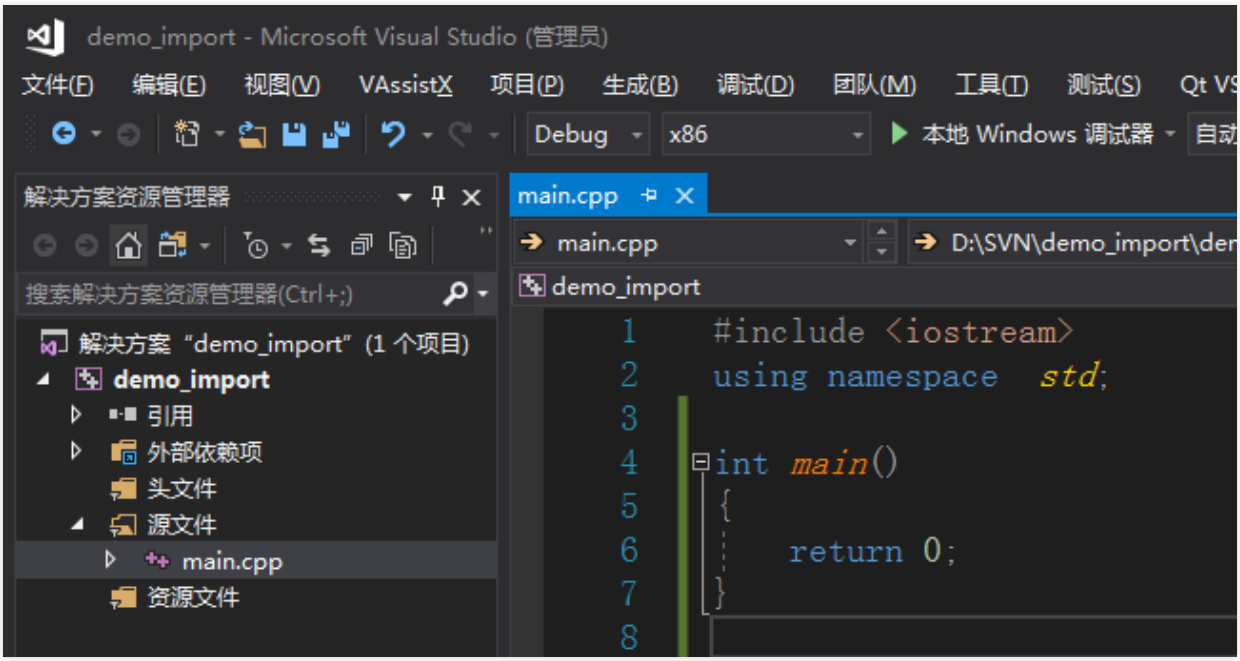
1. 打开 Visual Studio ，单击【 文件 】菜单，选择【 新建 】-【 项目 】-【新建一个项目】：



2. 创建一个空工程：



3. 往工程中添加一个 cpp 文件，编写一个空的主函数：

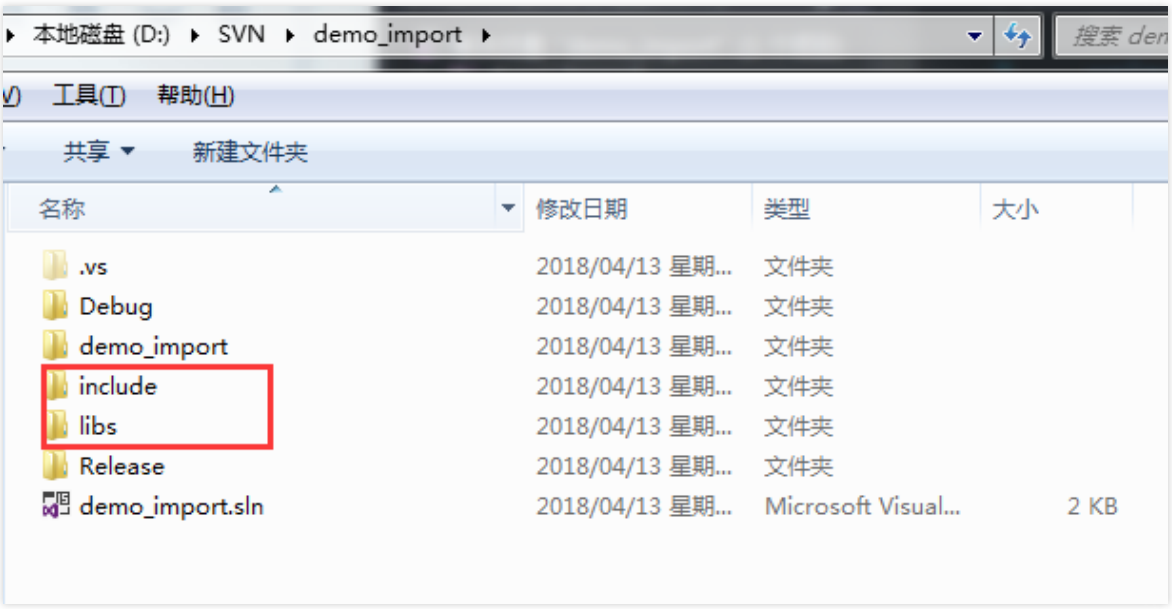


集成 iLiveSDK

1. 下载iLiveSDK
   从github上下载iLiveSDK

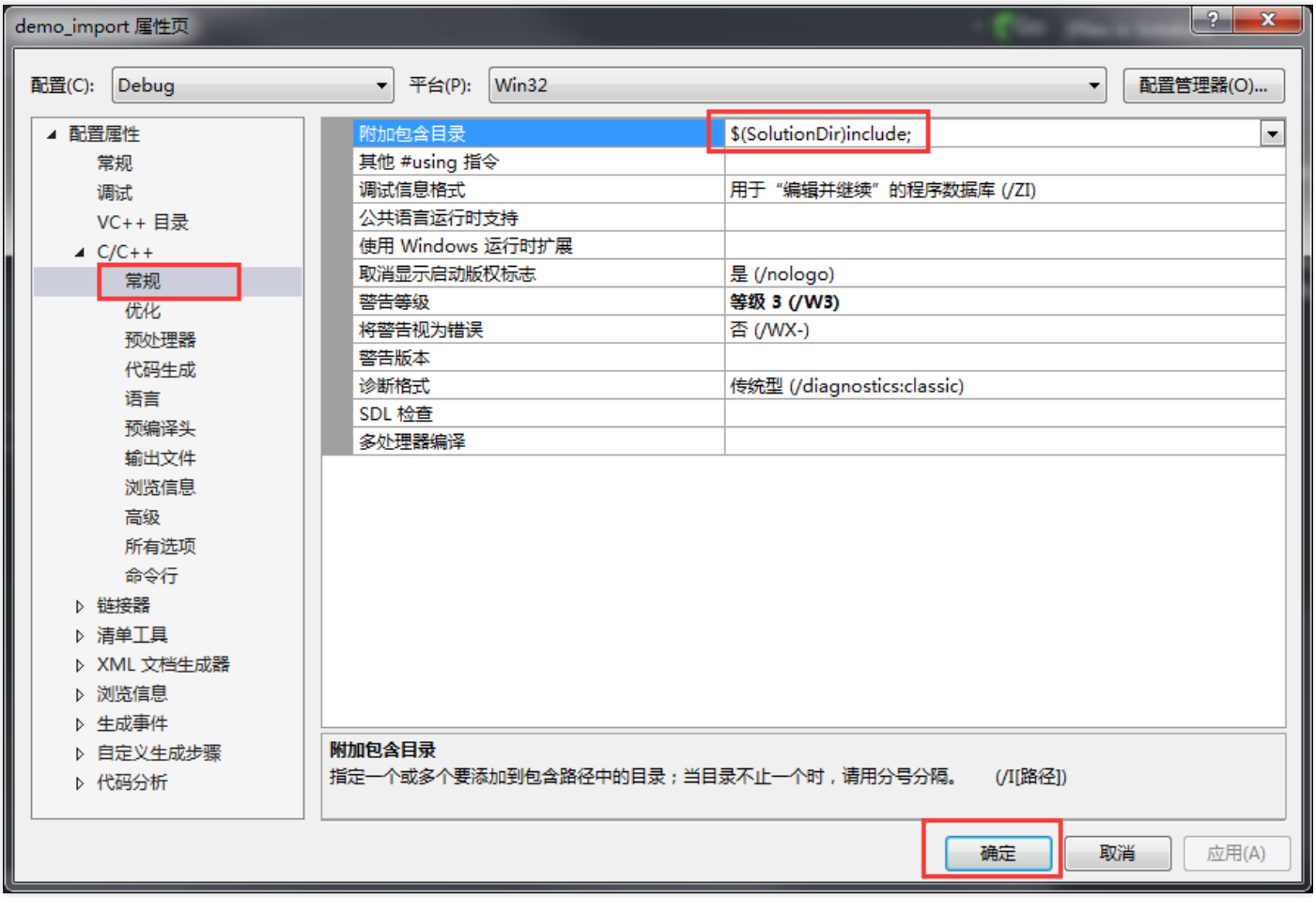
2. 拷贝文件

将iLiveSDK目录下的include和libs拷贝到解决方案文件(.sln文件)所在的目录：



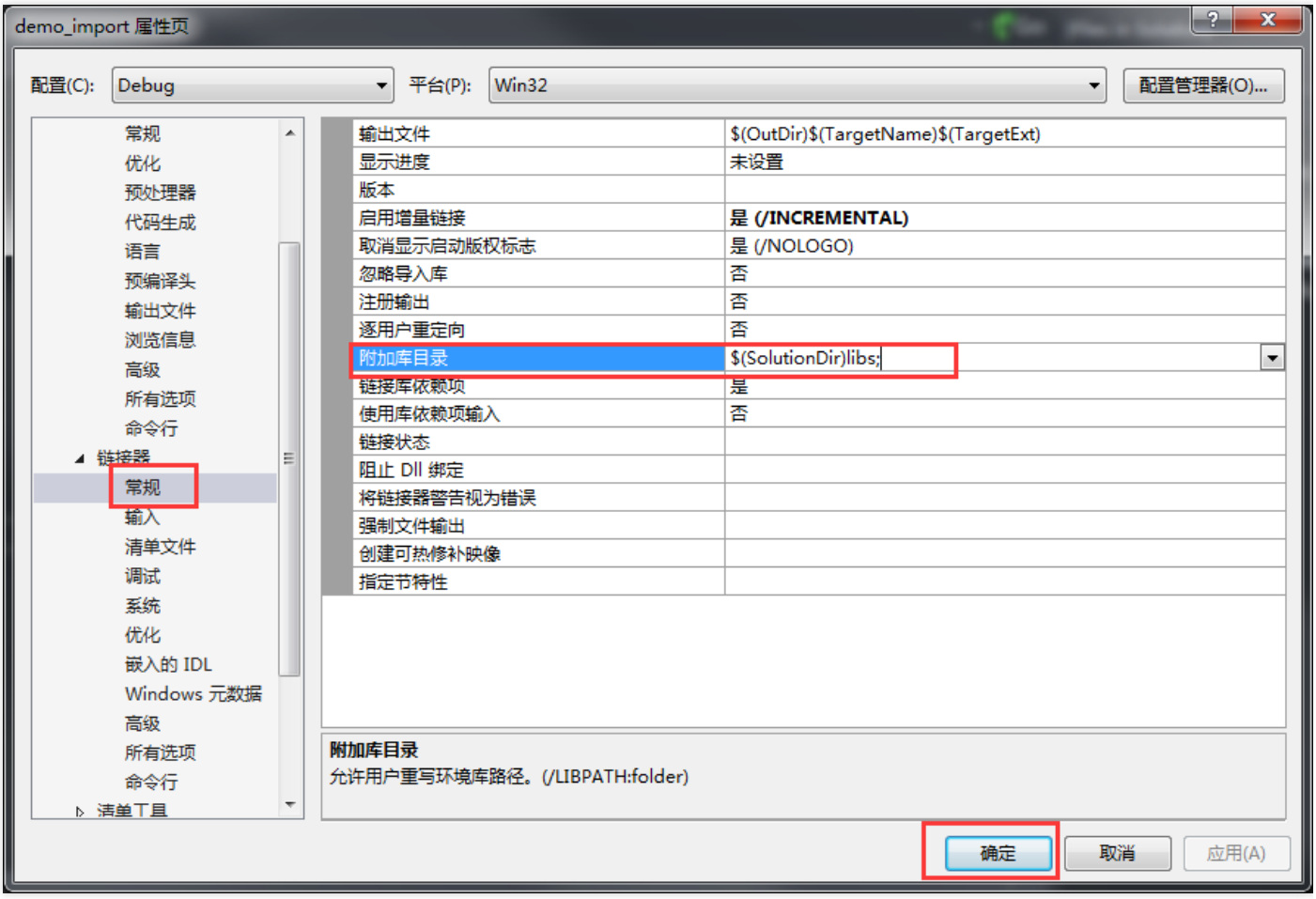
3. 添加 include 目录

在项目的附加包含目录中添加 include 目录， `$(SolutionDir)include` ：



4. 添加库目录

在项目的附加库目录中添加 lib 文件所在目录， `$(SolutionDir)libs` ：



注意:

上面添加 include 目录和库目录时，Debug 和 Release 版本都需要配置。

5. 包含头文件

在项目中包含头文件,并使用 ilive 命名空间，加载动态库的 lib 文件：

```
#include "iLive.h"
using namespace ilive;
#pragma comment(lib, "iLiveSDK.lib")
```

6. 拷贝 dll 文件到程序(.exe)所在目录：

将 libs 目录下的所有 dll 文件复制到解决方案的 Debug 和 Relase 目录下(至少编译一次 Debug 和 Relase 才会生成这两个目录)，此时可删除 libs 目录下的 dll 文件。

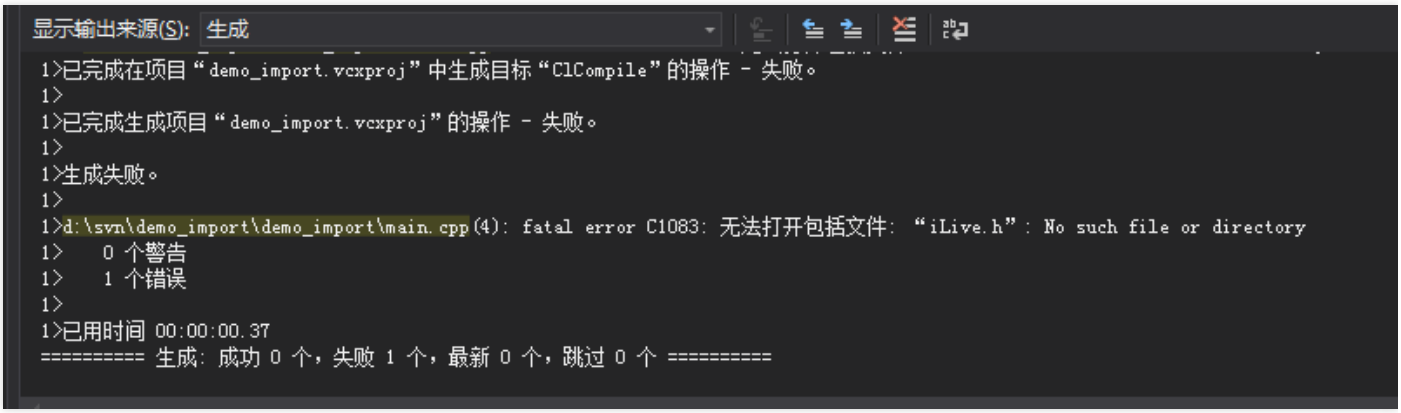
> **注意:**
>]这里不要将 iLiveSDK.lib 也删除了。

7. 验证是否配置成功
   调用 GetILive() -> getVersion() ,输出返回值，查看当前 iLiveSDK 的版本号：

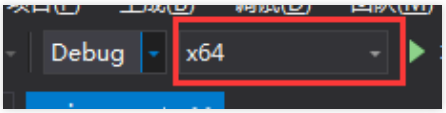
```
cout << GetILive()->getVersion() << endl;
```

源码说明

- Demo 源码编译报错：



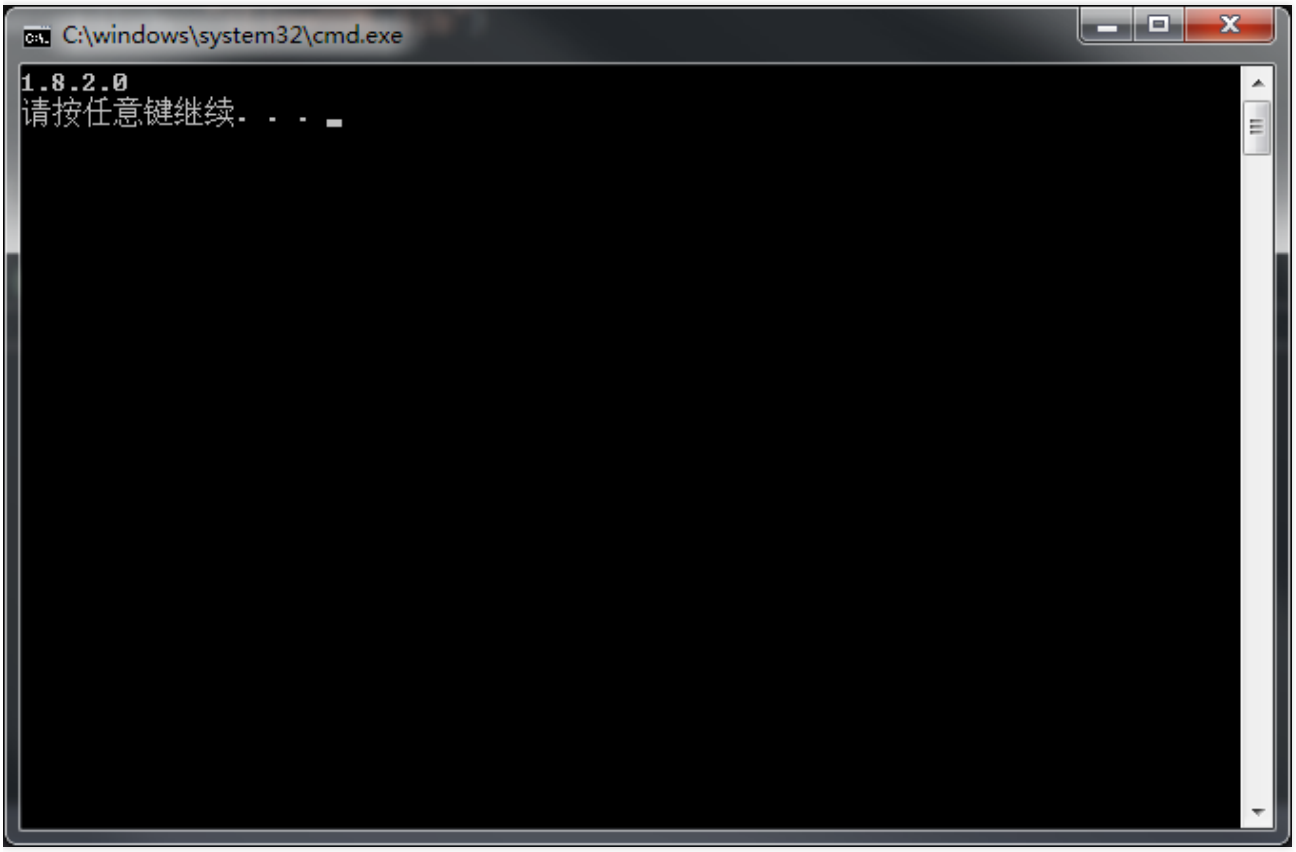
  - 可能是因为项目配置选择了64位。



    iLiveSDK暂时没有64位版本，需要切换成32位编译即可。

运行结果

按【 CTRL+F5 】运行程序，输出版本号：



恭喜，至此说明 iLiveSDK 已经成功集成。

联系邮箱

如果对上述文档有不明白的地方，请反馈到trtcfb@qq.com

集成 SDK（Mac）

最近更新时间：2018-09-28 17:00:40

本文将指导您完成在 Mac 端下实时音视频客户端功能的 SDK 集成。

源码下载

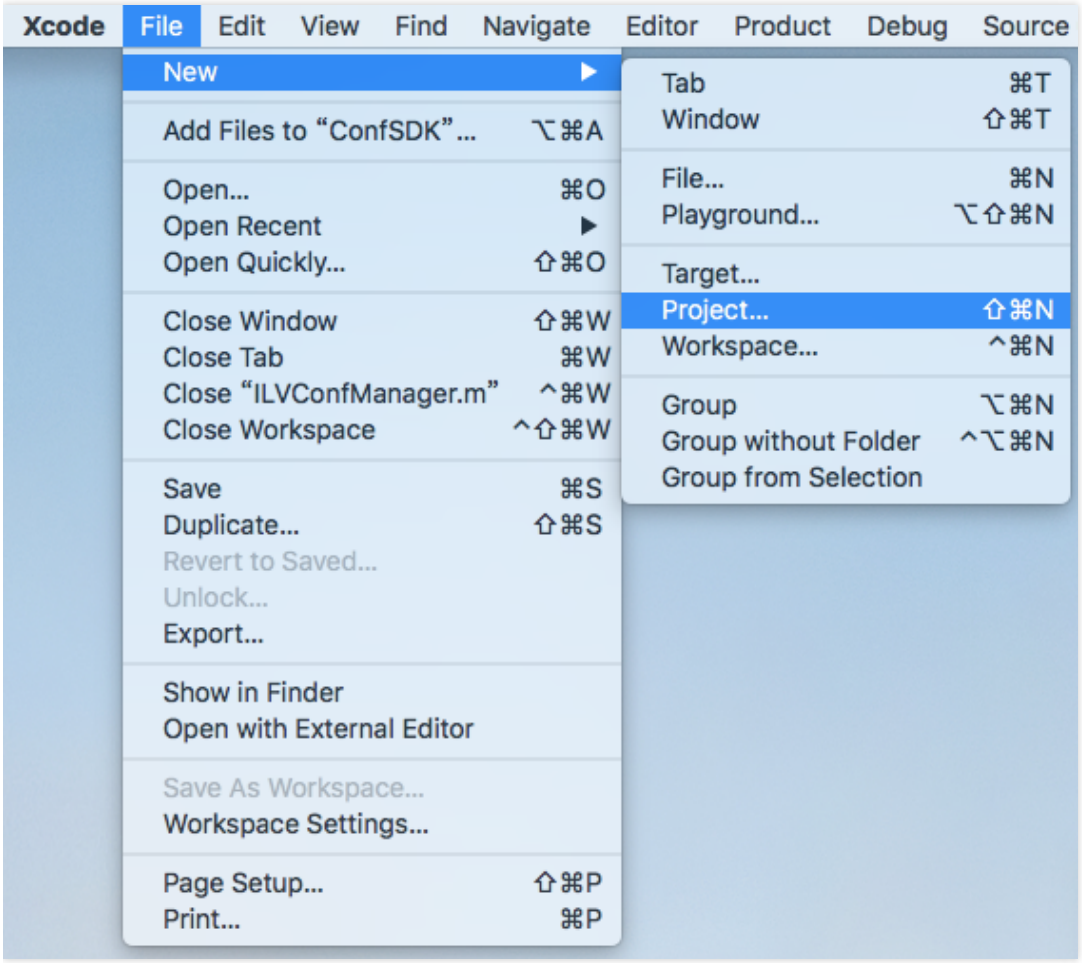
在此我们提供以下所讲到的完整 Demo 代码，如有需要请您自行下载。
Demo 代码下载

操作步骤

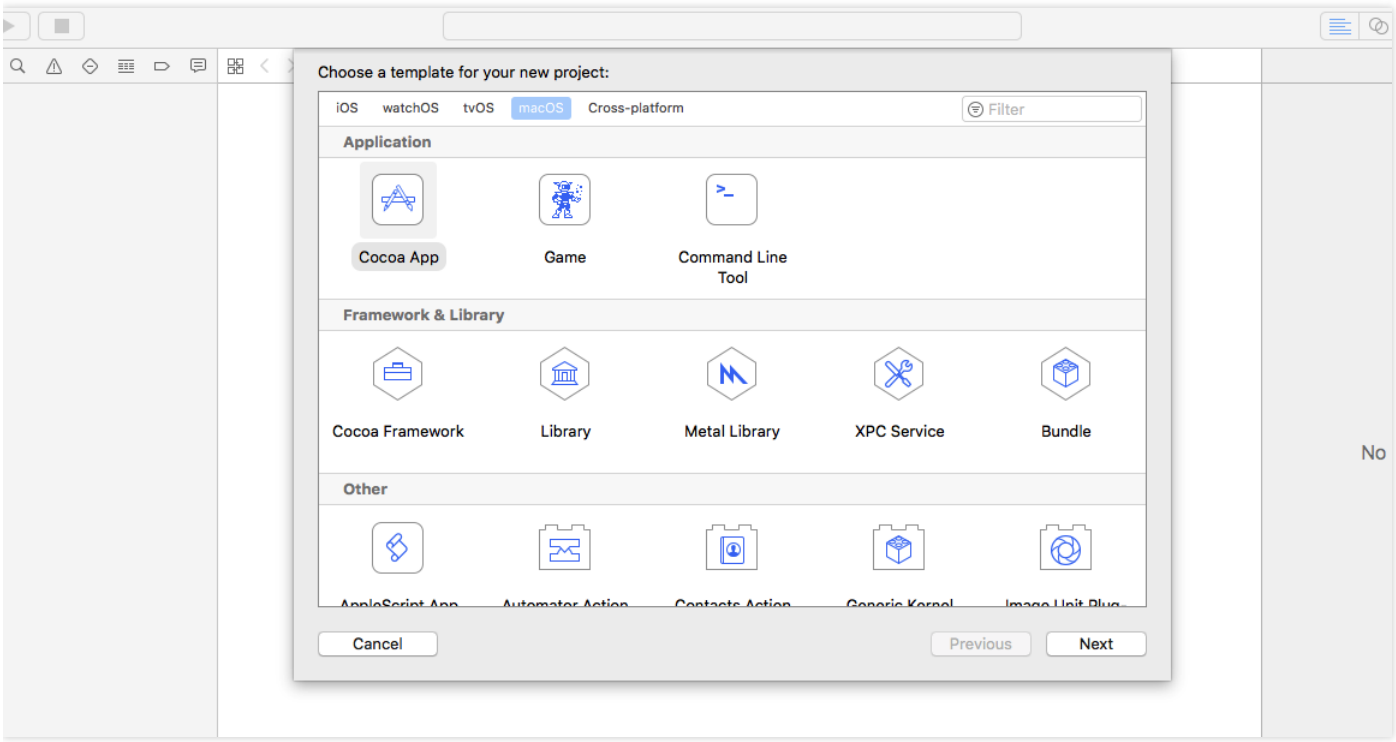
创建 Mac 工程

> 如果您已经有一个工程待集成，请直接跳到下一步 集成 SDK。

首先，使用 Xcode 创建一个新的工程用来集成我们的 SDK。
打开 Xcode，【 File 】-【 New 】-【 Project 】：



选择macOS ->【 Cocoa App 】



设置工程名为TRTCMac集成 SDK，语言选择为 Objective-C，Team、Organization Name 和 Organization Identifier 根据自身情况填写（也可随便填写）,然后选择下一步选择项目存放地址，单击 【 create 】即可 。

集成 SDK

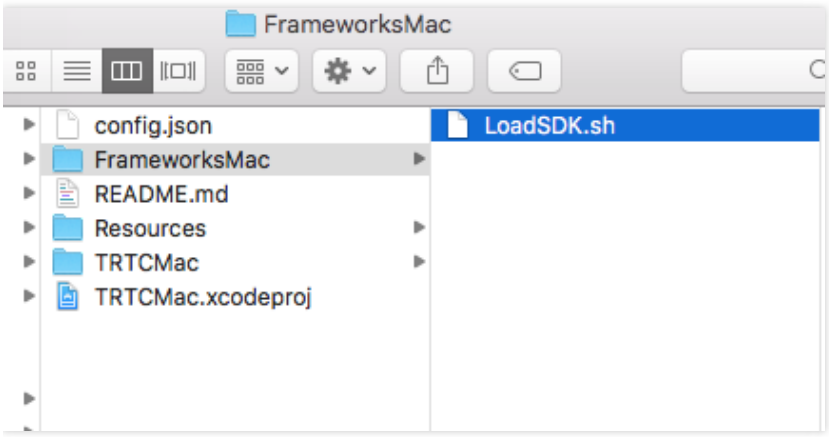
获取SDK

Mac版 ILiveSDK 包含如下一些子 SDK：

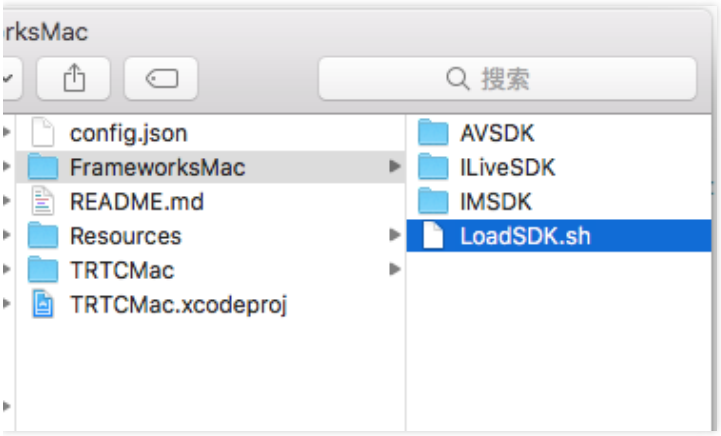
- IMSDK：提供 IM 即时通信功能
- AVSDK：提供底层音视频功能
- ILiveSDK：在 AVSDK 基础上封装而成，提供更简单易用的音视频功能接口

我们先在工程目录中新建一个名为 FrameworksMac 的文件夹，用来存放我们的 SDK，由于 ILiveSDK 包含若干个子 SDK，所以我们提供了一个 SDK 下载脚本方便获取所有的SDK。

单击下载脚本，将其放置于刚才创建的 FrameworksMac 文件夹下：



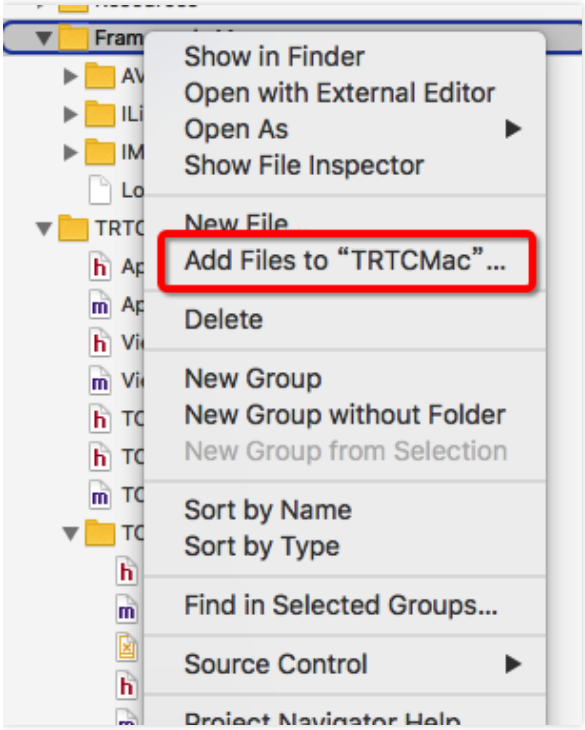
运行下载脚本（ 打开终端，cd 命令进入 FrameworksMac 目录下，运行命令 `sh LoadSDK.sh` ），就会自动下载所有 SDK，下载完成之后会自动解压，并删除下载的压缩包，稍等片刻即可，解压完成之后文件目录如下：



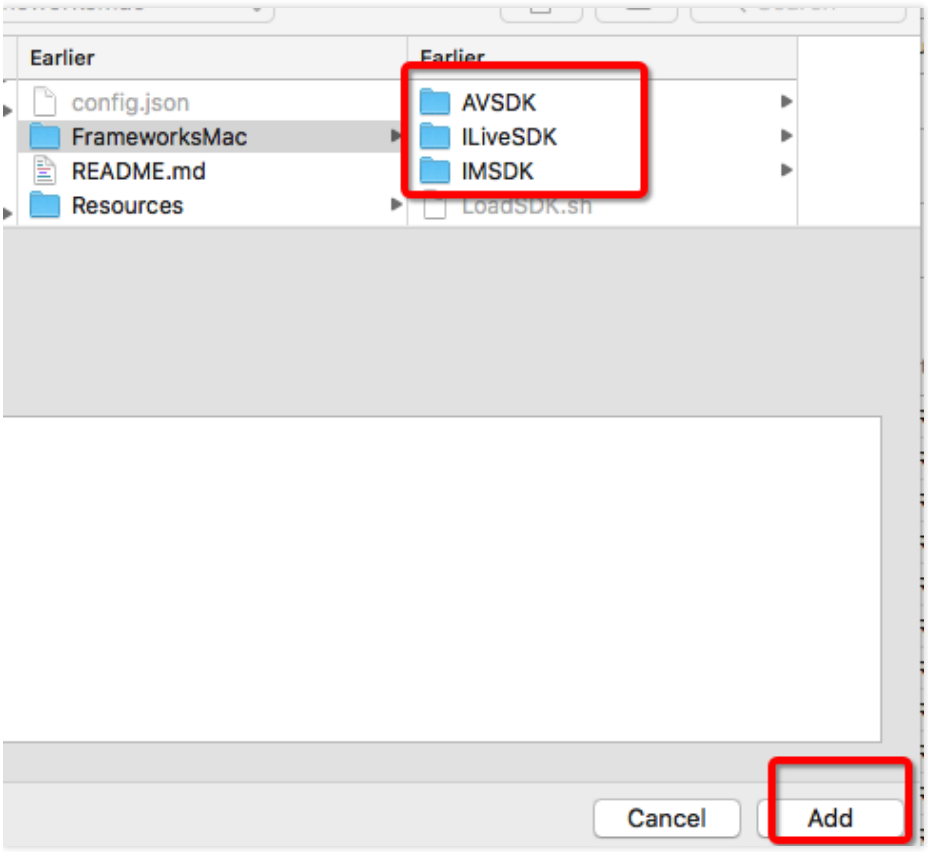
最新版本说明

导入SDK

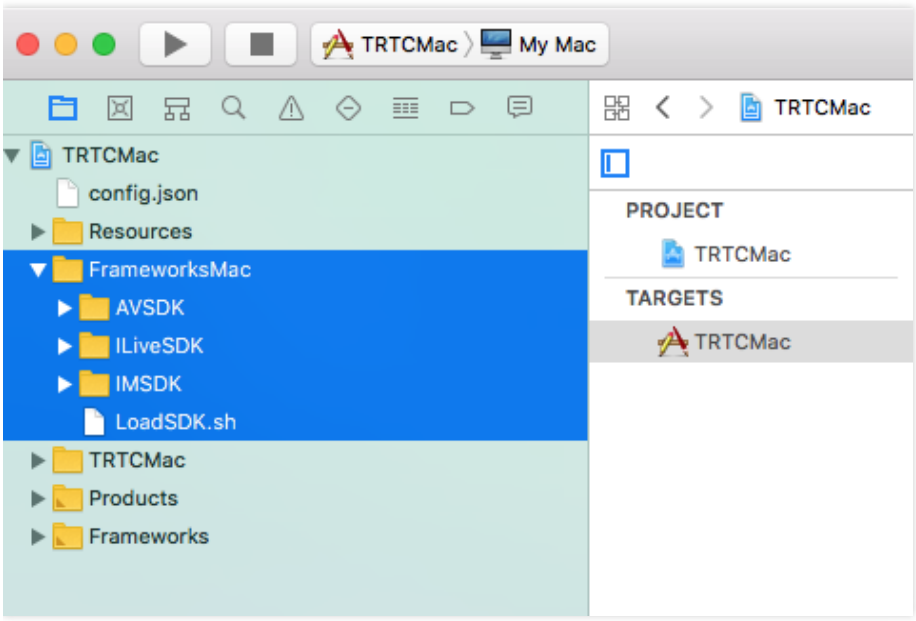
下载完成后，我们需将 SDK 导入工程，在工程根目录上单击右键->【 Add Files to "TRTCMac" 】：



在弹出的目录选择框中选择我们刚刚创建的【 FrameworksMac 文件夹 】->【 Add 】：



添加完成后的工程目录：

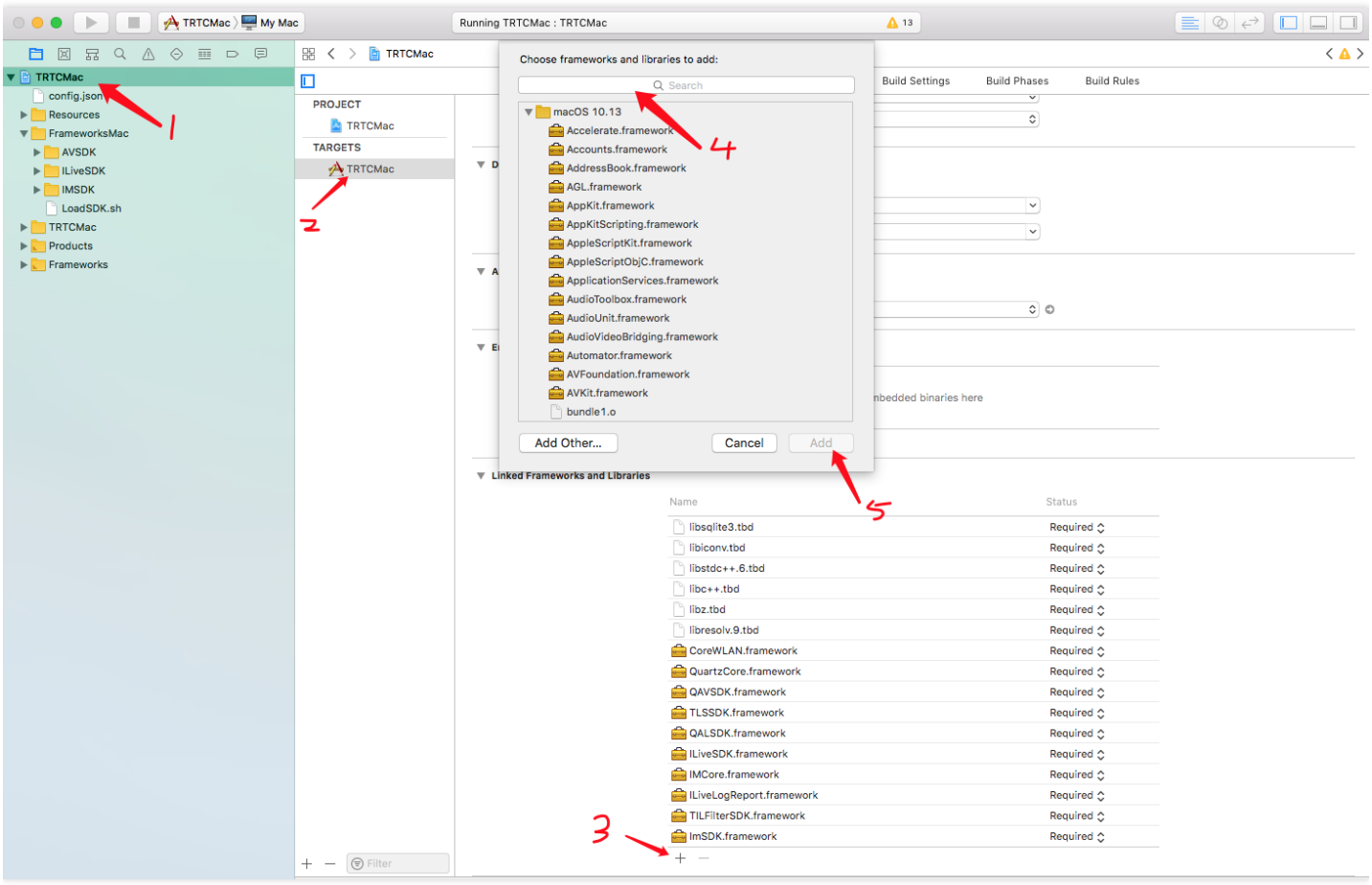


添加系统依赖库

ILiveSDK 中的 SDK 依赖了一些系统库，我们还需要将这些系统库添加到项目中来。

单击【 项目文件 】->【 targets 】->【 Genaeral 】- 拉到最下面的 Linked Frameworks and Libraries 区域 -> 单击【 + 】号 -> 输入系统库名称 -> 单击【 add 】添加。

版权所有：腾讯云计算（北京）有限责任公司



需添加的系统库清单
libsqlite3.tbd
libiconv.tbd
libstdc++.6.tbd
libc++.tbd
libz.tbd
libresolve.9.tbd
CoreWLAN.framework
QuartzCore.framework
QAVSDK.framework
TLSSDK.framework
QALSDK.framework

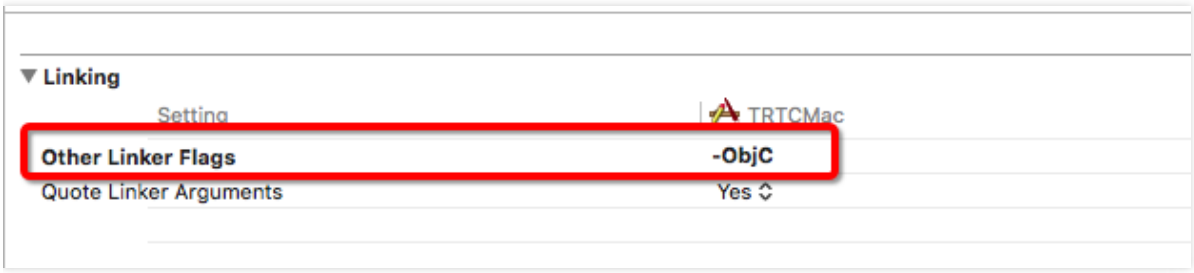
需添加的系统库清单
ILiveSDK.framework
IMCore.framework
IliveLogReport.framework
TILFilterSDK.framework
ImSDK.framework

添加系统库后，项目中会多出一个 Frameworks 文件夹，里面放的就是我们添加的系统库，由于需要添加的系统库较多，一个方便的方法是，直接在文末下载我们给出的 Demo 代码（单击下载），将其中的系统库直接拖拽到您的工程中（ 从 Frameworks 文件夹直接拖到您的项目的 Linked Frameworks and Libraries 区域 ）。

工程配置

为了能够正常使用 SDK，还需要进行一些工程配置：

-ObjC 配置
【 Build Settings 】 -> 【 Other Linker Flags 】 -> 【 -ObjC （注意大小写）】：



运行检查

如果以上步骤均无错误，这时我们就能正常使用 ILiveSDK 了，在 ViewController.m 的 viewDidLoad 函数中添加获取版本号的代码：

```
//导入头文件
#import <ILiveSDK/ILiveCoreHeader.h>

//获取版本号
NSLog(@"ILiveSDK version:%@",[[ILiveSDK getInstance] getVersion]);
NSLog(@"AVSDK version:%@",[QAVContext getVersion]);
NSLog(@"IMSDK version:%@",[[TIMManager sharedInstance] GetVersion]);

//打印结果
2018-09-03 11:49:28.060945+0800 TRTCMac[73399:23447259] ILiveSDK version:1.9.3.13966
```

```
2018-09-03 11:49:28.060956+0800 TRTCMac[73399:23447259] AVSDK version:1.9.9.1012.Local
2018-09-03 11:49:28.060969+0800 TRTCMac[73399:23447259] IMSDK version:v2.5.4.10421.10420
```

恭喜，至此说明 ILiveSDK 已经成功集成。

联系邮箱

如果对上述文档有不明白的地方，请反馈到trtcfb@qq.com

登录
登录（Android）

最近更新时间：2018-09-28 17:02:27

本文将指导您在客户端将通过仓库集成 iLiveSDK，并向腾讯云完成登录过程。

源码下载

在此我们提供以下所讲到的完整 Demo 代码，如有需要请您自行下载。
Demo 代码下载

前提条件

要求用户在 实时音视频官网 完成服务开通及应用创建。

相关概念

- 实时音视频应用
- 应用标识(sdkAppId)
- 帐号类型(accountType)
- 用户标识(userId)
- 用户签名(userSig)

获取 userSig

客户端的每一个用户都需要一个独立的 userSig，userSig 是有效期的(在生成时设置，一般为三个月)，如果 userSig 过期，用户登录时会收到错误码 8051，这时用户需要重新生成 userSig，拿到新的 userSig 再登录。

```
/** 票据过期(需更新票据userSig) */
public static final int ERR_EXPIRE = 8051;
```

注意： 关于 userSig 的获取请参考 用户鉴权 。 在调试期间您可以直接使用控制台的开发辅助工具生成 userSig。

添加依赖(集成 SDK)

修改 build.gradle 文件，在 dependencies 中添加 iLiveSDK 的依赖：

```
compile 'com.tencent.ilivesdk:ilivesdk:latest.release' //其中latest.release指代最新iLiveSDK版本号
```

初始化 iLiveSDK

首先添加 DemoApp.java 并继承 Application：

```
public class DemoApp extends Application {
@Override
public void onCreate() {
super.onCreate();

// 判断仅在主线程进行初始化
if (MsfSdkUtils.isMainProcess(this)) {
// 初始化iLiveSDK
ILiveSDK.getInstance().initSdk(this, Constants.SDKAPPID, Constants.ACCOUNTTYPE);
// 初始化iLiveSDK房间管理模块
ILiveRoomManager.getInstance().init(new ILiveRoomConfig());
}
}
}
```

并在 AndroidManifest.xml 的 application 中申明：

```
<application
.....
android:name=".DemoApp">
.....
</application>
```

创建登录模块

首先创建一个登录模块与 Acitivity 通讯的接口：

```
public interface ILoginView {
// 登录成功
void onLoginSDKSuccess();
// 登录失败
```

```
void onLoginSDKFailed(String module, int errCode, String errMsg);
}
```

同时创建一个 LoginHelper.java 用于登录操作：

```
public class LoginHelper {
private ILoginView loginView;

public LoginHelper(ILoginView view){
loginView = view;
}

public void loginSDK(String userId, String userSig){
ILiveLoginManager.getInstance().iLiveLogin(userId, userSig, new ILiveCallBack() {
@Override
public void onSuccess(Object data) {
loginView.onLoginSuccess();
}

@Override
public void onError(String module, int errCode, String errMsg) {
loginView.onLoginFailed(module, errCode, errMsg);
}
});
}
}
```

监听帐户状态

用户在登录后，也是有可能强制下线的(帐号重复登录被踢，userSig 过期)，所以应用需要监听这个状态。
为了全局监听，可以创建一个观察者：

```
/**
* 状态观察者
*/
public class StatusObservable implements ILiveLoginManager.TILVBStatusListener {

// 消息监听链表
private LinkedList<ILiveLoginManager.TILVBStatusListener> listObservers = new LinkedList<>();
// 句柄
private static StatusObservable instance;


public static StatusObservable getInstance(){
if (null == instance){
```

```
synchronized (StatusObservable.class){
if (null == instance){
instance = new StatusObservable();
}
}
}
return instance;
}


// 添加观察者
public void addObserver(ILiveLoginManager.TILVBStatusListener listener){
if (!listObservers.contains(listener)){
listObservers.add(listener);
}
}

// 移除观察者
public void deleteObserver(ILiveLoginManager.TILVBStatusListener listener){
listObservers.remove(listener);
}

// 获取观察者数量
public int getObserverCount(){
return listObservers.size();
}

@Override
public void onForceOffline(int error, String message) {
// 拷贝链表
LinkedList<ILiveLoginManager.TILVBStatusListener> tmpList = new LinkedList<>(listObservers);
for (ILiveLoginManager.TILVBStatusListener listener : tmpList){
listener.onForceOffline(error, message);
}
}
}
```

并在登录成功后调用接口设置监听：

```
ILiveLoginManager.getInstance().setUserStatusListener(StatusObservable.getInstance());
```

这样用户在收到 onForceOffline 事件就知道自己被踢下线了。

UI 开发

客户端需要登录，所以可能需要单独一个页面来输入用户名，密钥，这些都是 Android 开发基础知识，就不一一讲解了。

登录应用的 onCreated 事件中，可以创建上面定义的模块：

```
loginHelper = new LoginHelper(this);
```

然后在单击登录事件后，获取用户输入的 userId 和 userSig，调用 loginHelper 的登录接口进行登录：

```
loginHelper.loginSDK(userId, userSig);
```

常见问题

- 下载 aar 失败

```
Error:Could not resolve all files for configuration ':app:debugCompileClasspath'.
> Could not resolve com.tencent.ilivesdk:ilivesdk:1.8.3.
Required by:
project :app
> Could not resolve com.tencent.ilivesdk:ilivesdk:1.8.3.
> Could not get resource 'https://jcenter.bintray.com/com/tencent/ilivesdk/ilivesdk/1.8.3/ilivesdk-1.8.3.pom'.
> Could not GET 'https://jcenter.bintray.com/com/tencent/ilivesdk/ilivesdk/1.8.3/ilivesdk-1.8.3.pom'.
> Connect to jcenter.bintray.com:443 [jcenter.bintray.com/75.126.118.188] failed: Connection timed out: connect
```

  先检测网络是否正常，并通过上面链接，确认可以访问 jcenter 网站，同时如果网络需要代理检测是否有在 gradle.properties 中配置。

- 登录未返回错误模块 IMSDK, 错误码 70009, 错误描述：tls_check*signature failed decrypt sig failed failed iRet:-2 sdkappid:14000xxxxx,acctype:xxxx,identifier:guest sig:E9vB6Ocs42J8A5lZW6s*

> 这种问题一般为登录的 userSig 与 id 不匹配引起，需要生成 userSig 的密钥与初始化时使用的 sdkAppId 是否对应

联系邮箱

如果对上述文档有不明白的地方，请反馈到trtcfb@qq.com

登录（iOS）

最近更新时间：2018-09-28 17:05:39

本文将指导您初始化 SDK 和如何调用登录接口。

源码下载

在此我们提供以下所讲到的完整 Demo 代码，如有需要请您自行下载。
Demo 代码下载

前提条件

要求用户在 实时音视频官网 完成服务开通及应用创建。

相关概念

- 实时音视频应用
- 应用标识(sdkAppId)
- 帐号类型(accountType)
- 用户标识(userId)
- 用户签名(userSig)

获取 userSig

客户端的每一个用户都需要一个独立的 userSig，userSig 是有效期的(在生成时设置，一般为三个月)，如果 userSig 过期，用户登录时会收到错误码 8051，这时用户需要重新生成 userSig，拿到新的 userSig再登录。

```
ERR_EXPIRE = 8051, //票据过期(需更新票据userSig)
```

注意： 关于 userSig 的获取请参考 用户鉴权 。 在调试期间您可以直接使用控制台的开发辅助工具生成 userSig。

初始化 iLiveSDK

建议在程序启动时就将 SDK 初始化，初始化需要用到在腾讯云后台创建应用时分配的 SDKAppId 和 AccountType，示例代码如下：

```
> AppDelegate.m

// 导入头文件
#import <ILiveSDK/ILiveSDK.h>

// 定义SDKAppId和AccountType
static const int kSDKAppID = 后台创建应用对应的SDKAppId;
static const int kAccountType = 后台创建应用对应的AccountType;

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

// 初始化SDK
[[ILiveSDK getInstance] initSdk:kSDKAppID accountType:kAccountType];

return YES;
}
```

调用登录接口

登录接口位于 ILiveLoginManager 类中，使用时需将其导入，调用方法：

```
- (void)iLiveLogin:(NSString *)uid sig:(NSString *)sig succ:(TCIVoidBlock)succ failed:(TCIErrorBlock)failed;
```

填入后台开发辅助工具生成的 userId 和 userSig ：

```
> ViewController.m

// 导入头文件
#import <ILiveSDK/ILiveLoginManager.h>

- (void)viewDidLoad {
[super viewDidLoad];

// 设置userId和userSig
self.userIdTF.text = @"用户自己的userId";
self.userSigTF.text = @"用户自己生成的userSig";
}

// 登录按钮单击
```

```
- (IBAction)onLogin:(id)sender {
//登录sdk
[[ILiveLoginManager getInstance] iLiveLogin:self.userIdTF.text sig:self.userSigTF.text succ:^{
NSLog(@"登录成功！");
} failed:^(NSString *module, int errId, NSString *errMsg) {
NSLog(@"errId:%d, errMsg:%@",errId, errMsg);
}];
}
```

监听用户状态

用户在登录后，有可能被强制下线的(帐号重复登录被踢，userSig 过期)，所以当应用需要监听这两个状态时可进行如下步骤实现：

设置监听对象调用方法：

```
[[ILiveSDK getInstance] setUserStatusListener:self];
```

监听对象遵守TIMUserStatusListener协议，并实现以下方法：

```
/**
* 踢下线通知
*/
- (void)onForceOffline;

/**
* 用户登录的userSig过期（用户需要重新获取userSig后登录）
*/
- (void)onUserSigExpired;
```

注意：
示例程序中用 storyboard 搭建了一个简单界面，实际应用中客户需根据自己的业务逻辑，将登录代码放置与合适的位置。
且以上仅为示例代码，实际过程中（客户采用独立帐号模式），登录流程应为：

客户端显示登录界面，用户输入帐号密码，客户的业务后台验证通过后使用实时音视频后台 SDK 生成 userSig 返回给客户端，客户端再拿用户输入的用户名和业务后台返回的 userSig 调用 ILiveSDK的登录接口。

运行程序，控制台输出登录成功！，即说明 ILiveSDK 登录成功。

版权所有：腾讯云计算（北京）有限责任公司

登录（PC）

最近更新时间：2018-09-28 17:03:50

本文将指导您操作客户端将向腾讯云完成登录过程。

源码下载

在此我们提供以下所讲到的完整 Demo 代码，如有需要请您自行下载。
Demo 代码下载

前提条件

本课程要求用户在 实时音视频官网 完成服务开通及应用创建。

相关概念

- 实时音视频应用
- 应用标识(sdkAppId)
- 帐号类型(accountType)
- 用户标识(userId)
- 用户签名(userSig)

userSig的获取

关于 userSig 的获取请参考 用户鉴权 。
在这里您也可以直接开发辅助工具生成 userSig。

初始化 iLiveSDK

在登录之前，需要初始化 SDK，建议在程序运行时就初始化，示例代码如下：

```
#define SDKAppId 后台创建应用对应的SDKAppId
#define AccountType 后台创建应用对应的AccountType

int nRet = GetILive()->init(SDKAppId, AccountType, false);
if (nRet != NO_ERR)
```

版权所有：腾讯云计算（北京）有限责任公司

```
{
//初始化失败,nRet为错误码
}
else
{
//初始化成功
}
```

调用登录接口

初始化后，即可进行登录，示例代码如下：

```
GetILive()->login(userId, UserSig, [](void* data) {
//登录成功
}, [](const int code, const char *desc, void* data) {
//登录失败, code为错误码，desc为错误原因描述
}, NULL);
```

源码说明

- 演示代码使用了 C++11
  上面登录接口在传入 SDK 成功和失败回调时，使用了 C++11 的 lambda 表达式，如果用户使用的 vs2010 及更低版本（不支持或不完全支持 C++11），成功和失败的回调需要传入 C 语言的函数指针。调用示例如下：

```
void OnLoginSuc(void* data)
{
//登录成功
}

void OnLoginErr(const int code, const char *desc, void* data)
{
//登录失败, code为错误码，desc为错误原因描述
}

GetILive()->login(userId, UserSig, OnLoginSuc, OnLoginErr, NULL);
```

  后续都使用 lambda 表达式，不再重复说明。

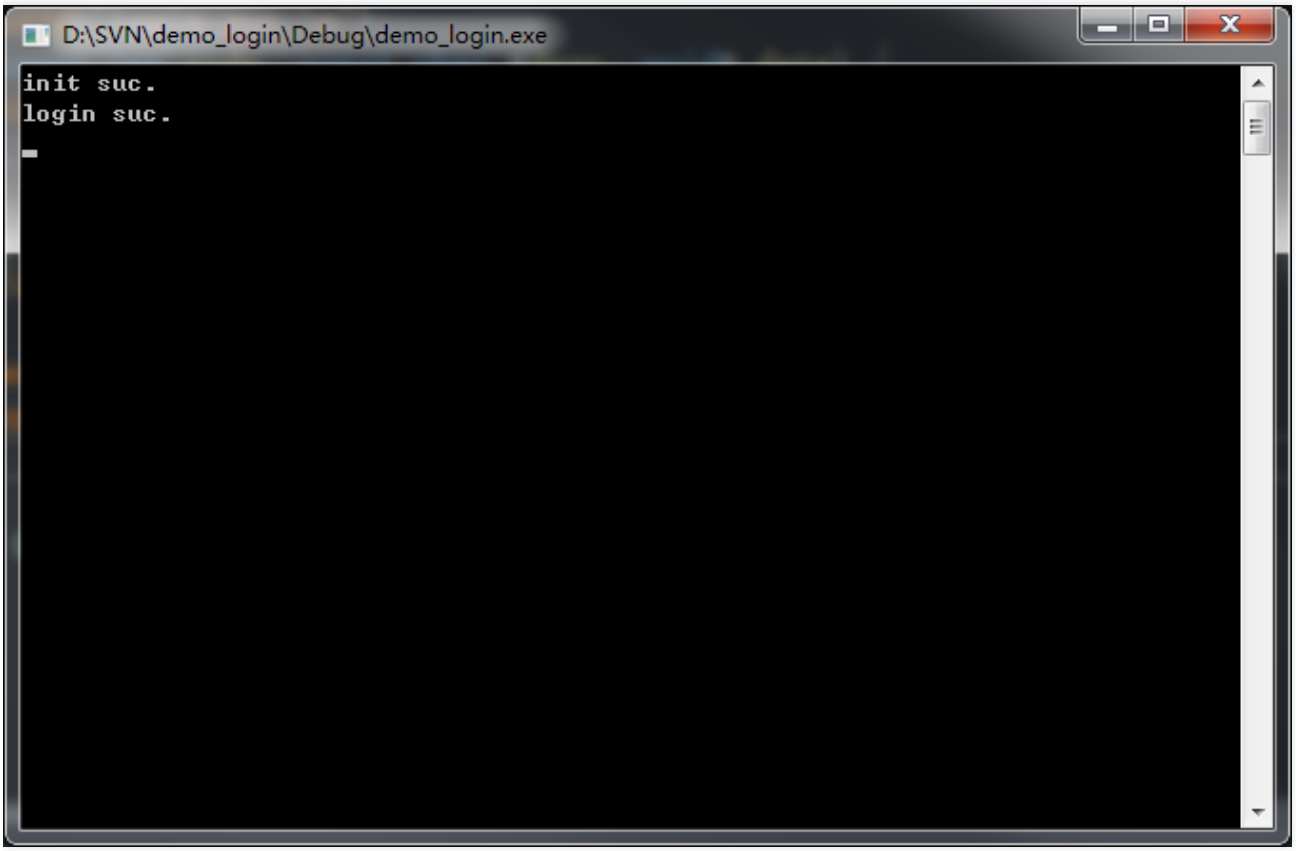
- 异步回调都带一个 void* data 参数
  由于回调是用 C 的函数指针实现的，回调中无法访问回调函数外部的变量，所以，设计了 void* 参数；异步回调传入的 void* 参数，会在回调中原封不动地传出；一般是将类对象的 this 指针传入，在回调中强制转换回类对象指针，然后，通过此指针访问类成员。

- iLiveSDK 依赖 Windows 消息循环
  iLiveSDK 内部使用 Windows 的消息循环机制将回调抛回主线程（GUI 线程），方便用户在回调中执行 UI 操作。所以，如果是 Win32 控制台程序，在程序中必须要有消息循环，代码如下：

```
#include <Windows.h>
MSG msg;
while (GetMessage(&msg, NULL, 0, 0))
{
TranslateMessage(&msg);
DispatchMessage(&msg);
}
```

- 本例 Demo 将 userId 和 userSig 写死了，userSig 是有有效期的，一般为三个月；如果过期，将会登录失败；用户在运行 demo 时，需要按照本文所述的方式生成自己的 userSig，然后换成自己的 SDKAppId、AccountType、userId 和 userSig 进行测试。

运行结果



联系邮箱

如果对上述文档有不明白的地方，请反馈到trtcfb@qq.com

登录（Mac）

最近更新时间：2018-09-28 17:03:16

本文将指导您初始化 SDK 和如何调用登录接口。

源码下载

在此我们提供以下所讲到的完整 Demo 代码，如有需要请您自行下载。
Demo 代码下载

前提条件

要求用户在 实时音视频官网 完成服务开通及应用创建。

相关概念

- 实时音视频应用
- 应用标识(sdkAppId)
- 帐号类型(accountType)
- 用户标识(userId)
- 用户签名(userSig)

获取 userSig

客户端的每一个用户都需要一个独立的 userSig，userSig 是有效期的(在生成时设置，一般为三个月)，如果 userSig 过期，用户登录时会收到错误码 8051，这时用户需要重新生成 userSig，拿到新的 userSig再登录。

```
ERR_EXPIRE = 8051, //票据过期(需更新票据userSig)
```

注意： 关于 userSig 的获取请参考 用户鉴权 。 在调试期间您可以直接使用控制台的开发辅助工具生成 userSig。

初始化 iLiveSDK

建议在程序启动时就将 SDK 初始化，初始化需要用到在腾讯云后台创建应用时分配的 SDKAppId，本示例代码中在本地配置了SDKAppId，读取配置获取信息即可，示例代码如下：

> AppDelegate.m

```
// 导入头文件
#import <ILiveSDK/ILiveSDK.h>


- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {

[[ILiveSDK getInstance] initSdk:[TCLiveConfigInfo getInstance].sdkAppID];

}
```

调用登录接口

登录接口位于 ILiveLoginManager 类中，使用时需将其导入，调用方法：

```
- (void)iLiveLogin:(NSString *)uid sig:(NSString *)sig succ:(TCIVoidBlock)succ failed:(TCIErrorBlock)failed;
```

填入后台开发辅助工具生成的 userId 和 userSig ：

> ViewController.m

```
// 导入头文件
#import <ILiveSDK/ILiveLoginManager.h>

- (IBAction)onLogin:(id)sender {
if ([self.inputTextField stringValue].length >0) {
NSDictionary *dic = [[TCLiveConfigInfo getInstance] parseLoginInfoConfig];
for (NSDictionary *user in dic[@"users"]) {
NSString *userid = user[@"userId"];
NSInteger selectIndex = [self.userName indexOfSelectedItem];
NSString *selectName = [self.userName itemTitleAtIndex:selectIndex];
if ([userid isEqualToString:selectName]) {
[TCLiveConfigInfo getInstance].userToken = user[@"userToken"];
[TCLiveConfigInfo getInstance].userID = selectName;
break;
}
}
//step2 登录
[[ILiveLoginManager getInstance] iLiveLogin:[TCLiveConfigInfo getInstance].userID sig:[TCLiveConfigInfo g
```

```
etInstance].userToken succ:^{
NSLog(@"-----> login succ");
} failed:^(NSString *module, int errId, NSString *errMsg) {

NSLog(@"-----> login fail,%@ %d %@",module, errId, errMsg);
}];

NSString *defaultRole = [[TCLiveConfigInfo getInstance].roles firstObject][@"name"];
_vc = [[TCLiveRoomWC alloc] initWithRoomID:[self.inputTextField stringValue] role:defaultRole];
[_vc.window orderFront:nil];
}
else{

}

}
```

监听用户状态

用户在登录后，有可能被强制下线的(帐号重复登录被踢，userSig 过期），所以当应用需要监听这两个状态时可进行如下步骤实现：

设置监听对象调用方法：

```
[[ILiveSDK getInstance] setUserStatusListener:self];
```

监听对象遵守TIMUserStatusListener协议，并实现以下方法：

```
/**
* 踢下线通知
*/
- (void)onForceOffline;

/**
* 用户登录的userSig过期（用户需要重新获取userSig后登录）
*/
- (void)onUserSigExpired;
```

注意：
示例程序中用 storyboard 搭建了一个简单界面，实际应用中客户需根据自己的业务逻辑，将登录代码放置与合适的位置。
且以上仅为示例代码，实际过程中（客户采用独立帐号模式），登录流程应为：

客户端显示登录界面，用户输入帐号密码，客户的业务后台验证通过后使用实时音视频后台 SDK 生成 userSig 返回给客户端，客户端再拿用户输入的用户名和业务后台返回的 userSig 调用 ILiveSDK的登录接口。

运行程序，控制台输出“登录成功！”，即说明 ILiveSDK 登录成功。

联系邮箱

如果对上述文档有不明白的地方，请反馈到trtcfb@qq.com

创建并加入房间
创建并加入房间（android）

最近更新时间：2018-10-19 16:32:00

本文将指导您在客户端中创建一个房间，开始发布自己的个人直播秀。

源码下载

在此我们提供以下所讲到的完整 Demo 代码，如有需要请您自行下载。
Demo 代码下载

相关概念

- 房间
- privateMapKey
- 角色配置
- 渲染控件
  在拿到视频数据时，需要一个展示数据的平台，这个就是渲染控件，可以实际对应到一个 Android 控件。

添加渲染控件

首先，需要在前面的 demo 布局中添加一个控件来渲染视频：

```
<com.tencent.ilivesdk.view.AVRootView
android:id="@+id/av_root_view"
android:layout_width="match_parent"
android:layout_height="match_parent" />
```

创建房间模块

定义一个房间模块与 Activity 通讯的接口：

```
public interface IRoomView {
// 进入房间成功
void onEnterRoom();
// 进房间失败
```

```
void onEnterRoomFailed(String module, int errCode, String errMsg);

// 退出房间成功
void onQuitRoomSuccess();
// 退出房间失败
void onQuitRoomFailed(String module, int errCode, String errMsg);

// 房间断开
void onRoomDisconnect(String module, int errCode, String errMsg);
}
```

然后创建房间模块：

```
public class RoomHelper implements ILiveRoomOption.onExceptionListener, ILiveRoomOption.onRoo
mDisconnectListener {
private IRoomView roomView;

public RoomHelper(IRoomView view){
roomView = view;
}
// 设置渲染控件
public void setRootView(AVRootView avRootView){
ILiveRoomManager.getInstance().initAvRootView(avRootView);
}
// 创建房间
public int createRoom(int roomId){
ILiveRoomOption option = new ILiveRoomOption()
.imsupport(false) // 不需要IM功能
.exceptionListener(this) // 监听异常事件处理
.roomDisconnectListener(this) // 监听房间中断事件
.controlRole("user") // 使用user角色
.autoCamera(true) // 进房间后自动打开摄像头并上行
.autoMic(true); // 进房间后自动要开Mic并上行

return ILiveRoomManager.getInstance().createRoom(roomId, option, new ILiveCallBack() {
@Override
public void onSuccess(Object data) {
roomView.onEnterRoom();
}

@Override
public void onError(String module, int errCode, String errMsg) {
roomView.onEnterRoomFailed(module, errCode, errMsg);
}
});
}
// 退出房间
public int quitRoom(){
```

```
return ILiveRoomManager.getInstance().quitRoom(new ILiveCallBack() {
@Override
public void onSuccess(Object data) {
roomView.onQuitRoomSuccess();
}

@Override
public void onError(String module, int errCode, String errMsg) {
roomView.onQuitRoomFailed(module, errCode, errMsg);
}
});
}

// 处理Activity事件
public void onPause(){
ILiveRoomManager.getInstance().onPause();
}
public void onResume(){
ILiveRoomManager.getInstance().onResume();
}

@Override
public void onException(int exceptionId, int errCode, String errMsg) {
//处理异常事件
}

@Override
public void onRoomDisconnect(int errCode, String errMsg) {
// 处理房间中断(一般为断网或长时间无长行后台回收房间)
}
}
```

UI 开发

同样在房间的 Activity 的 onCreate 事件中，可以上面创建的房间模块，并设置渲染控件。

```
roomHelper = new RoomHelper(this);
// 获取渲染控件
AVRootView avRootView = findViewById(R.id.av_root_view);
// 设置没有渲染时的背景色为蓝色(注意不支持在布局中直接设置)
avRootView.getVideoGroup().setBackgroundColor(Color.BLUE);
// 设置渲染控件
roomHelper.setRootView(avRootView);
```

界面中可以允许用户输入房间号的，并根据用户输入的房间后创建房间，也可以先直接写死（测试）。

```
roomHelper.createRoom(1234);
```

如果能够上抛 onEnterRoom 事件，并且能够看到自己的视频画面，那么恭喜您的房间已经成功创建了。

常见问题

进房失败10004，提示request room server address failed

确认正确配置了进房票据privateMapKey，若控制台在【帐号信息】开启【启用权限密钥】,则privateMapKey为必填字段

进房失败71，提示decodeSsoCmd_pbvideoapp_pbvideoinfoErr:user id error longConnHead.account=0

这种情况多帐号登录引起，请确认之前登录新帐号时，已注销老的帐号

onException 中收到 EXCEPTION_ENABLE_CAMERA_FAILED 并且 errCode 为 1，找开摄像头失败。

1. 确认 Android 设备有摄像头并且可以正常使用；
2. 确认后台没有其它应用占用摄像头；
3. 如果是 Android 6.0 以上设备需要确认已申请打开摄像头的动态权限 `Manifest.permission.CAMERA` 。

失败回调，错误码 1003 或 8011

1. 进房/退房为线性互斥操作，若请求太频繁，sdk便会上抛 8011，这种情况需要上次操作完成(回调上抛)再继续操作(进出房间)；
2. 用户一次只能加入一个房间，所以若上次房间未退出，再次调用创建(或加入)便会上抛 1003，这种情况需要先退出上次房间；

创建并加入房间（iOS）

最近更新时间：2018-10-19 16:32:23

本文将指导您将如何创建一个房间，获取直播画面并在合适的时候退出房间。

源码下载

在此我们提供以下所讲到的完整 Demo 代码，如有需要请您自行下载。
Demo 代码下载

相关概念

- 房间
- privateMapKey
- 角色配置
- 渲染控件
  在拿到视频数据时，我们需要一个展示数据的地方，这个就是渲染控件。
- 视频数据类型
  腾讯云支持一个帐号同时上行一路主流和一路辅流，这里用于区分视频流的来源称为视频类型，目前支持的视频类型有： 摄像头，屏幕分享，播片（PC 端产生）。

视频类型	流类型	描述
摄像头	主流	通过摄像头采集数据产生
屏幕分享	辅流	通过分享屏幕产生
播片	辅流	通过播放视频文件产生

操作步骤

创建房间

创建房间的方法在 `ILiveRoomManager.h` 中，该方法需要传入两个重要参数，房间 ID（roomId）和房间配置对象（option）：

- 房间 ID 可以搭建一个界面（如 demo 中）让用户输入，也可在代码中 hard code 用于测试，但是 roomID 需要符合前面预备知识中介绍的规则。

- 配置对象需要您创建，ILiveRoomOption 是用来对您准备创建的这个房间的音视频，即时通信等功能进行配置的类，这里我们可以使用默认配置 defaultHostLiveOption。

最后，创建房间的结果会以回调 Block 的方式返回，您可以根据自己的业务逻辑，在成功或者失败回调中做相应的处理。

```
// 导入头文件
#import <ILiveSDK/ILiveCoreHeader.h>

// 创建房间
- (IBAction)onCreateRoom:(id)sender {
// 1. 创建live房间页面
LiveRoomViewController *liveRoomVC = [[LiveRoomViewController alloc] init];

// 2. 创建房间配置对象
ILiveRoomOption *option = [ILiveRoomOption defaultHostLiveOption];
option.imOption.imSupport = NO;
// 设置房间内音视频监听
option.memberStatusListener = liveRoomVC;
// 设置房间中断事件监听
option.roomDisconnectListener = liveRoomVC;

// 该参数代表进房之后使用什么规格音视频参数，参数具体值为客户在腾讯云实时音视频控制台画面设定中配置的角色名（例如：默认角色名为user, 可设置controlRole = @"user"）
option.controlRole = #腾讯云控制台配置的角色名#;

// 3. 调用创建房间接口，传入房间ID和房间配置对象
[[ILiveRoomManager getInstance] createRoom:[self.roomIDTF.text intValue] option:option succ:^{
// 创建房间成功，跳转到房间页
[self.navigationController pushViewController:liveRoomVC animated:YES];
} failed:^(NSString *module, int errId, NSString *errMsg) {
// 创建房间失败
NSLog(@"创建房间失败 errId:%d errMsg:%@",errId, errMsg);
}];
```

用户进入房间后，会自动打开摄像头和麦克风并自动推流（这个是在创建房间时传入的 `ILiveRoomOption` 配置对象中设置）。在直播房间中，所有的音视频事件都是通过音视频事件回调来通知监听对象，首先您需要设置这样一个监听对象。

监听对象为创建课堂时传入的 `ILiveRoomOption` 对象的一个属性 `memberStatusListener` ，由于需要将房间页面对象 `LiveRoomViewController` 设置为监听者，所以在创建房间对象时将 `option` 通过自动以的初始化方法 `initWithOption` 传到了 `LiveRoomViewController` 内部。

```
> LiveRoomViewController.m

- (instancetype)initWithOption:(ILiveRoomOption *)option {
self = [super init];
if (self) {
```

```
option.memberStatusListener = self;
}
return self;
}
```

监听房间内事件

创建房间时，我们在配置对象中设置了房间音视频事件监听和房间中断事件监听，用来监听房间内的事件。
音视频事件监听对象遵守 ILiveMemStatusListener 协议，并实现以下方法：

```
@protocol ILiveMemStatusListener <NSObject>
/**
房间成员状态变化通知的函数，房间成员发生状态变化(如是否发音频、是否发视频等)时，会通过该函数通知业务侧。

@param event 状态变化id，详见QAVUpdateEvent的定义
@param endpoints 发生状态变化的成员id列表。

@return YES 执行成功
*/
- (BOOL)onEndpointsUpdateInfo:(QAVUpdateEvent)event updateList:(NSArray *)endpoints;
@end
```

说明：
该方法即为房间音视频事件回调方法，只要房间内有人开关摄像头、麦克风等，SDK 底层就会回调该方法，通知监听者，该方法有 event 和 endpoints 两个参数。

- event 是一个事件枚举值，定义了房间内成员可能发生的事件类型，包括成员进出房间，开关摄像头、麦克风等。
- endpoints 是一个数组，里面装的是 QAVEndpoint 对象，代表每一个发送事件的用户，该方法的作用即通知哪些用户发生了哪种类型的音视频改变，在监听到事件发生时，通常需要再界面上做出一些调整，比如，监听到某用户打开了摄像头，应该添加一个渲染图到界面上，将该用户的画面渲染出来。

event 有以下一些取值：

```
typedef NS_ENUM(NSInteger, QAVUpdateEvent) {
QAV_EVENT_ID_NONE = 0, ///< 默认值，无意义。
QAV_EVENT_ID_ENDPOINT_ENTER = 1, ///< 进入房间事件。
QAV_EVENT_ID_ENDPOINT_EXIT = 2, ///< 退出房间事件。
QAV_EVENT_ID_ENDPOINT_HAS_CAMERA_VIDEO = 3, ///< 有发摄像头视频事件。
QAV_EVENT_ID_ENDPOINT_NO_CAMERA_VIDEO = 4, ///< 无发摄像头视频事件。
```

版权所有：腾讯云计算（北京）有限责任公司

```
QAV_EVENT_ID_ENDPOINT_HAS_AUDIO = 5, ///< 有发语音事件。
QAV_EVENT_ID_ENDPOINT_NO_AUDIO = 6, ///< 无发语音事件。
QAV_EVENT_ID_ENDPOINT_HAS_SCREEN_VIDEO = 7, ///< 有发屏幕视频事件。
QAV_EVENT_ID_ENDPOINT_NO_SCREEN_VIDEO = 8, ///< 无发屏幕视频事件。
QAV_EVENT_ID_ENDPOINT_HAS_MEDIA_FILE_VIDEO = 9, ///< 有发文件视频事件。
QAV_EVENT_ID_ENDPOINT_NO_MEDIA_FILE_VIDEO = 10, ///< 无发文件视频事件。
};
```

可以看到，前面提到的 "视频数据类型" 在事件中都有定义。

接着您要做的就是设置监听对象，然后在代理方法中监听摄像头开启的事件

QAV_EVENT_ID_ENDPOINT_HAS_CAMERA_VIDEO，并将该用户的渲染画面添加到界面上。

```
// 导入头文件
#import <ILiveSDK/ILiveCoreHeader.h>

// 音视频事件回调
- (BOOL)onEndpointsUpdateInfo:(QAVUpdateEvent)event updateList:(NSArray *)endpoints {
switch (event) {
case QAV_EVENT_ID_ENDPOINT_HAS_CAMERA_VIDEO:
{
/*
创建并添加渲染视图，传入userID和渲染画面类型，这里传入 QAVVIDEO_SRC_TYPE_CAMERA（摄像头画面）
*/
ILiveFrameDispatcher *frameDispatcher = [[ILiveRoomManager getInstance] getFrameDispatcher];
ILiveRenderView *renderView = [frameDispatcher addRenderAt:self.view.bounds foruserId:[endpoints.firstO
bject userId] srcType:QAVVIDEO_SRC_TYPE_CAMERA];
[self.view addSubview:renderView];
}
break;
}
return YES;
}
```

如果一切顺利，在创建房间成跳转到直播页面时，您就能看到直播画面了。

> 由于本文目的是创建直播间和获取画面，所以这里对音视频事件回调的处理比较简单，但也是最基本的，在后续文档中还会介绍该方法使用，您可以根据自己需求监听不同的事件，例如，监听成员进出房间事件，在界面上显示成员进出的通知。

房间中断事件监听遵守 `ILiveRoomDisconnectListener` 协议，并实现以下方法：

```
/**
SDK主动退出房间提示。该回调方法表示SDK内部主动退出了房间。SDK内部会因为30s心跳包超时等原因主动退出
```

```
房间，APP需要监听此退出房间事件并对该事件进行相应处理

@param reason 退出房间的原因，具体值见返回码

@return YES 执行成功
*/
- (BOOL)onRoomDisconnect:(int)reason;
```

该方法为SDK内部主动退出房间的回调，开发者可以根据自己的业务需求，在方法中进行相应处理。

退出房间

调用 ILiveSDk 的退出房间接口。
接口选择在直播控制器的 dealloc 方法中调用。

> 注意，别让您的直播控制器被循环引用，否则 dealloc 方法将不会被调用。

```
// 房间销毁时记得调用退出房间接口
- (void)dealloc {
[[ILiveRoomManager getInstance] quitRoom:^{
NSLog(@"退出房间成功");
} failed:^(NSString *module, int errId, NSString *errMsg) {
NSLog(@"退出房间失败 %d : %@", errId, errMsg);
}];
}
```

退出房间成功之后，房间内资源将被回收，包括 roomID，我们可以以一个相同的 roomID 再创建一个新的房间。

常见问题

进房失败，提示没有权限

确认正确配置了进房票据privateMapKey

> 新接入用户进房票据为必填字段，老用户(不使用进房票据)需在初始化时配置
>
> ```
> [[ILiveSDK getInstance] setChannelMode:E_ChannelIMSDK withHost:@""];
> ```

失败回调，错误码 1003 或 8011

1. 进房/退房为线性互斥操作，若请求太频繁，sdk 便会上抛 8011，这种情况需要上次操作完成(回调上抛)再继续操作(进出房间)
2. 用户一次只能加入一个房间，所以若上次房间未退出，再次调用创建(或加入)便会上抛 1003，这种情况需要先退出上次房间

版权所有：腾讯云计算（北京）有限责任公司

创建并加入房间（PC）

最近更新时间：2018-10-19 16:33:12

本文将指导您在客户端中创建一个房间，打开摄像头和麦克风，并看到自己的视频画面。

源码下载

在此我们提供以下所讲到的完整 Demo 代码，如有需要请您自行下载。
Demo 代码下载

相关概念

- 房间
- privateMapKey
- 角色配置
- 视频渲染
  在拿到视频数据时，需要将视频数据绘制显示出来，此过程即为视频渲染。

设置视频数据回调

为了拿到视频数据，在创建/加入房间之前，还需要先设置本地和远端视频数据的回调函数。

```
//本地视频数据回调
void OnLocalVideo(const LiveVideoFrame* video_frame, void* data)
{
}

//远端视频数据回调
void OnRemoteVideo(const LiveVideoFrame* video_frame, void* data)
{
}

GetILive()->setLocalVideoCallBack(OnLocalVideo, NULL);
GetILive()->setRemoteVideoCallBack(OnRemoteVideo, NULL);
```

创建房间

创建房间需要先填写一个结构体 iLiveRoomOption ,它描述了所创建房间的相关信息,示例代码如下：

```
//房间内成员状态变化通知(如打开\关闭摄像头等)
void OnMemStatusChange(E_EndpointEventId eventId, const Vector<String> &ids, void* data)
{
}

iLiveRoomOption roomOption;
roomOption.roomId = RoomId; //要创建的房间id
roomOption.authBits = AUTH_BITS_DEFAULT; //拥有所有权限
roomOption.controlRole = "LiveMaster"; //使用Spear上配置的"LiveMaster"角色
roomOption.memberStatusListener = OnMemStatusChange;//房间内成员状态变化回调
roomOption.data = NULL;//在回调中原封不动传回的void*数据指针;

GetILive()->createRoom(roomOption, [](void* data) {
//创建房间成功
}, [](const int code, const char *desc, void* data) {
//创建房间失败
}, NULL);
```

打开摄像头和麦克风

在创建房间后，此时已经在房间中了，不需要再调用加入房间；此时，就可以打开摄像头、麦克风，进行音视频数据的上行了。

```
//打开摄像头
Vector< Pair<String/*id*/, String/*name*/> > cameraList;
GetILive()->getCameraList(cameraList);//获取可用摄像头列表
if (cameraList.size() > 0 )
{
GetILive()->openCamera(cameraList[0].first); //打开第一个摄像头(默认摄像头)
}

//打开麦克风
Vector< Pair<String/*id*/, String/*name*/> > micList;
GetILive()->getMicList(micList);//获取可用麦克风列表
if (micList.size() > 0)
{
GetILive()->openMic(micList[0].first); //打开第一个麦克风(默认麦克风);
}
```

此时，前面设置的回调函数OnLocalVideo会收到每一帧视频数据了；可以在回调中打印输出每一帧视频的数据大小，验证确实收到视频数据了。

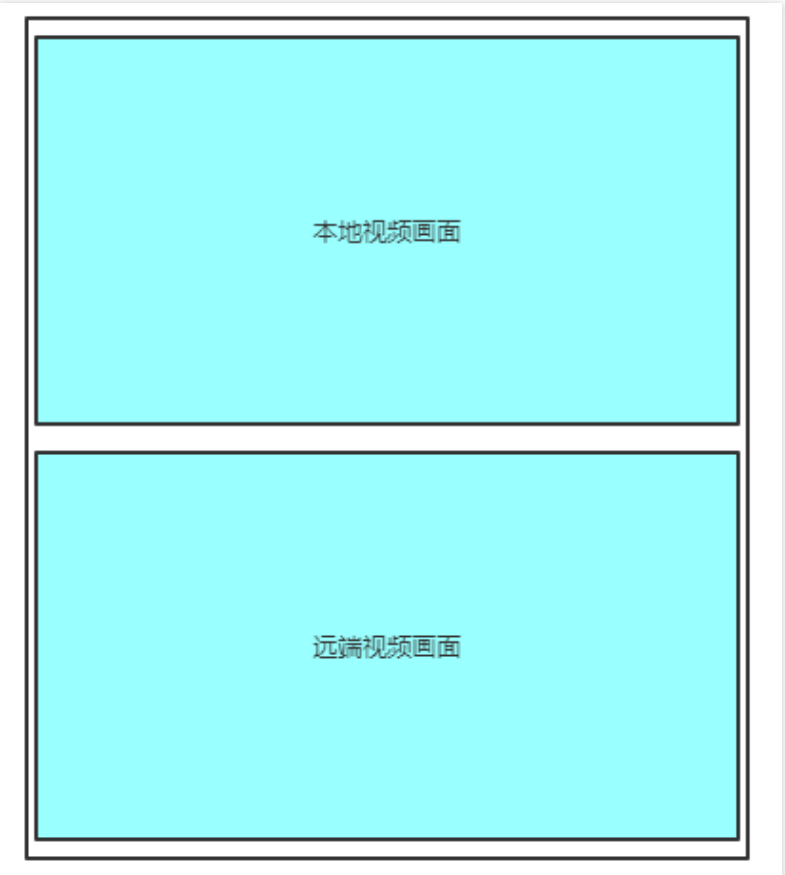
```
void OnLocalVideo(const LiveVideoFrame* video_frame, void* data)
{
printf("frame size: %d\n", video_frame->dataSize);
}
```

视频渲染

前面已经收到本地摄像头的视频数据了，要看到视频画面，还需要对收到的视频画面进行渲染。iLiveSDK 提供了渲染模块（iLiveRootView），为渲染模块指定一个窗口句柄，并往渲染模块中添加 view（要渲染画面的用户信息），最后，将每一帧视频画面传给渲染模块，即可进行渲染。请您仔细阅读 渲染模块使用文档 ,并结合本例 Demo 源码进行理解。

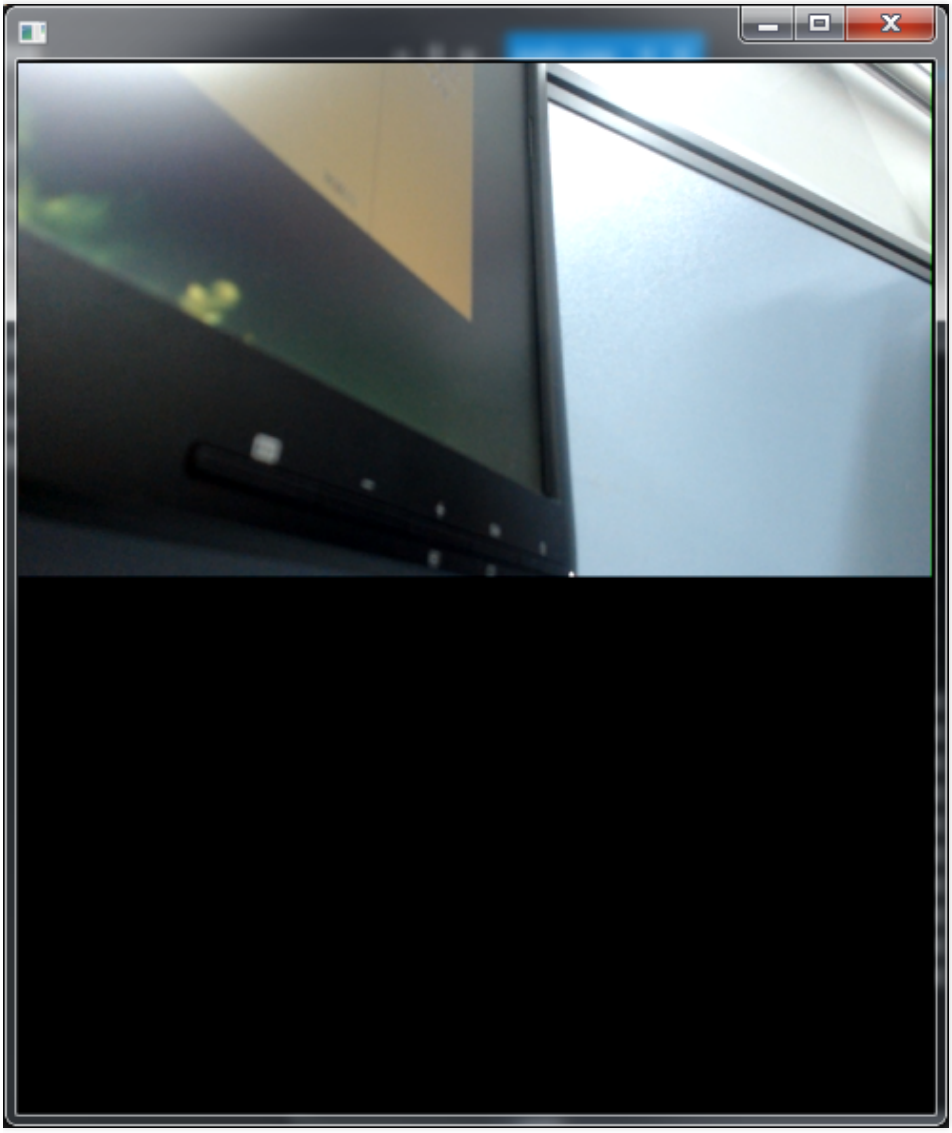
界面设计

本例 Demo，为了兼容后面课程，创建了一个窗口，上半部分显示本地视频画面，下半部分显示远端视频画面，示意图如下：



为降低 demo 的复杂度，此 demo 只演示两人音视频互通的情况，实际开发中，用户可以根据需要添加多个渲染视图。

运行结果



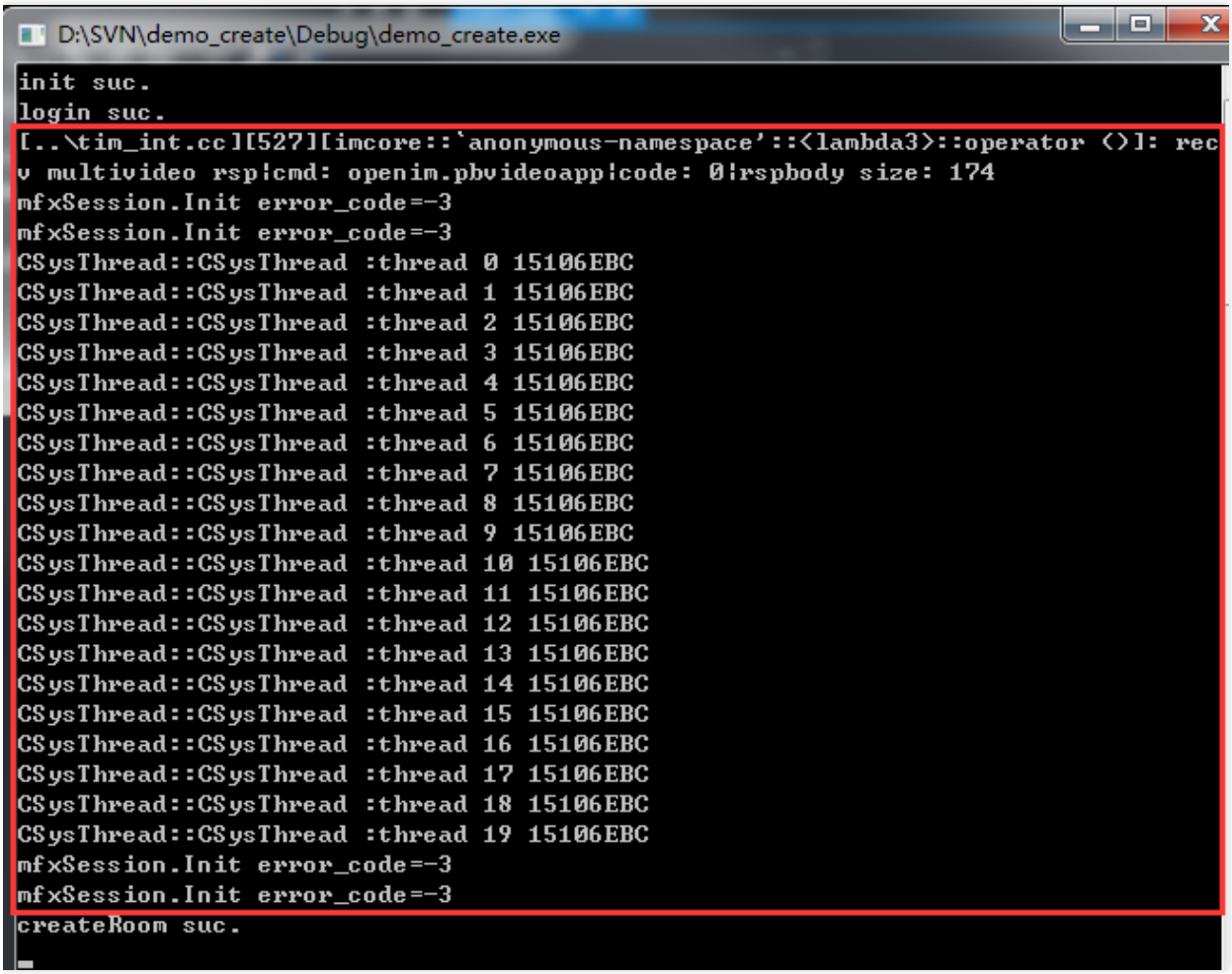
常见问题

进房失败，提示没有权限

确认正确配置了进房票据privateMapKey，若控制台在【帐号信息】开启【启用权限密钥】,则privateMapKey为必填字段

控制台输出一些无用信息:

在调用 SDK 某些接口时，控制台可能输出一些无用信息，例如，创建房间的输出如下图：



这是因为 iLiveSDK 内部使用了其他 SDK，这是其他 SDK 的打印输出信息，不会影响实际使用,忽略不管即可。

创建并加入房间（Mac）

最近更新时间：2018-10-19 16:33:28

本文将指导您将如何创建一个房间，获取直播画面并在合适的时候退出房间。

源码下载

在此我们提供以下所讲到的完整 Demo 代码，如有需要请您自行下载。
Demo 代码下载

相关概念

- 房间
- privateMapKey
- 角色配置
- 渲染控件
  在拿到视频数据时，我们需要一个展示数据的地方，这个就是渲染控件。
- 视频数据类型
  腾讯云支持一个帐号同时上行一路主流和一路辅流，这里用于区分视频流的来源称为视频类型，目前支持的视频类型有： 摄像头，屏幕分享，播片（PC 端产生）。

视频类型	流类型	描述
摄像头	主流	通过摄像头采集数据产生
屏幕分享	辅流	通过分享屏幕产生
播片	辅流	通过播放视频文件产生

操作步骤

创建房间

创建房间的方法在 `ILiveRoomManager.h` 中，该方法需要传入两个重要参数，房间 ID（roomId）和房间配置对象（option）：

- 房间 ID 可以搭建一个界面（如 demo 中）让用户输入，也可在代码中 hard code 用于测试，但是 roomID 需要符合前面预备知识中介绍的规则。

- 配置对象需要您创建，ILiveRoomOption 是用来对您准备创建的这个房间的音视频，即时通信等功能进行配置的类，这里我们可以使用默认配置 defaultHostLiveOption。

最后，创建房间的结果会以回调 Block 的方式返回，您可以根据自己的业务逻辑，在成功或者失败回调中做相应的处理。

```
> TCLiveRoomWC.m
// 导入头文件
#import <ILiveSDK/ILiveCoreHeader.h>

// 创建房间
- (void)enterRoom{
//step3 房间配置
ILiveRoomOption *option = [ILiveRoomOption defaultHostLiveOption];
option.imOption.imSupport = YES;
// 设置房间内音视频监听
option.memberStatusListener = self;
// 设置房间中断事件监听
option.roomDisconnectListener = self;
option.firstFrameListener = self;
// 该参数代表进房之后使用什么规格音视频参数，参数具体值为客户在腾讯云实时音视频控制台画面设定中配置的角色名（例如：默认角色名为user, 可设置controlRole = @"user"）
option.controlRole = self.role;

//step4 调用创建房间接口，传入房间ID和房间配置对象
[[ILiveRoomManager getInstance] createRoom:[self.roomID intValue] option:option succ:^{
NSLog(@"-----> create room succ");
} failed:^(NSString *module, int errId, NSString *errMsg) {
NSLog(@"-----> create room fail,%@ %d %@",module, errId, errMsg);
}];

}
```

用户进入房间后，会自动打开摄像头和麦克风并自动推流（这个是在创建房间时传入的 ILiveRoomOption 配置对象中设置）。在直播房间中，所有的音视频事件都是通过音视频事件回调来通知监听对象，所以需要先设置监听对象 option.memberStatusListener = self;。

监听房间内事件

创建房间时，我们在配置对象中设置了房间音视频事件监听和房间中断事件监听，用来监听房间内的事件。
音视频事件监听对象遵守 ILiveMemStatusListener 协议，并实现以下方法：

```
@protocol ILiveMemStatusListener <NSObject>
/**
房间成员状态变化通知的函数，房间成员发生状态变化(如是否发音频、是否发视频等)时，会通过该函数通知业务侧。
```

```
*@param event 状态变化id，详见QAVUpdateEvent的定义
*@param endpoints 发生状态变化的成员id列表。

*@return YES 执行成功
*/
- (BOOL)onEndpointsUpdateInfo:(QAVUpdateEvent)event updateList:(NSArray *)endpoints;
@end
```

说明：

该方法即为房间音视频事件回调方法，只要房间内有人开关摄像头、麦克风等，SDK 底层就会回调该方法，通知监听者，该方法有 event 和 endpoints 两个参数。

- event 是一个事件枚举值，定义了房间内成员可能发生的事件类型，包括成员进出房间，开关摄像头、麦克风等。
- endpoints 是一个数组，里面装的是 QAVEndpoint 对象，代表每一个发送事件的用户，该方法的作用即通知哪些用户发生了哪种类型的音视频改变，在监听到事件发生时，通常需要再界面上做出一些调整，比如，监听到某用户打开了摄像头，应该添加一个渲染图到界面上，将该用户的画面渲染出来。

event 有以下一些取值：

```
typedef NS_ENUM(NSInteger, QAVUpdateEvent) {
QAV_EVENT_ID_NONE = 0, ///< 默认值，无意义。
QAV_EVENT_ID_ENDPOINT_ENTER = 1, ///< 进入房间事件。
QAV_EVENT_ID_ENDPOINT_EXIT = 2, ///< 退出房间事件。
QAV_EVENT_ID_ENDPOINT_HAS_CAMERA_VIDEO = 3, ///< 有发摄像头视频事件。
QAV_EVENT_ID_ENDPOINT_NO_CAMERA_VIDEO = 4, ///< 无发摄像头视频事件。
QAV_EVENT_ID_ENDPOINT_HAS_AUDIO = 5, ///< 有发语音事件。
QAV_EVENT_ID_ENDPOINT_NO_AUDIO = 6, ///< 无发语音事件。
QAV_EVENT_ID_ENDPOINT_HAS_SCREEN_VIDEO = 7, ///< 有发屏幕视频事件。
QAV_EVENT_ID_ENDPOINT_NO_SCREEN_VIDEO = 8, ///< 无发屏幕视频事件。
QAV_EVENT_ID_ENDPOINT_HAS_MEDIA_FILE_VIDEO = 9, ///< 有发文件视频事件。
QAV_EVENT_ID_ENDPOINT_NO_MEDIA_FILE_VIDEO = 10, ///< 无发文件视频事件。
};
```

可以看到，前面提到的 "视频数据类型" 在事件中都有定义。

接着您要做的就是设置监听对象，然后在代理方法中监听摄像头开启的事件 QAV_EVENT_ID_ENDPOINT_HAS_CAMERA_VIDEO，并将该用户的渲染画面添加到界面上。

```
// 导入头文件
#import <ILiveSDK/ILiveCoreHeader.h>
```

```
// 音视频事件回调
- (BOOL)onEndpointsUpdateInfo:(QAVUpdateEvent)event updateList:(NSArray *)endpoints {
switch (event) {
case QAV_EVENT_ID_ENDPOINT_HAS_CAMERA_VIDEO:
{
/*
创建并添加渲染视图，传入userID和渲染画面类型，这里传入 QAVVIDEO_SRC_TYPE_CAMERA（摄像头画面）
*/
ILiveFrameDispatcher *frameDispatcher = [[ILiveRoomManager getInstance] getFrameDispatcher];
ILiveRenderViewForMac *renderView = [frameDispatcher addRenderAt:CGRectMake(0, 0, self.videoLayoutV
iew.frame.size.width, self.videoLayoutView.frame.size.height - 20) forIdentifier:endoption.identifier srcType:
QAVVIDEO_SRC_TYPE_CAMERA];
renderView.identifier = endoption.identifier;
[self.window.contentView addSubview:renderView];
}
break;
}
return YES;
}
```

如果一切顺利，在创建房间成跳转到直播页面时，您就能看到直播画面了。

> 由于本文目的是创建直播间和获取画面，所以这里对音视频事件回调的处理比较简单，但也是最基本的，在后续文档中还会介绍该方法使用，您可以根据自己需求监听不同的事件，例如，监听成员进出房间事件，在界面上显示成员进出的通知。

房间中断事件监听遵守 `ILiveRoomDisconnectListener` 协议，并实现以下方法：

```
/**
SDK主动退出房间提示。该回调方法表示SDK内部主动退出了房间。SDK内部会因为30s心跳包超时等原因主动退出房间，APP需要监听此退出房间事件并对该事件进行相应处理

@param reason 退出房间的原因，具体值见返回码

@return YES 执行成功
*/
- (BOOL)onRoomDisconnect:(int)reason;
```

该方法为SDK内部主动退出房间的回调，开发者可以根据自己的业务需求，在方法中进行相应处理。

退出房间

调用 ILiveSDk 的退出房间接口。

接口选择在关闭窗口时执行。

```
//关闭窗口退出房间
-(void)windowWillClose:(NSNotification *)notification{
[[ILiveRoomManager getInstance] quitRoom:^{
NSLog(@"-----> quit room succ");
} failed:^(NSString *module, int errId, NSString *errMsg) {
NSLog(@"-----> quit room fail,%@ %d %@",module, errId, errMsg);
}];
}
```

退出房间成功之后，房间内资源将被回收，包括 roomID，我们可以以一个相同的 roomID 再创建一个新的房间。

常见问题

进房失败，提示没有权限

确认正确配置了进房票据privateMapKey

新接入用户进房票据为必填字段，老用户(不使用进房票据)需在初始化时配置

```
[[ILiveSDK getInstance] setChannelMode:E_ChannelIMSDK withHost:@""];
```

失败回调，错误码 1003 或 8011

1. 进房/退房为线性互斥操作，若请求太频繁，sdk 便会上抛 8011，这种情况需要上次操作完成(回调上抛)再继续操作(进出房间)
2. 用户一次只能加入一个房间，所以若上次房间未退出，再次调用创建(或加入)便会上抛 1003，这种情况需要先退出上次房间

加入房间
加入房间（Android）

最近更新时间：2018-10-19 16:36:56

本文将指导您的客户端加入一个房间，并自己打开摄像头麦克风与其他用户互动。

源码下载

在此我们提供以下所讲到的完整 Demo 代码，如有需要请您自行下载。
Demo 代码下载

相关概念

- 实时音视频应用
- privateMapKey
- 角色配置
- 摄像头 id （cameraId）
  Andorid 手机中一般有两个摄像头: 前置摄像头和后置摄像头，SDK 通过 cameraId 来区分。

常量	描述
ILiveConstants.NONE_CAMERA	无效摄像头 id(一般表示摄像头未开启)
ILiveConstants.FRONT_CAMERA	前置摄像头 id
ILiveConstants.BACK_CAMERA	后置摄像头 id

加入房间

加入房间与 创建房间 中的房间模块基本一致，不同的是这里需要的方式是 joinRoom。

```
// 加入房间
public int joinRoom(int roomId){
ILiveRoomOption option = new ILiveRoomOption()
.imsupport(false) // 不需要IM功能
.exceptionListener(this) // 监听异常事件处理
.roomDisconnectListener(this) // 监听房间中断事件
.controlRole("user") // 使用user角色
```

```
.autoCamera(false) // 进房间后不需要打开摄像头
.autoMic(false); // 进房间后不需打开Mic

return ILiveRoomManager.getInstance().joinRoom(roomId, option, new ILiveCallBack() {
@Override
public void onSuccess(Object data) {
roomView.onEnterRoom();
}

@Override
public void onError(String module, int errCode, String errMsg) {
roomView.onEnterRoomFailed(module, errCode, errMsg);
}
});
}
```

打开采集设备（摄像头和麦克风）

如果需要上行音视频与其他用户互动，您就需要在房间模块中添加两个接口分别用于控制摄像头和麦克风。

```
// 摄像头
public int enableCamera(int cameraId, boolean enable){
return ILiveRoomManager.getInstance().enableCamera(cameraId, enable);
}
// 麦克风
public int enableMic(boolean enable){
return ILiveRoomManager.getInstance().enableMic(enable);
}
```

UI开发

界面中我们可以丰富一些，在渲染控件上层布局一组按钮，用于切换角色，开关摄像头和麦克风。
这里就不具体赘述。

常见问题

进房失败，提示没有权限

确认正确配置了进房票据privateMapKey，若控制台在【帐号信息】开启【启用权限密钥】,则privateMapKey为必填字段

加入房间（iOS）

最近更新时间：2018-10-19 16:37:46

本文将指导您的观众端如何加入一个直播房间，并打开摄像头麦克风与其他用户视频互动。

源码下载

在此我们提供以下所讲到的完整 Demo 代码，如有需要请您自行下载。
Demo 代码下载

加入房间

加入房间的方法在 ILiveRoomManager.h 中，跟创建房间的方法一样，该方法也需要传入两个参数，房间 ID roomId 和房间配置对象 option （加入房间的 roomId 要和创建的 roomId 一致），创建配置对象时，我们要关闭自动打开摄像头和麦克风。

```
- (IBAction)onJoinRoom:(id)sender {
// 1. 创建live房间页面
LiveRoomViewController *liveRoomVC = [[LiveRoomViewController alloc] init];

// 2. 创建房间配置对象
ILiveRoomOption *option = [ILiveRoomOption defaultHostLiveOption];
option.imOption.imSupport = NO;
// 不自动打开摄像头
option.avOption.autoCamera = NO;
// 不自动打开mic
option.avOption.autoMic = NO;
// 设置房间内音视频监听
option.memberStatusListener = liveRoomVC;
// 设置房间中断事件监听
option.roomDisconnectListener = liveRoomVC;

// 该参数代表进房之后使用什么规格音视频参数，参数具体值为客户在腾讯云实时音视频控制台画面设定中配置的角色名（例如：默认角色名为user, 可设置controlRole = @"user"）
option.controlRole = #腾讯云控制台配置的角色名#;

// 3. 调用创建房间接口，传入房间ID和房间配置对象
[[ILiveRoomManager getInstance] joinRoom:[self.roomIDTF.text intValue] option:option succ:^{
// 加入房间成功，跳转到房间页
[self.navigationController pushViewController:liveRoomVC animated:YES];animated:YES];
} failed:^(NSString *module, int errId, NSString *errMsg) {
// 加入房间失败
```

```
NSLog(@"加入房间失败 errId:%d errMsg:%@",errId, errMsg);
}];
}
```

加入房间成功后直接跳转到房间页面。

视频互动

要与房间内其他用户进行视频互动，只需要打开摄像头和麦克风即可：

```
/**
打开/关闭 相机

@param cameraPos 相机位置 cameraPos
@param bEnable YES:打开 NO:关闭
@param succ 成功回调
@param fail 失败回调
*/
- (void)enableCamera:(cameraPos)cameraPos enable:(BOOL)bEnable succ:(TCIVoidBlock)succ failed:(TCIErrorBlock)fail;

/**
打开/关闭 麦克风

@param bEnable YES:打开 NO:关闭
@param succ 成功回调
@param fail 失败回调
*/
- (void)enableMic:(BOOL)bEnable succ:(TCIVoidBlock)succ failed:(TCIErrorBlock)fail;
```

在触发相应的动作之后，调用对应的接口即可。

音视频事件监听

音视频事件回调方法中需要处理多人的事件，主要是对摄像头开关事件的处理。

> 检测到有人开启了摄像头 QAV_EVENT_ID_ENDPOINT_HAS_CAMERA_VIDEO，就添加该用户的渲染视图；
> 检测到有人关闭了摄像头 QAV_EVENT_ID_ENDPOINT_NO_CAMERA_VIDEO，则移除该用户的渲染视图。

渲染视图的 frame 不是固定的，需要根据当前渲染视图的个数和产品需求进行计算。在本文提供的 Demo 中，简单的对渲染视图做了一个从上到下等分的布局，您也可根据需要自己定义布局逻辑。

```
// 音视频事件回调
- (BOOL)onEndpointsUpdateInfo:(QAVUpdateEvent)event updateList:(NSArray *)endpoints {
if (endpoints.count <= 0) {
return NO;
}
for (QAVEndpoint *endpoint in endpoints) {
switch (event) {
case QAV_EVENT_ID_ENDPOINT_HAS_CAMERA_VIDEO:
{
/*
创建并添加渲染视图，传入userID和渲染画面类型，这里传入 QAVVIDEO_SRC_TYPE_CAMERA（摄像头画面）
*/
ILiveFrameDispatcher *frameDispatcher = [[ILiveRoomManager getInstance] getFrameDispatcher];
ILiveRenderView *renderView = [frameDispatcher addRenderAt:CGRectZero foruserId:endpoint.userId srcTy
pe:QAVVIDEO_SRC_TYPE_CAMERA];
[self.view addSubview:renderView];
[self.view sendSubviewToBack:renderView];
// 房间内打开摄像头用户数量变化，重新布局渲染视图
[self onCameraNumChange];
}
break;
case QAV_EVENT_ID_ENDPOINT_NO_CAMERA_VIDEO:
{
// 移除渲染视图
ILiveFrameDispatcher *frameDispatcher = [[ILiveRoomManager getInstance] getFrameDispatcher];
ILiveRenderView *renderView = [frameDispatcher removeRenderViewFor:endpoint.userId srcType:QAVVIDE
O_SRC_TYPE_CAMERA];
[renderView removeFromSuperview];
// 房间内打开摄像头用户数量变化，重新布局渲染视图
[self onCameraNumChange];
}
break;
default:
break;
}
}
return YES;
}


// 房间内打开摄像头用户数量变化时调用，重新布局所有渲染视图，这里简单处理，从上到下等分布局
- (void)onCameraNumChange {
// 获取当前所有渲染视图
NSArray *allRenderViews = [[TILLiveManager getInstance] getAllAVRenderViews];

// 检测异常情况
if (allRenderViews.count == 0) {
```

```
return;
}

// 计算并设置每一个渲染视图的frame
CGFloat renderViewHeight = [UIScreen mainScreen].bounds.size.height / allRenderViews.count;
CGFloat renderViewWidth = [UIScreen mainScreen].bounds.size.width;
__block CGFloat renderViewY = 0.f;
CGFloat renderViewX = 0.f;

[allRenderViews enumerateObjectsUsingBlock:^(ILiveRenderView *renderView, NSUInteger idx, BOOL * _No
nnull stop) {
renderViewY = renderViewY + renderViewHeight * idx;
CGRect frame = CGRectMake(renderViewX, renderViewY, renderViewWidth, renderViewHeight);
renderView.frame = frame;
}];
}
```

注意：

1. 添加渲染视图时，客户不用担心重复添加，SDK 内部对同一用户的同一种类型视频源的渲染视图只会添加一次。
2. 目前 SDK 内部限制了同时最多可以存在 10 个渲染视图。

常见问题

进房失败，提示没有权限

确认正确配置了进房票据privateMapKey，若控制台在【帐号信息】开启【启用权限密钥】,则privateMapKey为必填字段

切换角色失败，错误码 - 1 。

这表示配置后台找不到要切换角色，这里需要确认角色名是否填写正常（区分大小写）。

加入房间（PC）

最近更新时间：2018-10-19 16:38:43

本文将指导您的客户端加入之前所创建的房间，并与其他用户音视频互动。

源码下载

在此我们提供以下所讲到的完整 Demo 代码，如有需要请您自行下载。
单击下载

加入房间

与创建房间类似，需要先初始化、登录之后才能加入房间；加入房间也需要先填写 iLiveRoomOption 结构体，用于描述所加入房间的相关信息，然后调用 joinRoom() 接口进行加入房间。

```
//房间内成员状态变化通知(如打开\关闭摄像头等)
void OnMemStatusChange(E_EndpointEventId eventId, const Vector<String> &ids, void* data)
{
}

iLiveRoomOption roomOption;
roomOption.roomId = RoomId; //要加入的房间id
roomOption.authBits = AUTH_BITS_DEFAULT; //拥有所有权限
roomOption.controlRole = "user"; //使用Spear上配置的"user"角色
roomOption.memberStatusListener = OnMemStatusChange;//房间内成员状态变化回调
roomOption.data = NULL;//在回调中原封不动传回的void*数据指针;

GetILive()->joinRoom(roomOption, [](void* data) {
//加入房间成功
}, [](const int code, const char *desc, void* data) {
//加入房间失败
}, NULL);
```

打开摄像头和麦克风

进入房间后，就可以打开摄像头和麦克风与其他用户互动了,打开摄像头和麦克风的方式与之前完全一样。
详细请参考 创建房间-打开摄像头和麦克风。

远端视频渲染

加入房间后，SDK 会自动请求房间内其他成员的视频画面,即此时在此回调中就能拿到远端视频数据了；您需要按照之前视频渲染的方式，把远端画面渲染出来。详细请参考 创建房间-视频渲染。

至此，房间内成员便可以进行音视频互动。

源码说明

- 测试说明
  本文 Demo 和 创建房间 的完整 Demo 进行互通测试时，需要分别在两台电脑上进行测试，因为代码中都写死打开第一个摄像头和第一个麦克风；如果要在一台电脑上测试，需要电脑至少有两个摄像头及麦克风，且两个用户不能使用同一个设备。

运行结果

常见问题

进房失败，提示没有权限

确认正确配置了进房票据privateMapKey，若控制台在【帐号信息】开启【启用权限密钥】,则privateMapKey为必填字段

加入房间（Mac）

最近更新时间：2018-10-19 16:39:29

本文将指导您的观众端如何加入一个直播房间，并打开摄像头麦克风与其他用户视频互动。

源码下载

在此我们提供以下所讲到的完整 Demo 代码，如有需要请您自行下载。
Demo 代码下载

加入房间

加入房间的方法在 ILiveRoomManager.h 中，跟创建房间的方法一样，该方法也需要传入两个参数，房间 ID roomId 和房间配置对象 option （加入房间的 roomId 要和创建的 roomId 一致），创建配置对象时，我们要关闭自动打开摄像头和麦克风。

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

// 创建房间配置对象
ILiveRoomOption *option = [ILiveRoomOption defaultHostLiveOption];
option.imOption.imSupport = NO;
// 不自动打开摄像头
option.avOption.autoCamera = NO;
// 不自动打开mic
option.avOption.autoMic = NO;
// 设置房间内音视频监听
option.memberStatusListener = liveRoomVC;
// 设置房间中断事件监听
option.roomDisconnectListener = liveRoomVC;

// 该参数代表进房之后使用什么规格音视频参数，参数具体值为客户在腾讯云实时音视频控制台画面设定中配置的角色名（例如：默认角色名为user, 可设置controlRole = @"user"）
option.controlRole = #腾讯云控制台配置的角色名#;

// 调用加入房间接口，传入房间ID和房间配置对象
[[ILiveRoomManager getInstance] joinRoom:[self.roomID intValue] option:option succ:^{
// 加入房间成功
NSLog(@"-----> join room succ");
} failed:^(NSString *module, int errId, NSString *errMsg) {
// 加入房间失败
NSLog(@"加入房间失败 errId:%d errMsg:%@",errId, errMsg);
```

```
}];
}
```

加入房间成功后直接跳转到房间页面。

视频互动

要与房间内其他用户进行视频互动，只需要打开摄像头和麦克风即可：

```
/**
打开/关闭 相机

@param cameraPos 相机位置 cameraPos
@param bEnable YES:打开 NO:关闭
@param succ 成功回调
@param fail 失败回调
*/
- (void)enableCamera:(cameraPos)cameraPos enable:(BOOL)bEnable succ:(TCIVoidBlock)succ failed:(TCIErrorBlock)fail;

/**
打开/关闭 麦克风

@param bEnable YES:打开 NO:关闭
@param succ 成功回调
@param fail 失败回调
*/
- (void)enableMic:(BOOL)bEnable succ:(TCIVoidBlock)succ failed:(TCIErrorBlock)fail;
```

在触发相应的动作之后，调用对应的接口即可。

音视频事件监听

音视频事件回调方法中需要处理多人的事件，主要是对摄像头开关事件的处理。

> 检测到有人开启了摄像头 QAV_EVENT_ID_ENDPOINT_HAS_CAMERA_VIDEO，就添加该用户的渲染视图；
> 检测到有人关闭了摄像头 QAV_EVENT_ID_ENDPOINT_NO_CAMERA_VIDEO，则移除该用户的渲染视图。

渲染视图的 frame 不是固定的，需要根据当前渲染视图的个数和产品需求进行计算。在本文提供的 Demo 中，支持4路视频同时展示，对渲染视图做了一个主画面大图在下面，其他3录小画面横向排布在上面，您也可根据需要自己定义布局逻辑。

```
// 音视频事件回调
- (BOOL)onEndpointsUpdateInfo:(QAVUpdateEvent)event updateList:(NSArray *)endpoints {
if (endpoints.count <= 0) {
return NO;
}
for (QAVEndpoint *endpoint in endpoints) {
switch (event) {
case QAV_EVENT_ID_ENDPOINT_HAS_CAMERA_VIDEO:
{
/*
创建并添加渲染视图，传入userID和渲染画面类型，这里传入 QAVVIDEO_SRC_TYPE_CAMERA（摄像头画面）
*/
ILiveFrameDispatcher *frameDispatcher = [[ILiveRoomManager getInstance] getFrameDispatcher];
ILiveRenderViewForMac *renderView = [frameDispatcher addRenderAt:CGRectZero forIdentifier:endoption.
identifier srcType:QAVVIDEO_SRC_TYPE_CAMERA];
renderView.identifier = endoption.identifier;
// 房间内打开摄像头用户数量变化，重新布局渲染视图
[self updateVideoFrame:renderView];
}
break;
case QAV_EVENT_ID_ENDPOINT_NO_CAMERA_VIDEO:
{
// 移除渲染视图
ILiveFrameDispatcher *frameDispatcher = [[ILiveRoomManager getInstance] getFrameDispatcher];
ILiveRenderViewForMac *renderView = [frameDispatcher removeRenderViewFor:endoption.identifier srcTy
pe:QAVVIDEO_SRC_TYPE_CAMERA];
[renderView removeFromSuperview];

// 房间内打开摄像头用户数量变化，重新布局渲染视图
[self updateVideoFrame:nil];
}
break;
default:
break;
}
}
return YES;
}


// 房间内打开摄像头用户数量变化时调用，重新布局所有渲染视图
- (void)updateVideoFrame:(ILiveRenderViewForMac *)renderView{
if (renderView && ![self.window.contentView.subviews containsObject:renderView]) {
[self.videoLayoutView addSubview:renderView];
}
NSArray *allRenderView = [[[ILiveRoomManager getInstance] getFrameDispatcher] getAllRenderViews];
```

```
if (allRenderView.count == 1) {
ILiveRenderViewForMac *bigView = allRenderView[0];
bigView.frame = CGRectMake(0, 0, self.videoLayoutView.frame.size.width, self.videoLayoutView.frame.size.
height - 20);
[bigView viewWithTag:1001].frame = CGRectMake(bigView.frame.size.width/2, bigView.frame.size.height -
10, 300, 20);
}
else if (allRenderView.count == 2){
ILiveRenderViewForMac *bigView = allRenderView[0];
ILiveRenderViewForMac *smallView1 = allRenderView[1];
bigView.frame = CGRectMake(0, 0, self.videoLayoutView.frame.size.width, self.videoLayoutView.frame.size.
height - 160 );
smallView1.frame = CGRectMake(self.videoLayoutView.frame.size.width/2 - 90, self.window.frame.size.heig
ht - 150, 180, 120);

}
else if (allRenderView.count == 3){
ILiveRenderViewForMac *bigView = allRenderView[0];
ILiveRenderViewForMac *smallView1 = allRenderView[1];
ILiveRenderViewForMac *smallView2 = allRenderView[2];
bigView.frame = CGRectMake(0, 0, self.videoLayoutView.frame.size.width, self.videoLayoutView.frame.size.
height - 160 );
smallView1.frame = CGRectMake(self.videoLayoutView.frame.size.width/2 - 180 - 10, self.window.frame.siz
e.height - 150, 180, 120);
smallView2.frame = CGRectMake(self.videoLayoutView.frame.size.width/2 + 10, self.window.frame.size.heig
ht - 150, 180, 120);
}
else if (allRenderView.count == 4 || allRenderView.count > 4 ){
ILiveRenderViewForMac *bigView = allRenderView[0];
ILiveRenderViewForMac *smallView1 = allRenderView[1];
ILiveRenderViewForMac *smallView2 = allRenderView[2];
ILiveRenderViewForMac *smallView3 = allRenderView[3];
bigView.frame = CGRectMake(0, 0, self.videoLayoutView.frame.size.width, self.videoLayoutView.frame.size.
height - 160 );
smallView1.frame = CGRectMake(self.videoLayoutView.frame.size.width/2 - 180 - 90 - 10, self.window.fram
e.size.height - 150, 180, 120);
smallView2.frame = CGRectMake(self.videoLayoutView.frame.size.width/2 - 90, self.window.frame.size.heig
ht - 150, 180, 120);
smallView3.frame = CGRectMake(self.videoLayoutView.frame.size.width/2 + 90 + 10, self.window.frame.siz
e.height - 150, 180, 120);
}
}
```

注意：

1. 添加渲染视图时，客户不用担心重复添加，SDK 内部对同一用户的同一种类型视频源的渲染视图只会添加一次。

2. 目前 SDK 内部限制了同时最多可以存在 10 个渲染视图。

常见问题

进房失败，提示没有权限

确认正确配置了进房票据privateMapKey，若控制台在【帐号信息】开启【启用权限密钥】,则privateMapKey为必填字段

切换角色失败，错误码 - 1 。

这表示配置后台找不到要切换角色，这里需要确认角色名是否填写正常（区分大小写）。

发送消息
发送消息（Android）

最近更新时间：2018-06-25 15:24:40

本文将指导您的客户端使用IM功能，在房间内收发消息。

源码下载

在此我们提供以下所讲到的完整 Demo 代码，如有需要请您自行下载。
Demo 代码下载

相关概念

本课程涉及到的概念有：

- 群组系统

类型	名称	类	描述
ILiveMessage.ILIVE_MSG_TYPE_TEXT	文本消息	ILiveTextMessage	消息内容为一个字符串(String 类型)
ILiveMessage.ILIVE_MSG_TYPE_CUSTOM	自定义消息	ILiveCustomMessage	消息内容为一个 byte 数组
ILiveMessage.ILIVE_MSG_TYPE_OTHER	其它消息	ILiveOtherMessage	IMSDK消息类型 其它消息

开启IM功能

修改房间模块中的创建和加入房间，配置 imsupport 为true 。

> 在配置imsupport为true时，createRoom会自动创建IM群组，quitRoom时创建者会自动解散群组

```
// 创建房间
public int createRoom(int roomId){
ILiveRoomOption option = new ILiveRoomOption()
.imsupport(true) // 开启IM功能
```

```
.groupType("AVChatRoom") // 使用实时音视频聊天室(默认)
.exceptionListener(this)
.roomDisconnectListener(this)
.controlRole("LiveMaster")
.autoCamera(true)
.autoMic(true);

return ILiveRoomManager.getInstance().createRoom(roomId, option, new ILiveCallBack() {
@Override
public void onSuccess(Object data) {
roomView.onEnterRoom();
}

@Override
public void onError(String module, int errCode, String errMsg) {
roomView.onEnterRoomFailed(module, errCode, errMsg);
}
});
}
// 加入房间
public int joinRoom(int roomId){
ILiveRoomOption option = new ILiveRoomOption()
.imsupport(true) // 开启IM功能
.groupType("AVChatRoom") // 使用实时音视频聊天室(默认),与创建一致
.exceptionListener(this)
.roomDisconnectListener(this)
.controlRole("Guest")
.autoCamera(false)
.autoMic(false);

return ILiveRoomManager.getInstance().createRoom(roomId, option, new ILiveCallBack() {
@Override
public void onSuccess(Object data) {
roomView.onEnterRoom();
}

@Override
public void onError(String module, int errCode, String errMsg) {
roomView.onEnterRoomFailed(module, errCode, errMsg);
}
});
}
```

设置消息监听

为了全局监听，可以在每个应用中都可以监听到消息，可以创建一个消息观察者：

```
/**
* 消息观察者
*/
public class MessageObservable implements ILiveMessageListener {
// 消息监听链表
private LinkedList<ILiveMessageListener> listObservers = new LinkedList<>();
// 句柄
private static MessageObservable instance;


public static MessageObservable getInstance(){
if (null == instance){
synchronized (MessageObservable.class){
if (null == instance){
instance = new MessageObservable();
}
}
}
return instance;
}

// 添加观察者
public void addObserver(ILiveMessageListener listener){
if (!listObservers.contains(listener)){
listObservers.add(listener);
}
}

// 移除观察者
public void deleteObserver(ILiveMessageListener listener){
listObservers.remove(listener);
}

@Override
public void onNewMessage(ILiveMessage message) {
// 拷贝链表
LinkedList<ILiveMessageListener> tmpList = new LinkedList<>(listObservers);
for (ILiveMessageListener listener : tmpList){
listener.onNewMessage(message);
}
}
}
```

并在 Application 初始化的地方设置监听：

```
if (MsfSdkUtils.isMainProcess(this)) {
ILiveSDK.getInstance().initSdk(this, Constants.SDKAPPID, Constants.ACCOUNTTYPE);
// 初始化iLiveSDK房间管理模块并设置消息监听
ILiveRoomManager.getInstance().init(new ILiveRoomConfig()
.setRoomMsgListener(MessageObservable.getInstance()));
}
```

然后只需要在关心消息的Acitivty的onCreate中设置消息监听：

```
MessageObservable.getInstance().addObserver(this);
```

并在onDestory中移除消息：

```
MessageObservable.getInstance().deleteObserver(this);
```

这样在收到消息时，就可以在Acitivty中收到 onNewMessage 事件。

发送消息

这里以发送文本消息为例，您可以直接在房间模块中添加一个发送群文本消息的接口：

```
public void sendGroupMsg(String test){
ILiveMessage message = new ILiveTextMessage(test);
ILiveRoomManager.getInstance().sendGroupMessage(message, new ILiveCallBack() {
@Override
public void onSuccess(Object data) {
// 处理发消息成功
}

@Override
public void onError(String module, int errCode, String errMsg) {
// 处理发消息失败
}
});
}
```

UI开发

本文 Demo 中需要一个展示消息列表的地方，推荐使用 ListView 控件，可以盖在渲染控件上，具体实现可以自己定义。

常见问题

- 加入房间失败，错误模块 IMSDK，错误码 10010。

> 这表示要加入的IM群组不存在，需要检测是否先创建了群组（创建房间 imsupport 为 true 时会自动创建群组），并确认群组类型一致。

联系邮箱

如果对上述文档有不明白的地方，请反馈到trtcfb@qq.com

发送消息（iOS）

最近更新时间：2018-06-25 15:24:56

本文将指导您实现房间内的用户如何收发 IM 消息。

源码下载

在此我们提供以下所讲到的完整 Demo 代码，如有需要请您自行下载。
Demo 代码下载

相关概念

- 群组系统

名称	类	描述
文本消息	ILVLiveTextMessage	消息内容为一个字符串(String 类型)
自定义消息	ILVLiveCustomMessage	消息内容为一个 NSData 对象
其它消息	TIMMessage	IMSDK 消息类型 其它消息的封装

开启IM功能

首先我们得开启IM功能，修改创建和加入房间时的配置属性 imSupport 为YES，即开启了房间内的IM功能。

> 在配置imSupport为YES时，createRoom会自动创建IM群组，quitRoom时创建者会自动解散群组

```
option.imOption.imSupport = YES;
```

发送消息

房间内成员可以发送实时消息交流互动，这个功能是由 IMSDK 提供的，在 ILiveRoomManager 中对 IMSDK 的接口进行了封装，便于使用，先来看看如何发送消息。

版权所有：腾讯云计算（北京）有限责任公司

```
/**
发送C2C消息

@param dstUser 接收方ID
@param message IM消息
@param succ 发送成功回调
@param fail 发送失败回调
*/
- (void)sendC2CMessage:(NSString *)dstUser message:(TIMMessage *)message succ:(TCIVoidBlock)succ fail
ed:(TCIErrorBlock)fail;

/**
发送在线C2C消息

@param dstUser 接收方ID
@param message IM消息
@param succ 发送成功回调
@param fail 发送失败回调
*/
- (void)sendOnlineC2CMessage:(NSString *)dstUser message:(TIMMessage *)message succ:(TCIVoidBlock)s
ucc failed:(TCIErrorBlock)fail;

/**
发送Group消息
此处发送group，仅限于在当前直播间中发送group消息,或者绑定过IM群组id

@param message IM消息
@param succ 发送成功回调
@param fail 发送失败回调
*/
- (void)sendGroupMessage:(TIMMessage *)message succ:(TCIVoidBlock)succ failed:(TCIErrorBlock)fail;

/**
发送Group消息

@param message IM消息
@param succ 发送成功回调
@param fail 发送失败回调
*/
- (void)sendOnlineGroupMessage:(TIMMessage *)message succ:(TCIVoidBlock)succ failed:(TCIErrorBlock)fa
il;
```

注释：
这里可以看到消息分为群组类型和 C2C 类型（个人发给个人，可以理解为私聊），且分别提供了两种接口，在线消息和非在线消息，它们的区别就是，服务器不会存储在线消息，消息发出时，要是接收者不在线，下次接收者登录

后也不会收到发送给自己的在线消息，而非在线消息则会被服务器存储，如果发送时接收者不在线，在他下次登录时会再次发给他。

接口使用起来也比较方便，创建一个想要发送的消息类型对象，然后调用方法即可。
这里以发送一条群组文本消息为例（实时音视频的群组中一般都是此类消息）：

```
// 1. 创建消息对象
TIMMessage *msg = [[TIMMessage alloc] init];

// 1.1 设置消息文本
TIMTextElem *textElem = [[TIMTextElem alloc] init];
textElem.text = @"Hello ILiveSDK!";
[msg addElem:textElem];

// 2. 调用接口发送消息
[[ILiveRoomManager getInstance] sendGroupMessage:msg succ:^{
NSLog(@"消息发送成功");
} failed:^(NSString *module, int errId, NSString *errMsg) {
NSLog(@"消息发送失败");
}];
```

接收消息

房间内所有的音视频事件都是通过音视频回调方法通知，IM 消息也是一样，房间内所有类型的消息都是通过各自的回调方法通知，只需要设置监听对象，然后实现相应的监听回调方法即可：

```
> LiveRoomViewController.m

// 设置IM消息监听
[[[ILiveSDK getInstance] getTIMManager] addMessageListener:self];

// 实现代理协议方法
#pragma mark - TIMMessageListener
/**
* 新消息回调通知
*
* @param msgs 新消息列表，TIMMessage 类型数组
*/
- (void)onNewMessage:(NSArray*) msgs {
for (TIMMessage *msg in msgs) {
TIMConversation *conversation = [msg getConversation];
int count = [msg elemCount];
for(int i = 0; i < count; i++) {
```

```
TIMElem *elem = [msg getElem:i];
if([elem isKindOfClass:[TIMTextElem class]]){
// 接收到房间内其他成员发出的文本消息
}
}
}
}
```

注释：
实现细节：在我们调用创建房间或者加入房间的接口时，方法内部会调用 IMSDK 的接口，将用户加入一个 IM 群组（ 即本节名词解释中介绍的实时音视频聊天室 ），用户发送的消息会转发给群组内所有人，也就是同一个房间内的所有人都能收到其他人发出的群组消息（ 自己发出的群组消息自己不会收到 ）。

UI开发

收发消息时，房间内应该有一个公屏消息列表来展示这些消息，本文 Demo 只是简单实现了一个展示效果，您可以根据自己的需求自己实现。

常见问题

- 加入房间失败，错误模块 IMSDK，错误码 10010。

  这表示要加入的IM群组不存在，需要检测是否先创建了群组(创建房间时传入的配置对象中 imsupport 为 true 时会自动创建群组)，并确认群组类型一致。

联系邮箱

如果对上述文档有不明白的地方，请反馈到trtcfb@qq.com

发送消息（PC）

最近更新时间：2018-06-25 15:25:12

本文将指导您的客户端如何使用 IM （即时通讯）功能，在房间内收发消息。

源码下载

在此我们提供以下所讲到的完整 Demo 代码，如有需要请您自行下载。
Demo 代码下载

相关概念

- 群组系统

类型	名称	类	描述
TEXT	文本消息	MessageTextElem	消息内容为一个字符串（String 类型）
CUSTOM	自定义消息	MessageCustomElem	消息内容包含 data 和 ext 字段，其中可以放入用户自定义的数据
IMAGE	图片	MessageImageElem	消息内容为发送\接收的图片信息
FACE	表情消息	MessageFaceElem	表情消息一般传输一个表情的索引值，收到索引后，显示对应的表情

开启IM功能

初始化 SDK

```
GetILive()->init(SDKAppId, AccountType, false);
```

这里最后一个参数 imSupport 被设置为 false，此参数表示是否使用 IM 功能；要使用 IM 收发消息，就需要开启 IM 功能，即此参数设置为 true,即

```
GetILive()->init(SDKAppId, AccountType, true);
```

启用 IM 功能后，用户创建房间将会创建对应的 IM 群组，群组类型可以在创建房间参数中指定(iLiveRoomOption的groupType),一般使用 E_AVChatRoom_Group，即默认值；

创建房间的用户退出房间，将会解散对应的IM群组。

设置消息监听

开启 IM 功能情况下，要收到消息，需要设置接收消息的回调函数。

```
//接收消息回调
void OnMessageCallback(const Message &msg, void* data)
{
for (int i=0; i<msg.elems.size(); ++i)
{
MessageElem *pElem = msg.elems[i];
if (pElem->type == TEXT)
{
MessageTextElem *elem = dynamic_cast<MessageTextElem*>(pElem);
printf("recv msg from %s : %s\n", msg.sender.c_str(), elem->content.c_str());
}
}
}
GetILive()->setMessageCallBack(OnMessageCallback, NULL);
```

发送消息

要往群里发送消息，只需要调用 sendGroupMessage() 接口即可，例如要往群里发送文本消息，代码如下：

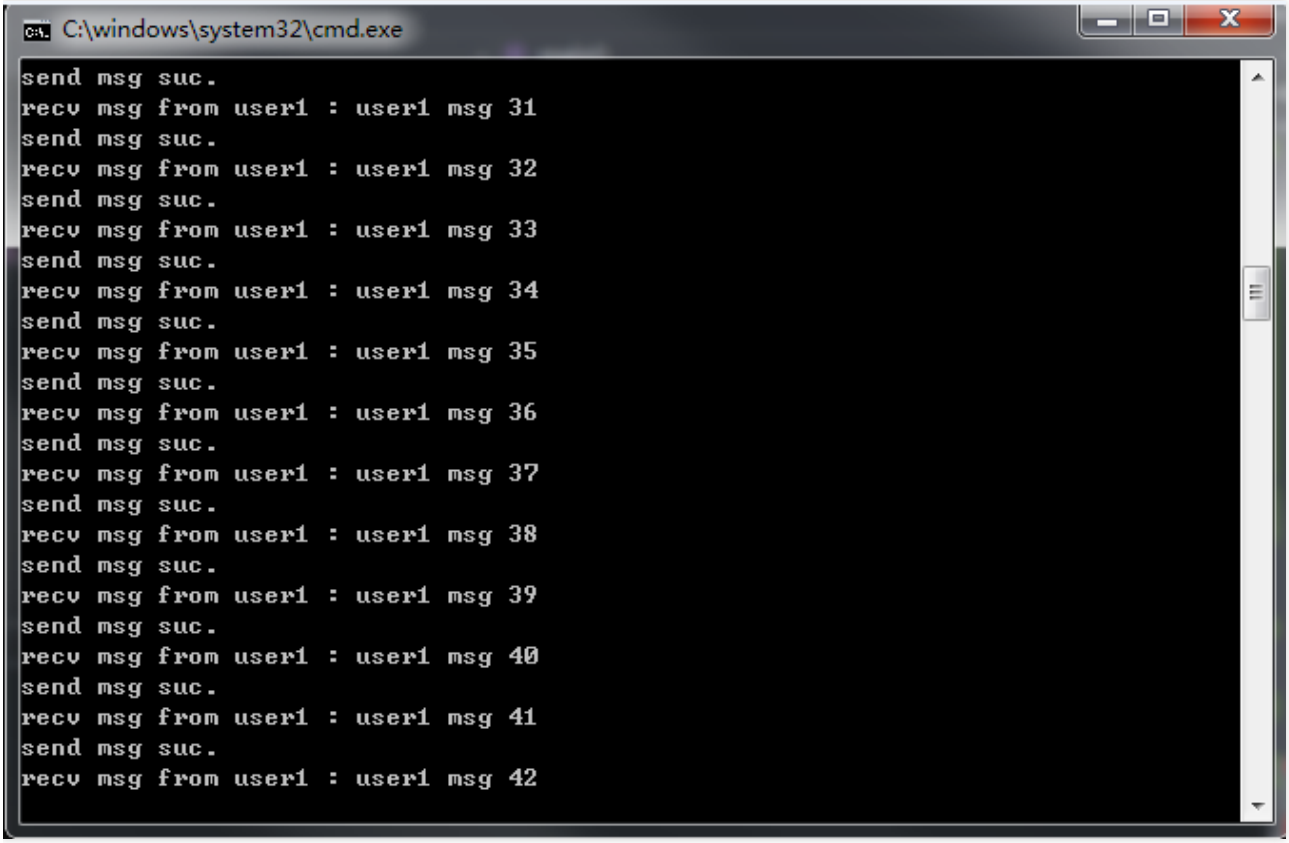
```
Message msg;
MessageTextElem *elem = new MessageTextElem("这里是文本消息内容");
msg.elems.push_back(elem);
GetILive()->sendGroupMessage(msg, [](void* data) {
//发送消息成功
}, [](const int code, const char* desc, void* data) {
//发送消息失败
}, NULL);
```

源码说明

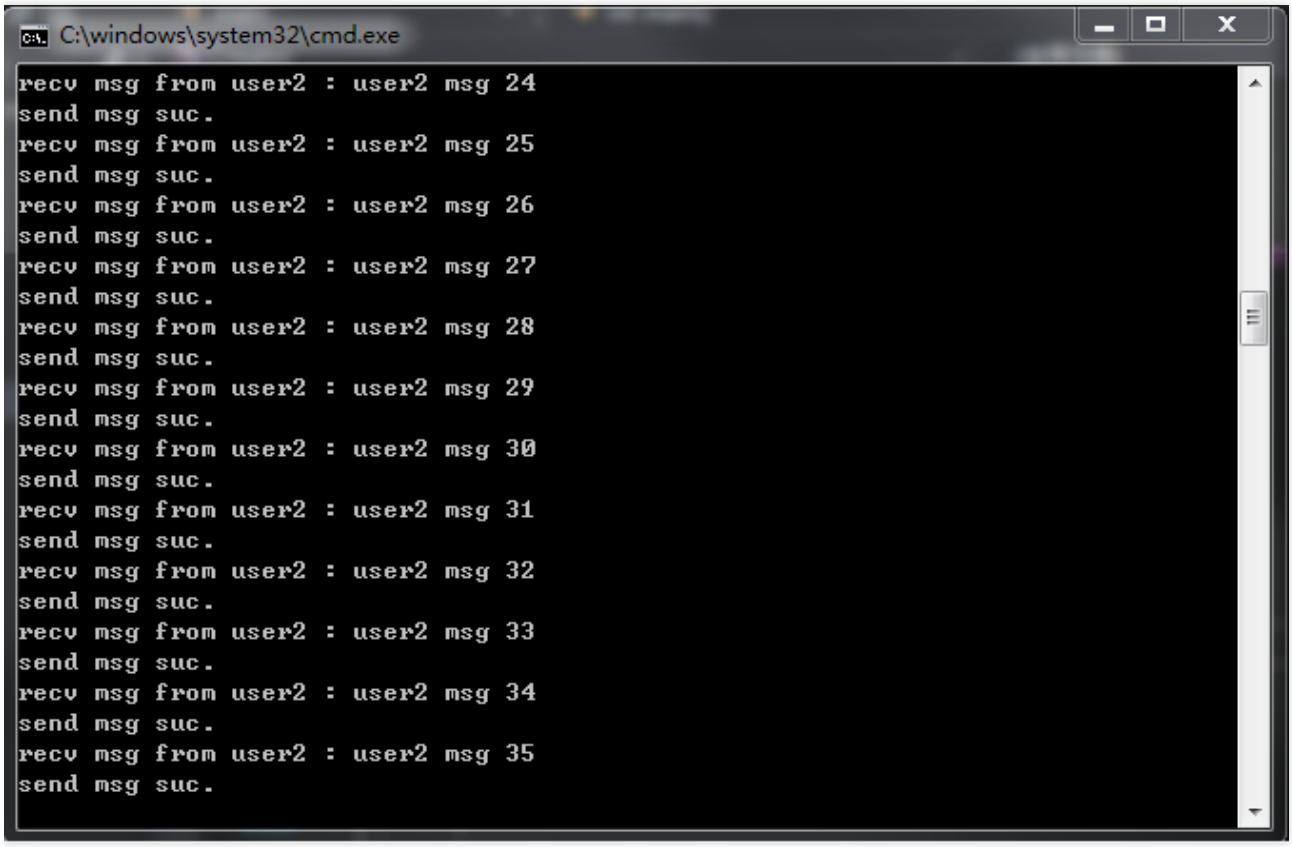
本例 Demo，新建了两个项目，一个创建房间(Creator)，一个加入房间(Joiner)；两个用户都开一个定时器，定时往群里发送消息，收到消息后，都将其打印输出到控制台。

版权所有：腾讯云计算（北京）有限责任公司

运行结果



版权所有：腾讯云计算（北京）有限责任公司



联系邮箱

如果对上述文档有不明白的地方，请反馈到trtcfb@qq.com

发送消息（Mac）

最近更新时间：2018-09-06 16:22:12

本文将指导您实现房间内的用户如何收发 IM 消息。

源码下载

在此我们提供以下所讲到的完整 Demo 代码，如有需要请您自行下载。
Demo 代码下载

相关概念

- 群组系统

名称	类	描述
文本消息	ILVLiveTextMessage	消息内容为一个字符串(String 类型)
自定义消息	ILVLiveCustomMessage	消息内容为一个 NSData 对象
其它消息	TIMMessage	IMSDK 消息类型 其它消息的封装

开启IM功能

首先我们得开启IM功能，修改创建和加入房间时的配置属性 imSupport 为YES，即开启了房间内的IM功能。

> 在配置imSupport为YES时，createRoom会自动创建IM群组，quitRoom时创建者会自动解散群组

```
option.imOption.imSupport = YES;
```

发送消息

房间内成员可以发送实时消息交流互动，这个功能是由 IMSDK 提供的，在 ILiveRoomManager 中对 IMSDK 的接口进行了封装，便于使用，先来看看如何发送消息。

版权所有：腾讯云计算（北京）有限责任公司

```
/**
发送C2C消息

@param dstUser 接收方ID
@param message IM消息
@param succ 发送成功回调
@param fail 发送失败回调
*/
- (void)sendC2CMessage:(NSString *)dstUser message:(TIMMessage *)message succ:(TCIVoidBlock)succ fail
ed:(TCIErrorBlock)fail;

/**
发送在线C2C消息

@param dstUser 接收方ID
@param message IM消息
@param succ 发送成功回调
@param fail 发送失败回调
*/
- (void)sendOnlineC2CMessage:(NSString *)dstUser message:(TIMMessage *)message succ:(TCIVoidBlock)s
ucc failed:(TCIErrorBlock)fail;

/**
发送Group消息
此处发送group，仅限于在当前直播间中发送group消息,或者绑定过IM群组id

@param message IM消息
@param succ 发送成功回调
@param fail 发送失败回调
*/
- (void)sendGroupMessage:(TIMMessage *)message succ:(TCIVoidBlock)succ failed:(TCIErrorBlock)fail;

/**
发送Group消息

@param message IM消息
@param succ 发送成功回调
@param fail 发送失败回调
*/
- (void)sendOnlineGroupMessage:(TIMMessage *)message succ:(TCIVoidBlock)succ failed:(TCIErrorBlock)fa
il;
```

注释：
这里可以看到消息分为群组类型和 C2C 类型（个人发给个人，可以理解为私聊），且分别提供了两种接口，在线消息和非在线消息，它们的区别就是，服务器不会存储在线消息，消息发出时，要是接收者不在线，下次接收者登录

后也不会收到发送给自己的在线消息，而非在线消息则会被服务器存储，如果发送时接收者不在线，在他下次登录时会再次发给他。

接口使用起来也比较方便，创建一个想要发送的消息类型对象，然后调用方法即可。
这里以发送一条群组文本消息为例（实时音视频的群组中一般都是此类消息）：

```
// 创建消息对象
TIMMessage *msg = [[TIMMessage alloc] init];
// 设置消息文本
TIMTextElem *elem = [[TIMTextElem alloc] init];
elem.text = textView.string;
[msg addElem:elem];
// 调用接口发送消息
__weak typeof(self) ws = self;
[[ILiveRoomManager getInstance] sendGroupMessage:msg succ:^{
NSLog(@"消息发送成功");
} failed:^(NSString *module, int errId, NSString *errMsg) {
NSLog(@"消息发送失败");
}];
```

接收消息

房间内所有的音视频事件都是通过音视频回调方法通知，IM 消息也是一样，房间内所有类型的消息都是通过各自的回调方法通知，只需要设置监听对象，然后实现相应的监听回调方法即可：

```
> TCLiveRoomWC.m

// 设置IM消息监听
[[[ILiveSDK getInstance] getTIMManager] addMessageListener:self];

// 实现代理协议方法
#pragma mark - TIMMessageListener
/**
* 新消息回调通知
*
* @param msgs 新消息列表，TIMMessage 类型数组
*/
- (void)onNewMessage:(NSArray *)msgs{
if (msgs.count <= 0){
return;
}
TIMMessage *msg = msgs[0];
```

```
if (msg.elemCount <= 0) {
return;
}
//是文本消息
if ([self isTextMsg:msg]) {
for(int i = 0; i < msg.elemCount; i++) {
TIMElem *elem = [msg getElem:i];
//消息解析展示
if([elem isKindOfClass:[TIMTextElem class]] && ((TIMTextElem *)elem).text){
NSString *showTextInfo = [NSString stringWithFormat:@"%@: %@",msg.sender, ((TIMTextElem *)elem).tex
t];
[self insertMessageToUI:showTextInfo];
break;
}
else if ([elem isKindOfClass:[TIMCustomElem class]]){
NSString *nick = msg.sender;
NSString *dataStr = [[NSString alloc] initWithData:((TIMCustomElem *)elem).data encoding:NSUTF8StringE
ncoding];
NSDictionary *descDic = [NSJSONSerialization JSONObjectWithData:[((TIMCustomElem *)elem).desc dataU
singEncoding:NSUTF8StringEncoding] options:NSJSONReadingAllowFragments error:nil];
NSString *nickNmae = descDic[@"nickName"];
if (nickNmae.length > 0) {
nick = nickNmae;
}
NSString *showTextInfo = [NSString stringWithFormat:@"%@: %@",nick, dataStr];
[self insertMessageToUI:showTextInfo];
break;
}
}
}
}
```

注释：
实现细节：在我们调用创建房间或者加入房间的接口时，方法内部会调用 IMSDK 的接口，将用户加入一个 IM 群组（ 即本节名词解释中介绍的实时音视频聊天室 ），用户发送的消息会转发给群组内所有人，也就是同一个房间内的所有人都能收到其他人发出的群组消息（ 自己发出的群组消息自己不会收到 ）。

UI 开发

收发消息时，房间内应该有一个公屏消息列表来展示这些消息，本文 Demo 只是简单实现了一个展示效果，您可以根据自己的需求自己实现。

常见问题

加入房间失败，错误模块 IMSDK，错误码 10010。

这表示要加入的 IM 群组不存在，需要检测是否先创建了群组(创建房间时传入的配置对象中 imsupport 为 true 时会自动创建群组)，并确认群组类型一致。

联系邮箱

如果对上述文档有不明白的地方，请反馈到trtcfb@qq.com

旁路直播与录制
旁路推流与录制（Android）

最近更新时间：2018-06-25 15:25:48

本文将指导您的客户端如何将房间中的视频通过旁路直播分享给别人，如何将房间中视频录制成视频文件。

相关概念

- 旁路直播

手动开启推流和录制

直接在房间模块中添加手动推流接口：

```
private void startPush(boolean bRecord){
ILivePushOption.RecordFileType recordFileType = bRecord ?
ILivePushOption.RecordFileType.RECORD_HLS_FLV_MP4 : ILivePushOption.RecordFileType.NONE
ILivePushOption option = new ILivePushOption()
.encode(ILivePushOption.Encode.HLS_AND_RTMP) // 旁路直播协议类型
.setRecordFileType(recordFileType) // 录制文件格式
//手动推流自动录制时，如果需要后台识别特定的录制文件，用户可以通过这个字段做区分。
// (使用这个字段时，控制台的“自动旁路直播”开关必须关闭)
.setRecordId(123456);
ILiveRoomManager.getInstance().startPushStream(option, new ILiveCallBack<ILivePushRes>() {
@Override
public void onSuccess(ILivePushRes data) {
if (null != data.getUrls()){
// 遍历推流类型及地址
for (ILivePushUrl url : data.getUrls()){
// 处理播放地址
}
}
}

@Override
public void onError(String module, int errCode, String errMsg) {
// 处理推流失败
}
});
}
```

注意：
这个接口可以同时开启旁路直播和录制（录制无法单独开启）。
此处的 encode 用于配置旁路的视频流类型，目前支持RTMP、HLS 和 FLV。
setRecordFileType 设置录制的视频文件格式，目前支持 HLS、FLV 和 MP4，纯音频可以录制为MP3格式。

停止手动推流和录制

```
private void stopPush(){
ILiveRoomManager.getInstance().stopPushStream(0, // 直播码模式下填0即可
new ILiveCallBack() {
@Override
public void onSuccess(Object data) {
// 停止成功
}

@Override
public void onError(String module, int errCode, String errMsg) {
// 停止失败
}
});
}
```

联系邮箱

如果对上述文档有不明白的地方，请反馈到trtcfb@qq.com

旁路推流和录制（iOS）

最近更新时间：2018-06-25 15:26:10

本文将指导您如何将房间中的视频通过旁路直播分享给别人，以及如何将房间中视频录制成视频文件。

相关概念

- 旁路直播

推流录制的 4 种方式

旁路直播开启方式	旁路直播录制方式	多录制格式	录制回调	开发便利程度	可靠性	资源消耗
自动	自动	FLV/HLS/MP4	☒	※※※※	※※※※	※
手动	自动	FLV/HLS/MP4	☒	※※※	※※※	※※
自动	手动	MP4	☒	※※	※※	※※※
手动	手动	MP4	☒	※	※	※※※※

自动旁路推流和自动旁路直播录制是最简单的方式，也是目前我们推荐的方式。

手动旁路直播和录制

在某些情况下，用户可能不想要所有视频都进行录制，可以通过 iLiveSDK 手动控制旁路直播和录制。

手动开启推流和录制

手动推流接口如下：

- 开始旁路直播

```
// 创建推流配置对象
ILiveChannelInfo *info = [[ILiveChannelInfo alloc] init];
info.channelName = [NSString stringWithFormat:@"前缀_%@",[[ILiveLoginManager getInstance] getLogi
```

```
nId]];
info.channelDesc = [NSString stringWithFormat:@"推流描述字符串"];

ILivePushOption *option = [[ILivePushOption alloc] init];
option.channelInfo = info;
option.encodeType = ILive_ENCODE_RTMP; //使用RTMP协议旁路直播
option.recrodFileType = ILive_RECORD_FILE_TYPE_MP4; //旁路直播时，如果需要自动录制，则填写自动录制时生成文件的格式，如果不需要自动录制，则不需要处理本字段
// 调用开始推流接口开始推流
[[ILiveRoomManager getInstance] startPushStream:option succ:^(id selfPtr) {
//旁路推流成功，返回的为`AVStreamerResp`类型的对象，其中包含了旁路直播成功的url信息;
} failed:^(NSString *module, int errId, NSString *errMsg) {
//旁路推流失败
}];
```

停止手动推流和录制

```
// 调用停止旁路推流接口，参数为要停止推流的频道ID数组，channelId在开启推流成功的回调中会返回
[[ILiveRoomManager getInstance] stopPushStreams:@[channelId] succ:^{
//停止旁路直播成功;
} failed:^(NSString *module, int errId, NSString *errMsg) {
//停止旁路直播失败;
}];
```

- 补充说明：
  如果开了自动旁路直播，打开摄像头就已经开始自动旁路直播了，调用此接口，只是获取旁路直播地址而已；结束推流接口会无效。

联系邮箱

如果对上述文档有不明白的地方，请反馈到trtcfb@qq.com

旁路推流和录制（PC）

最近更新时间：2018-06-25 15:26:36

本文将指导您如何将房间中的视频通过旁路直播分享给别人，以及如何将房间中视频录制成视频文件。

相关概念

- 旁路直播

手动旁路直播和录制

在某些情况下，用户可能不想要所有视频都进行录制，可以通过 iLiveSDK 手动控制旁路直播和录制。

手动开启推流和录制

手动录制的开始和结束分别调用 SDK 的 startPushStream() 和 stopPushStream() 接口，实例代码如下：

- 开始旁路直播

```
PushStreamOption pushOpt;
pushOpt.pushDataType = E_PushCamera; //旁路直播数据类型
pushOpt.encode = HLS_AND_RTMP; //旁路直播所用协议
pushOpt.recordFileType = RecordFile_HLS_FLV_MP4; //旁路直播时，录制视频文件格式
GetILive()->startPushStream( pushOpt, [](PushStreamRsp &value, void *data){
//旁路直播成功,在value参数中包含了旁路直播成功的url信息;
}, [](int code, const char * desc, void* data){
//旁路直播失败
}, NULL );
```

停止手动推流和录制

```
E_PushDataType pushDataType = E_PushCamera;
GetILive()->stopPushStream(0, pushDataType, [](void* data){
//停止旁路直播成功;
}, [](int code, const char *desc, void* data){
//停止旁路直播失败;
}, NULL); //第一个参数(channelId)是为了兼容老版本的频道模式的，现在此模式已废弃，直接填0即可;
```

- 补充说明：
  如果开了自动旁路直播，打开摄像头就已经开始自动旁路直播了，调用此接口，只是获取旁路直播地址而已；结束推流接口会无效。

联系邮箱

如果对上述文档有不明白的地方，请反馈到trtcfb@qq.com

旁路推流和录制（Web）

最近更新时间：2018-10-26 15:53:46

实时视频流的延时低，但成本高，如果您还有其他对延时要求不高的业务需求，可以考虑将实时视频通过旁路直播的方式分享给别人，并在此基础上完成录制，生成视频文件。

数据流通道可以理解成

```
实时音视频数据流 => 直播流 => 生成录制文件
```

相关概念

- 旁路直播

旁路直播与录制

Web端目前**不支持**手工旁路直播和手工录制，需要实现业务需求请参考下面的文档：

业务需求	方案
直播	需要开通自动旁路直播
混流	通过服务端接口调用混流接口
录制	通过服务端接口调用录制接口

版权所有：腾讯云计算（北京）有限责任公司